

Speech service documentation

Recognize speech, synthesize speech, get real-time translations, transcribe conversations, or integrate speech into your bot experiences.



OVERVIEW

[What is the Speech service?](#)

OVERVIEW

[About the Speech SDK](#)

WHAT'S NEW

[What's new](#)

REFERENCE

[Language support in the Speech service](#)

Speech-to-text

- [About speech-to-text](#)
- [Get started with speech-to-text](#)
- [Improve accuracy with Custom Speech](#)
- [Use batch transcription](#)

[More >](#)

Text-to-speech

- [About text-to-speech](#)
- [Get started with text-to-speech](#)
- [Improve synthesis with SSML](#)
- [Create a custom voice](#)
- [Make text-to-speech sound natural](#)

[More >](#)

Scenario deep-dives

- [Captioning with speech to text](#)
- [Call Center Overview](#)
- [Pronunciation assessment](#)

Tools

- [About Speech Studio](#)
- [About Speech CLI](#)

Speaker Recognition

- [About Speaker Recognition](#)
- [Get started with Speaker Recognition](#)

Speech translation

- [About speech translation](#)
- [Get started with speech translation](#)

Custom Keyword

-  [About Custom Keyword](#)
-  [Get started with Custom Keyword](#)

Intent recognition

-  [Recognize speech, intents, and entities](#)
-  [Recognize speech intents with LUIS](#)
-  [Recognize speech intents with simple pattern matching](#)
-  [Recognize speech intents with custom entity matching](#)
-  [Language Understanding \(LUIS\) portal ↗](#)

[More >](#)

Hosting

-  [Speech service containers](#)
-  [Speech service containers with Kubernetes](#)
-  [Speech service containers on Azure Container Instances](#)

[More >](#)

Migration

-  [Migrate to neural voice](#)
-  [Migrate to the v3.1 REST API](#)
-  [Migrate to Batch synthesis REST API](#)

Software development kits (SDKs)

Get started with the Speech SDK in your favorite programming language.



C#



C++



Java



JavaScript



iOS

Objective-C and Swift



Python

REST APIs

[Speech-to-text](#)
[Batch transcription](#)
[Text-to-speech](#)
[Conversation transcription ↗](#)

Resources

[Pricing ↗](#)
[Regions](#)
[Learn](#)

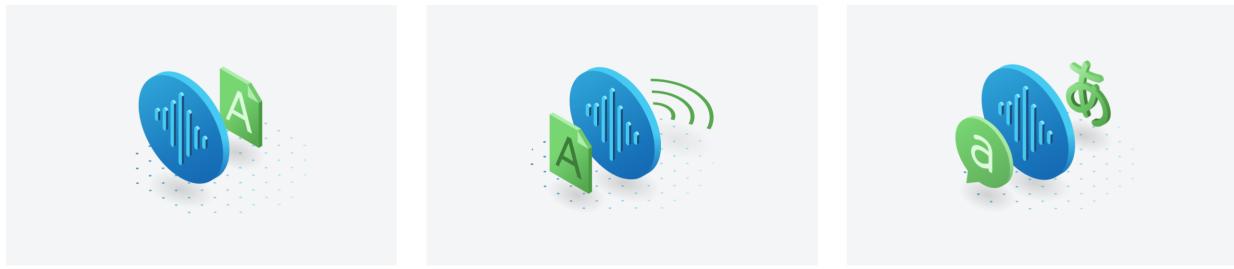
Support

[Support and help](#)
[GitHub issues ↗](#)

What is the Speech service?

Article • 01/13/2023 • 4 minutes to read

The Speech service provides speech-to-text and text-to-speech capabilities with an [Azure Speech resource](#). You can transcribe speech to text with high accuracy, produce natural-sounding text-to-speech voices, translate spoken audio, and use speaker recognition during conversations.



Create custom voices, add specific words to your base vocabulary, or build your own models. Run Speech anywhere, in the cloud or at the edge in containers. It's easy to speech enable your applications, tools, and devices with the [Speech CLI](#), [Speech SDK](#), [Speech Studio](#), or [REST APIs](#).

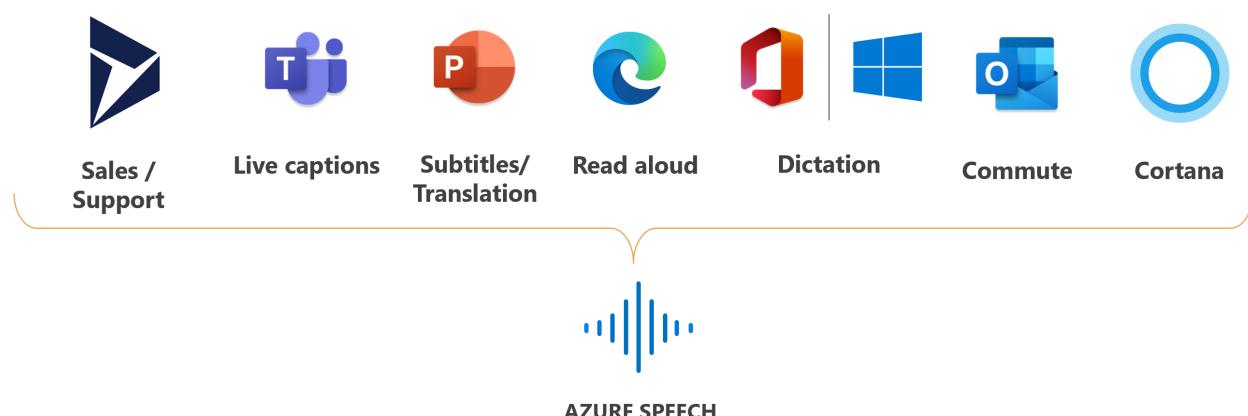
Speech is available for many [languages](#), [regions](#), and [price points](#) ↗.

Speech scenarios

Common scenarios for speech include:

- [Captioning](#): Learn how to synchronize captions with your input audio, apply profanity filters, get partial results, apply customizations, and identify spoken languages for multilingual scenarios.
- [Audio Content Creation](#): You can use neural voices 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.
- [Call Center](#): Transcribe calls in real-time or process a batch of calls, redact personally identifying information, and extract insights such as sentiment to help with your call center use case.
- [Voice assistants](#): Create natural, humanlike conversational interfaces for their applications and experiences. The voice assistant feature provides fast, reliable interaction between a device and an assistant implementation.

Microsoft uses Speech for many scenarios, such as captioning in Teams, dictation in Office 365, and Read Aloud in the Edge browser.



Speech capabilities

Speech feature summaries are provided below with links for more information.

Speech-to-text

Use [speech-to-text](#) to transcribe audio into text, either in real time or asynchronously.

Tip

You can try speech-to-text in [Speech Studio](#) without signing up or writing any code.

Convert audio to text from a range of sources, including microphones, audio files, and blob storage. Use speaker diarisation to determine who said what and when. Get readable transcripts with automatic formatting and punctuation.

The base model may not be sufficient if the audio contains ambient noise or includes a lot of industry and domain-specific jargon. In these cases, you can create and train [custom speech models](#) with acoustic, language, and pronunciation data. Custom speech models are private and can offer a competitive advantage.

Text-to-speech

With [text to speech](#), you can convert input text into humanlike synthesized speech. Use neural voices, which are humanlike voices powered by deep neural networks. Use the [Speech Synthesis Markup Language \(SSML\)](#) to fine-tune the pitch, pronunciation, speaking rate, volume, and more.

- Prebuilt neural voice: Highly natural out-of-the-box voices. Check the prebuilt neural voice samples the [Voice Gallery](#) and determine the right voice for your business needs.
- Custom neural voice: Besides the pre-built neural voices that come out of the box, you can also create a [custom neural voice](#) that is recognizable and unique to your brand or product. Custom neural voices are private and can offer a competitive advantage. Check the custom neural voice samples [here](#).

Speech translation

[Speech translation](#) enables real-time, multilingual translation of speech to your applications, tools, and devices. Use this feature for speech-to-speech and speech-to-text translation.

Language identification

[Language identification](#) is used to identify languages spoken in audio when compared against a list of [supported languages](#). Use language identification by itself, with speech-to-text recognition, or with speech translation.

Speaker recognition

[Speaker recognition](#) provides algorithms that verify and identify speakers by their unique voice characteristics. Speaker recognition is used to answer the question, "Who is speaking?".

Pronunciation assessment

[Pronunciation assessment](#) evaluates speech pronunciation and gives speakers feedback on the accuracy and fluency of spoken audio. With pronunciation assessment, language learners can practice, get instant feedback, and improve their pronunciation so that they can speak and present with confidence.

Intent recognition

[Intent recognition](#): Use speech-to-text with [Language Understanding \(LUIS\)](#) to derive user intents from transcribed speech and act on voice commands.

Delivery and presence

You can deploy Azure Cognitive Services Speech features in the cloud or on-premises.

With [containers](#), you can bring the service closer to your data for compliance, security, or other operational reasons.

Speech service deployment in sovereign clouds is available for some government entities and their partners. For example, the Azure Government cloud is available to US government entities and their partners. Azure China cloud is available to organizations with a business presence in China. For more information, see [sovereign clouds](#).



Use Speech in your application

The [Speech Studio](#) is a set of UI-based tools for building and integrating features from Azure Cognitive Services Speech service in your applications. You create projects in Speech Studio by using a no-code approach, and then reference those assets in your applications by using the [Speech SDK](#), the [Speech CLI](#), or the REST APIs.

The [Speech CLI](#) is a command-line tool for using Speech service without having to write any code. Most features in the Speech SDK are available in the Speech CLI, and some advanced features and customizations are simplified in the Speech CLI.

The [Speech SDK](#) exposes many of the Speech service capabilities you can use to develop speech-enabled applications. The Speech SDK is available in many programming languages and across all platforms.

In some cases, you can't or shouldn't use the [Speech SDK](#). In those cases, you can use REST APIs to access the Speech service. For example, use REST APIs for [batch transcription](#) and [speaker recognition](#) REST APIs.

Get started

We offer quickstarts in many popular programming languages. Each quickstart is designed to teach you basic design patterns and have you running code in less than 10 minutes. See the following list for the quickstart for each feature:

- [Speech-to-text quickstart](#)
- [Text-to-speech quickstart](#)
- [Speech translation quickstart](#)

Code samples

Sample code for the Speech service 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. 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 \(SDK\) ↗](#)

Next steps

- [Get started with speech-to-text](#)
 - [Get started with text-to-speech](#)
-

Additional resources

Documentation

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[Speech-to-text overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the speech-to-text feature of the Speech Service.

[Language identification - Speech service - Azure Cognitive Services](#)

Language identification is used to determine the language being spoken in audio when compared against a list of provided languages.

[Speech Studio overview - Speech service - Azure Cognitive Services](#)

Speech Studio is a set of UI-based tools for building and integrating features from Azure Speech service in your applications.

[How to use compressed input audio - Speech service - Azure Cognitive Services](#)

Learn how to use compressed input audio the Speech SDK and CLI.

[Keyword recognition overview - Speech service - Azure Cognitive Services](#)

An overview of the features, capabilities, and restrictions for keyword recognition by using the Speech Software Development Kit (SDK).

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Show 5 more](#)

What's new in Azure Cognitive Service for Speech?

Article • 02/01/2023 • 89 minutes to read

Azure Cognitive Service for Speech is updated on an ongoing basis. To stay up-to-date with recent developments, this article provides you with information about new releases and features.

Recent highlights

- Custom Speech-to-Text container disconnected mode was released in January 2023.
- Speech SDK 1.25.0 was released in January 2023.
- Text-to-speech Batch synthesis API is available in public preview.
- Speech-to-text REST API version 3.1 is generally available.
- Speech-to-text and text-to-speech container versions were updated in October 2022.
- TTS Service November 2022, several voices for `es-MX`, `it-IT`, and `pr-BR` locales were made generally available.

Release notes

Choose a service or resource

SDK

Speech SDK 1.25.0: January 2023 release

Breaking changes

- Language Identification (preview) APIs have been simplified. If you update to Speech SDK 1.25 and see a build break, please visit the [Language Identification \(preview\)](#) page to learn about the new property `SpeechServiceConnection_LanguageIdMode`. This single property replaces the two previous ones `SpeechServiceConnection_SingleLanguageIdPriority` and `SpeechServiceConnection_ContinuousLanguageIdPriority`. Prioritizing between

low latency and high accuracy is no longer necessary following recent model improvements. Now, you only need to select whether to run at-start or continuous Language Identification when doing continuous speech recognition or translation.

New features

- **C#/C++/Java:** Embedded Speech SDK is now released under gated public preview. See [Embedded Speech \(preview\)](#) documentation. You can now do on-device speech-to-text and text-to-speech when cloud connectivity is intermittent or unavailable. Supported on Android, Linux, MacOS and Windows platforms
- **C# MAUI:** Support added for iOS and Mac Catalyst targets in Speech SDK NuGet ([Customer issue ↗](#))
- **Unity:** Android x86_64 architecture added to Unity package ([Customer issue ↗](#))
- **Go:**
 - ALAW/MULAW direct streaming support added for speech recognition ([Customer issue ↗](#))
 - Added support for PhraseListGrammar. Thank you GitHub user [czkoko ↗](#) for the community contribution!
- **C#/C++:** Intent Recognizer now supports Conversational Language Understanding models in C++ and C# with orchestration on the Microsoft service

Bug fixes

- Fix an occasional hang in **KeywordRecognizer** when trying to stop it
- **Python:**
 - Fix for getting Pronunciation Assessment results when `PronunciationAssessmentGranularity.FullText` is set ([Customer issue ↗](#))
 - Fix for gender property for Male voices not being retrieved, when getting speech synthesis voices
- **JavaScript**
 - Fix for parsing some WAV files that were recorded on iOS devices ([Customer issue ↗](#))
 - JS SDK now builds without using npm-force-resolutions ([Customer issue ↗](#))
 - Conversation Translator now correctly sets service endpoint when using a `speechConfig` instance created using `SpeechConfig.fromEndpoint()`

Samples

- Added samples showing how to use Embedded Speech
- Added Speech-to-text sample for MAUI

See [Speech SDK samples repository](#).

Speech SDK 1.24.2: November 2022 release

New features

- No new features, just an embedded engine fix to support new model files.

Bug fixes

- All programming languages
 - Fixed an issue with encryption of embedded speech recognition models.

Speech SDK 1.24.1: November 2022 release

New features

- Published packages for the Embedded Speech preview. See <https://aka.ms/embedded-speech> for more information.

Bug fixes

- All programming languages
 - Fix embedded TTS crash when voice font isn't supported
 - Fix stopSpeaking() can't stop playback on Linux ([#1686](#))
- JavaScript SDK
 - Fixed regression in how conversation transcriber gated audio.
- Java
 - Temporarily Published updated POM and Javadocs files to Maven Central to enable the docs pipeline to update online reference docs.
- Python
 - Fix regression where Python speak_text(ssml) returns void.

Speech SDK 1.24.0: October 2022 release

New features

- **All programming languages:** AMR-WB (16kHz) added to the supported list of Text-to-speech audio output formats
- **Python:** Package added for Linux ARM64 for supported Linux distributions.
- **C#/C++/Java/Python:** Support added for ALAW & MULAW direct streaming to the speech service (in addition to existing PCM stream) using `AudioStreamWaveFormat`.
- **C# MAUI:** NuGet package updated to support Android targets for [.NET MAUI](#) developers ([Customer issue ↗](#))
- **Mac:** Added separate XCframework for Mac, which doesn't contain any iOS binaries. This offers an option for developers who need only Mac binaries using a smaller XCframework package.
- **Microsoft Audio Stack (MAS):**
 - When beam-forming angles are specified, sound originating outside of specified range will be suppressed better.
 - Approximately 70% reduction in the size of `libMicrosoft.CognitiveServices.Speech.extension.mas.so` for Linux ARM32 and Linux ARM64.
- **Intent Recognition using pattern matching:**
 - Add orthography support for the languages `fr`, `de`, `es`, `jp`
 - Added pre-built integer support for language `es`.

Bug fixes

- **iOS:** fix speech synthesis error on iOS 16 caused by compressed audio decoding failure ([Customer Issue ↗](#)).
- **JavaScript:**
 - Fix authentication token not working when getting speech synthesis voice list ([Customer issue ↗](#)).
 - Use data URL for worker loading ([Customer issue ↗](#)).
 - Create audio processor worklet only when AudioWorklet is supported in browser ([Customer issue ↗](#)). This was a community contribution by [William Wong ↗](#). Thank you William!
 - Fix recognized callback when LUIS response `connectionMessage` is empty ([Customer issue ↗](#)).
 - Properly set speech segmentation timeout.
- **Intent Recognition using pattern matching:**
 - Non-json characters inside models will now load properly.

- Fix hanging issue when `recognizeOnceAsync(text)` was called during continuous recognition.

Speech SDK 1.23.0: July 2022 release

New features

- **C#, C++, Java:** Added support for languages `zh-cn` and `zh-hk` in Intent Recognition with Pattern Matching.
- **C#:** Added support for `AnyCPU` .NET Framework builds

Bug fixes

- **Android:** Fixed OpenSSL vulnerability CVE-2022-2068 by updating OpenSSL to 1.1.1q
- **Python:** Fix crash when using PushAudioInputStream
- **iOS:** Fix "EXC_BAD_ACCESS: Attempted to dereference null pointer" as reported on iOS ([GitHub issue ↗](#))

Speech SDK 1.22.0: June 2022 release

New features

- **Java:** IntentRecognitionResult API for `getEntities()`, `applyLanguageModels()`, and `recognizeOnceAsync(text)` added to support the simple pattern matching engine.
- **Unity:** Added support for Mac M1 (Apple Silicon) for Unity package ([GitHub issue ↗](#))
- **C#:** Added support for x86_64 for Xamarin Android ([GitHub issue ↗](#))
- **C#:** .NET framework minimum version updated to v4.6.2 for SDK C# package as v4.6.1 has retired (see [Microsoft .NET Framework Component Lifecycle Policy](#))
- **Linux:** Added support for Debian 11 and Ubuntu 22.04 LTS. Ubuntu 22.04 LTS requires manual installation of libssl1.1 either as a binary package from [here ↗](#) (for example, `libssl1.1_1.1.1l-1ubuntu1.3_amd64.deb` or newer for x64), or by compiling from sources.

Bug fixes

- **UWP:** OpenSSL dependency removed from UWP libraries and replaced with WinRT websocket and HTTP APIs to meet security compliance and smaller binary footprint.
- **Mac:** Fixed "MicrosoftCognitiveServicesSpeech Module Not Found" issue when using Swift projects targeting macOS platform
- **Windows, Mac:** Fixed a platform-specific issue where audio sources that were configured via properties to stream at a real-time rate sometimes fell behind and eventually exceeded capacity

Samples ([GitHub ↗](#))

- **C#:** .NET framework samples updated to use v4.6.2
- **Unity:** Virtual-assistant sample fixed for Android and UWP
- **Unity:** Unity samples updated for Unity 2020 LTS version

Speech SDK 1.21.0: April 2022 release

New features

- **Java & JavaScript:** Added support for Continuous Language Identification when using the SpeechRecognizer object
- **JavaScript:** Added Diagnostics APIs to enable console logging level and (Node only) file logging, to help Microsoft troubleshoot customer-reported issues
- **Python:** Added support for Conversation Transcription
- **Go:** Added support for Speaker Recognition
- **C++ & C#:** Added support for a required group of words in the Intent Recognizer (simple pattern matching). For example: "(set|start|begin) a timer" where either "set", "start" or "begin" must be present for the intent to be recognized.
- **All programming languages, Speech Synthesis:** Added duration property in word boundary events. Added support for punctuation boundary and sentence boundary
- **Objective-C/Swift/Java:** Added word-level results on the Pronunciation Assessment result object (similar to C#). The application no longer needs to parse a JSON result string to get word-level information ([GitHub issue ↗](#))
- **iOS platform:** Added experimental support for ARMv7 architecture

Bug fixes

- **iOS platform:** Fix to allow building for the target "Any iOS Device", when using CocoaPod ([GitHub issue ↗](#))
- **Android platform:** OpenSSL version has been updated to 1.1.1n to fix security vulnerability [CVE-2022-0778 ↗](#)
- **JavaScript:** Fix issue where wav header wasn't updated with file size ([GitHub issue ↗](#))
- **JavaScript:** Fix request ID desync issue breaking translation scenarios ([GitHub issue ↗](#))
- **JavaScript:** Fix issue when instantiating SpeakerAudioDestination with no stream ([GitHub issue ↗](#))
- **C++:** Fix C++ headers to remove a warning when compiling for C++17 or newer

Samples [GitHub ↗](#)

- New **Java** samples for Speech Recognition with Language Identification
- New **Python** and **Java** samples for Conversation Transcription
- New **Go** sample for Speaker Recognition
- New **C++ and C#** tool for Windows that enumerates all audio capture and render devices, for the purpose of finding their Device ID. This ID is needed by the Speech SDK if you plan to [capture audio from, or render audio to, a non-default device](#).

Speech SDK 1.20.0: January 2022 release

New features

- **Objective-C, Swift, and Python:** Added support for DialogServiceConnector, used for [Voice-Assistant scenarios](#).
- **Python:** Support for Python 3.10 was added. Support for Python 3.6 was removed, per Python's [end-of-life for 3.6 ↗](#).
- **Unity:** Speech SDK is now supported for Unity applications on Linux.
- **C++, C#:** IntentRecognizer using pattern matching is now supported in C#. In addition, scenarios with custom entities, optional groups, and entity roles are now supported in C++ and C#.
- **C++, C#:** Improved diagnostics trace logging using new classes FileLogger, MemoryLogger, and EventLogger. SDK logs are an important tool for Microsoft to diagnose customer-reported issues. These new classes make it easier for customers to integrate Speech SDK logs into their own logging system.

- **All programming languages:** PronunciationAssessmentConfig now has properties to set the desired phoneme alphabet (IPA or SAPI) and N-Best Phoneme Count (avoiding the need to author a configuration JSON as per [GitHub issue 1284](#)). Also, syllable level output is now supported.
- **Android, iOS and MacOS (all programming languages):** GStreamer is no longer needed to support limited-bandwidth networks. SpeechSynthesizer now uses the operating system's audio decoding capabilities to decode compressed audio streamed from the text-to-speech service.
- **All programming languages:** SpeechSynthesizer now supports three new raw output Opus formats (without container), which are widely used in live streaming scenarios.
- **JavaScript:** Added getVoicesAsync() API to SpeechSynthesizer to retrieve the list of supported synthesis voices ([GitHub issue 1350](#))
- **JavaScript:** Added getWaveFormat() API to AudioStreamFormat to support non-PCM wave formats ([GitHub issue 452](#))
- **JavaScript:** Added volume getter/setter and mute()/unmute() APIs to SpeakerAudioDestination ([GitHub issue 463](#))

Bug fixes

- **C++, C#, Java, JavaScript, Objective-C, and Swift:** Fix to remove a 10-second delay while stopping a speech recognizer that uses a PushAudioInputStream. This is for the case where no new audio is pushed in after StopContinuousRecognition is called ([GitHub issues 1318](#), [331](#))
- **Unity on Android and UWP:** Unity meta files were fixed for UWP, Android ARM64, and Windows Subsystem for Android (WSA) ARM64 ([GitHub issue 1360](#))
- **iOS:** Compiling your Speech SDK application on any iOS Device when using CocoaPods is now fixed ([GitHub issue 1320](#))
- **iOS:** When SpeechSynthesizer is configured to output audio directly to a speaker, playback stopped at the beginning in rare conditions. This was fixed.
- **JavaScript:** Use script processor fallback for microphone input if no audio worklet is found ([GitHub issue 455](#))
- **JavaScript:** Add protocol to agent to mitigate bug found with Sentry integration ([GitHub issue 465](#))

Samples [GitHub](#)

- **C++**, **C#**, **Python**, and **Java** samples showing how to get detailed recognition results. The details include alternative recognition results,

confidence score, Lexical form, Normalized form, Masked Normalized form, with word-level timing for each.

- [iOS sample](#) added using AVFoundation as external audio source.
- [Java sample](#) added to show how to get SRT (SubRip Text) format using WordBoundary event.
- [Android samples](#) for Pronunciation Assessment.
- [C++](#), [C#](#) showing usage of the new Diagnostics Logging classes.

Speech SDK 1.19.0: 2021-Nov release

Highlights

- Speaker Recognition service is generally available (GA) now. Speech SDK APIs are available on C++, C#, Java, and JavaScript. With Speaker Recognition, you can accurately verify and identify speakers by their unique voice characteristics. For more information about this topic, see the [documentation](#).
- We've dropped support for Ubuntu 16.04 in conjunction with Azure DevOps and GitHub. Ubuntu 16.04 reached end of life back in April of 2021. Migrate your Ubuntu 16.04 workflows to Ubuntu 18.04 or newer.
- OpenSSL linking in Linux binaries changed to dynamic. Linux binary size has been reduced by about 50%.
- Mac M1 ARM-based silicon support added.

New features

- **C++/C#/Java:** New APIs added to enable audio processing support for speech input with Microsoft Audio Stack. Documentation [here](#).
- **C++:** New APIs for intent recognition to facilitate more advanced pattern matching. This includes List and Prebuilt Integer entities as well as support for grouping intents and entities as models (Documentation, updates, and samples are under development and will be published in the near future).
- **Mac:** Support for ARM64 (M1) based silicon for CocoaPod, Python, Java, and NuGet packages related to [GitHub issue 1244](#).
- **iOS/Mac:** iOS and macOS binaries are now packaged into xcframework related to [GitHub issue 919](#).
- **iOS/Mac:** Support for Mac catalyst related to [GitHub issue 1171](#).

- **Linux:** New tar package added for CentOS7 [About the Speech SDK](#). The Linux .tar package now contains specific libraries for RHEL/CentOS 7 in `lib/centos7-x64`. Speech SDK libraries in `lib/x64` are still applicable for all the other supported Linux x64 distributions (including RHEL/CentOS 8) and won't work on RHEL/CentOS 7.
- **JavaScript:** VoiceProfile & SpeakerRecognizer APIs made `async/awaitable`.
- **JavaScript:** Support added for US government Azure regions.
- **Windows:** Support added for playback on Universal Windows Platform (UWP).

Bug fixes

- **Android:** OpenSSL security update (updated to version 1.1.1l) for Android packages.
- **Python:** Resolved bug where selecting speaker device on Python fails.
- **Core:** Automatically reconnect when a connection attempt fails.
- **iOS:** Audio compression disabled on iOS packages due instability and bitcode build problems when using GStreamer. Details are available via [GitHub issue 1209 ↗](#).

Samples [GitHub ↗](#)

- **Mac/iOS:** Updated samples and quickstarts to use xcframework package.
- **.NET:** Samples updated to use .NET core 3.1 version.
- **JavaScript:** Added sample for Voice Assistants.

Speech SDK 1.18.0: 2021-July release

Note: Get started with the Speech SDK [here](#).

Highlights summary

- Ubuntu 16.04 reached end of life in April of 2021. With Azure DevOps and GitHub, we'll drop support for 16.04 in September 2021. Migrate ubuntu-16.04 workflows to ubuntu-18.04 or newer before then.

New features

- **C++:** Simple Language Pattern matching with the Intent Recognizer now makes it easier to [implement simple intent recognition scenarios](#).
- **C++/C#/Java:** We added a new API, `GetActivationPhrasesAsync()` to the `VoiceProfileClient` class for receiving a list of valid activation phrases in Speaker Recognition enrollment phase for independent recognition scenarios.
 - **Important:** The Speaker Recognition feature is in Preview. All voice profiles created in Preview will be discontinued 90 days after the Speaker Recognition feature is moved out of Preview into General Availability. At that point the Preview voice profiles will stop functioning.
- **Python:** Added [support for continuous Language Identification \(LID\)](#) on the existing `SpeechRecognizer` and `TranslationRecognizer` objects.
- **Python:** Added a [new Python object](#) named `SourceLanguageRecognizer` to do one-time or continuous LID (without recognition or translation).
- **JavaScript:** `getActivationPhrasesAsync` API added to `VoiceProfileClient` class for receiving a list of valid activation phrases in Speaker Recognition enrollment phase for independent recognition scenarios.
- **JavaScript** `VoiceProfileClient`'s `enrollProfileAsync` API is now `async` awaitable. See [this independent identification code](#), for example, usage.

Improvements

- **Java:** `AutoCloseable` support added to many Java objects. Now the try-with-resources model is supported to release resources. See [this sample that uses try-with-resources](#). Also see the Oracle Java documentation tutorial for [The try-with-resources Statement](#) to learn about this pattern.
- **Disk footprint** has been significantly reduced for many platforms and architectures. Examples for the `Microsoft.CognitiveServices.Speech.core` binary: x64 Linux is 475KB smaller (8.0% reduction); ARM64 Windows UWP is 464KB smaller (11.5% reduction); x86 Windows is 343KB smaller (17.5% reduction); and x64 Windows is 451KB smaller (19.4% reduction).

Bug fixes

- **Java:** Fixed synthesis error when the synthesis text contains surrogate characters. Details [here](#).
- **JavaScript:** Browser microphone audio processing now uses `AudioWorkletNode` instead of deprecated `ScriptProcessorNode`. Details [here](#).

- **JavaScript:** Correctly keep conversations alive during long running conversation translation scenarios. Details [here ↗](#).
- **JavaScript:** Fixed issue with recognizer reconnecting to a mediastream in continuous recognition. Details [here ↗](#).
- **JavaScript:** Fixed issue with recognizer reconnecting to a pushStream in continuous recognition. Details [here ↗](#).
- **JavaScript:** Corrected word level offset calculation in detailed recognition results. Details [here ↗](#).

Samples

- Java quickstart samples updated [here ↗](#).
- JavaScript Speaker Recognition samples updated to show new usage of `enrollProfileAsync()`. See samples [here ↗](#).

Speech SDK 1.17.0: 2021-May release

① Note

Get started with the Speech SDK [here](#).

Highlights summary

- Smaller footprint - we continue to decrease the memory and disk footprint of the Speech SDK and its components.
- A new stand-alone Language Identification API allows you to recognize what language is being spoken.
- Develop speech enabled mixed reality and gaming applications using Unity on macOS.
- You can now use Text-to-Speech in addition to speech recognition from the Go programming language.
- Several Bug fixes to address issues YOU, our valued customers, have flagged on GitHub! THANK YOU! Keep the feedback coming!

New features

- **C++/C#:** New stand-alone At-Start and Continuous Language Detection via the `SourceLanguageRecognizer` API. If you only want to detect the language(s) spoken in audio content, this is the API to do that. See details for [C++](#) and [C#](#).

- **C++/C#:** Speech Recognition and Translation Recognition now support both at-start and continuous Language Identification so you can programmatically determine which language(s) are being spoken before they're transcribed or translated. See documentation [here](#) for Speech Recognition and [here](#) for Speech Translation.
- **C#:** Added support Unity support to macOS (x64). This unlocks speech recognition and speech synthesis use cases in mixed reality and gaming!
- **Go:** We added support for speech synthesis/Text-to-Speech to the Go programming language to make speech synthesis available in even more use cases. See our [quickstart](#) or our [reference documentation ↗](#).
- **C++/C#/Java/Python/Objective-C/Go:** The speech synthesizer now supports the `connection` object. This helps you manage and monitor the connection to the Speech service, and is especially helpful to pre-connect to reduce latency. See documentation [here](#).
- **C++/C#/Java/Python/Objective-C/Go:** We now expose the latency and underrun time in `SpeechSynthesisResult` to help you monitor and diagnose speech synthesis latency issues. See details for [C++](#), [C#](#), [Java](#), [Python](#), [Objective-C](#) and [Go ↗](#).
- **C++/C#/Java/Python/Objective-C:** Text-to-Speech [now uses neural voices](#) by default when you don't specify a voice to be used. This gives you higher fidelity output by default, but also [increases the default price ↗](#). You can specify any of our [over 70 standard voices](#) or [over 130 neural voices](#) to change the default.
- **C++/C#/Java/Python/Objective-C/Go:** We added a Gender property to the synthesis voice info to make it easier to select voices based on gender. This addresses [GitHub issue #1055 ↗](#).
- **C++, C#, Java, JavaScript:** We now support `retrieveEnrollmentResultAsync`, `getAuthorizationPhrasesAsync`, and `getAllProfilesAsync()` in Speaker Recognition to ease user management of all voice profiles for a given account. See documentation for [C++](#), [C#](#), [Java](#), [JavaScript](#). This addresses [GitHub issue #338 ↗](#).
- **JavaScript:** We added retry for connection failures that will make your JavaScript-based speech applications more robust.

Improvements

- Linux and Android Speech SDK binaries have been updated to use the latest version of OpenSSL (1.1.1k)
- Code Size improvements:
 - Language Understanding is now split into a separate "lu" library.

- Windows x64 core binary size decreased by 14.4%.
- Android ARM64 core binary size decreased by 13.7%.
- other components also decreased in size.

Bug fixes

- All: Fixed [GitHub issue #842](#) for ServiceTimeout. You can now transcribe long audio files using the Speech SDK without the connection to the service terminating with this error. However, we still recommend you use [batch transcription](#) for long files.
- C#: Fixed [GitHub issue #947](#) where no speech input could leave your app in a bad state.
- Java: Fixed [GitHub Issue #997](#) where the Speech SDK for Java 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (for example, using CTRL+C on console app).
- Java: Added a fix to delete temporary files on Windows when using Speech SDK for Java.
- Java: Fixed [GitHub issue #994](#) where calling `DialogServiceConnector.stopListeningAsync` could result in an error.
- Java: Fixed a customer issue in the [virtual assistant quickstart](#).
- JavaScript: Fixed [GitHub issue #366](#) where `ConversationTranslator` threw an error 'this.cancelSpeech isn't a function'.
- JavaScript: Fixed [GitHub issue #298](#) where 'Get result as an in-memory stream' sample played sound out loud.
- JavaScript: Fixed [GitHub issue #350](#) where calling `AudioConfig` could result in a 'ReferenceError: MediaStream isn't defined'.
- JavaScript: Fixed an UnhandledPromiseRejection warning in Node.js for long-running sessions.

Samples

- Updated Unity samples documentation for macOS [here](#).
- A React Native sample for the Cognitive Services speech recognition service is now available [here](#).

Speech SDK 1.16.0: 2021-March release

(!) Note

The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019. Download it [here](#).

New features

- **C++/C#/Java/Python:** Moved to the latest version of GStreamer (1.18.3) to add support for transcribing any media format on Windows, Linux, and Android. See documentation [here](#).
- **C++/C#/Java/Objective-C/Python:** Added support for decoding compressed TTS/synthesized audio to the SDK. If you set output audio format to PCM and GStreamer is available on your system, the SDK will automatically request compressed audio from the service to save bandwidth and decode the audio on the client. You can set `SpeechServiceConnection_SynthEnableCompressedAudioTransmission` to `false` to disable this feature. Details for [C++](#), [C#](#), [Java](#), [Objective-C](#), [Python](#).
- **JavaScript:** Node.js users can now use the [AudioConfig.fromWavFileInput API](#). This addresses [GitHub issue #252](#).
- **C++/C#/Java/Objective-C/Python:** Added `GetVoicesAsync()` method for TTS to return all available synthesis voices. Details for [C++](#), [C#](#), [Java](#), [Objective-C](#), and [Python](#).
- **C++/C#/Java/JavaScript/Objective-C/Python:** Added `visemeReceived` event for TTS/speech synthesis to return synchronous viseme animation. See documentation [here](#).
- **C++/C#/Java/JavaScript/Objective-C/Python:** Added `BookmarkReached` event for TTS. You can set bookmarks in the input SSML and get the audio offsets for each bookmark. See documentation [here](#).
- **Java:** Added support for Speaker Recognition APIs. Details [here](#).
- **C++/C#/Java/JavaScript/Objective-C/Python:** Added two new output audio formats with WebM container for TTS (Webm16Khz16BitMonoOpus and Webm24Khz16BitMonoOpus). These are better formats for streaming audio with the Opus codec. Details for [C++](#), [C#](#), [Java](#), [JavaScript](#), [Objective-C](#), [Python](#).
- **C++/C#/Java:** Added support for retrieving voice profile for Speaker Recognition scenario. Details for [C++](#), [C#](#), and [Java](#).
- **C++/C#/Java/Objective-C/Python:** Added support for separate shared library for audio microphone and speaker control. This allows the developer to use the SDK in environments that don't have required audio library dependencies.

- **Objective-C/Swift:** Added support for module framework with umbrella header. This allows the developer to import Speech SDK as a module in iOS/Mac Objective-C/Swift apps. This addresses [GitHub issue #452 ↗](#).
- **Python:** Added support for [Python 3.9](#) and dropped support for Python 3.5 per Python's [end-of-life for 3.5 ↗](#).

Known issues

- **C++/C#/Java:** `DialogServiceConnector` can't use a `CustomCommandsConfig` to access a Custom Commands application and will instead encounter a connection error. This can be worked around by manually adding your application ID to the request with `config.SetServiceProperty("X-CommandsAppId", "your-application-id", ServicePropertyChannel.UriQueryParameter)`. The expected behavior of `CustomCommandsConfig` will be restored in the next release.

Improvements

- As part of our multi-release effort to reduce the Speech SDK's memory usage and disk footprint, Android binaries are now 3% to 5% smaller.
- Improved accuracy, readability, and see-also sections of our C# reference documentation [here](#).

Bug fixes

- **JavaScript:** Large WAV file headers are now parsed correctly (increases header slice to 512 bytes). This addresses [GitHub issue #962 ↗](#).
- **JavaScript:** Corrected microphone timing issue if mic stream ends before stop recognition, addressing an issue with Speech Recognition not working in Firefox.
- **JavaScript:** We now correctly handle initialization promise when the browser forces mic off before turnOn completes.
- **JavaScript:** We replaced URL dependency with url-parse. This addresses [GitHub issue #264 ↗](#).
- **Android:** Fixed callbacks not working when `minifyEnabled` is set to true.
- **C++/C#/Java/Objective-C/Python:** `TCP_NODELAY` will be correctly set to underlying socket IO for TTS to reduce latency.
- **C++/C#/Java/Python/Objective-C/Go:** Fixed an occasional crash when the recognizer was destroyed just after starting a recognition.
- **C++/C#/Java:** Fixed an occasional crash in the destruction of speaker recognizer.

Samples

- **JavaScript:** [Browser samples](#) no longer require separate JavaScript library file download.

Speech SDK 1.15.0: 2021-January release

ⓘ Note

The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019. Download it [here](#).

Highlights summary

- Smaller memory and disk footprint making the SDK more efficient.
- Higher fidelity output formats available for custom-neural voice private preview.
- Intent Recognizer can now get return more than the top intent, giving you the ability to make a separate assessment about your customer's intent.
- Voice assistants and bots are now easier to set up, and you can make it stop listening immediately, and exercise greater control over how it responds to errors.
- Improved on device performance through making compression optional.
- Use the Speech SDK on Windows ARM/ARM64.
- Improved low-level debugging.
- Pronunciation Assessment feature is now more widely available.
- Several Bug fixes to address issues YOU, our valued customers, have flagged on GitHub! THANK YOU! Keep the feedback coming!

Improvements

- The Speech SDK is now more efficient and lightweight. We've started a multi-release effort to reduce the Speech SDK's memory usage and disk footprint. As a first step we made significant file size reductions in shared libraries on most platforms. Compared to the 1.14 release:
 - 64-bit UWP-compatible Windows libraries are about 30% smaller.
 - 32-bit Windows libraries aren't yet seeing a size improvement.
 - Linux libraries are 20-25% smaller.
 - Android libraries are 3-5% smaller.

New features

- All: New 48 KHz output formats available for the private preview of custom-neural voice through the TTS speech synthesis API:
Audio48Khz192KBitRateMonoMp3, audio-48khz-192kbitrate-mono-mp3,
Audio48Khz96KBitRateMonoMp3, audio-48khz-96kbitrate-mono-mp3,
Raw48Khz16BitMonoPcm, raw-48khz-16bit-mono-pcm,
Riff48Khz16BitMonoPcm, riff-48khz-16bit-mono-pcm.
- All: Custom voice is also easier to use. Added support for setting custom voice via `EndpointId` ([C++](#), [C#](#), [Java](#), [JavaScript](#), [Objective-C](#), [Python](#)). Before this change, custom voice users needed to set the endpoint URL via the `FromEndpoint` method. Now customers can use the `FromSubscription` method just like prebuilt voices, and then provide the deployment ID by setting `EndpointId`. This simplifies setting up custom voices.
- C++/C#/Java/Objective-C/Python: Get more than the top intent from `IntentRecognizer`. It now supports configuring the JSON result containing all intents and not only the top scoring intent via `LanguageUnderstandingModel FromEndpoint` method by using `verbose=true` `uri` parameter. This addresses [GitHub issue #880](#). See updated documentation [here](#).
- C++/C#/Java: Make your voice assistant or bot stop listening immediately. `DialogServiceConnector` ([C++](#), [C#](#), [Java](#)) now has a `StopListeningAsync()` method to accompany `ListenOnceAsync()`. This will immediately stop audio capture and gracefully wait for a result, making it perfect for use with "stop now" button-press scenarios.
- C++/C#/Java/JavaScript: Make your voice assistant or bot react better to underlying system errors. `DialogServiceConnector` ([C++](#), [C#](#), [Java](#), [JavaScript](#)) now has a new `TurnStatusReceived` event handler. These optional events correspond to every `ITurnContext` resolution on the Bot and will report turn execution failures when they happen, for example, as a result of an unhandled exception, timeout, or network drop between Direct Line Speech and the bot. `TurnStatusReceived` makes it easier to respond to failure conditions. For example, if a bot takes too long on a backend database query (for example, looking up a product), `TurnStatusReceived` allows the client to know to reprompt with "sorry, I didn't quite get that, could you please try again" or something similar.
- C++/C#: Use the Speech SDK on more platforms. The [Speech SDK NuGet package](#) now supports Windows ARM/ARM64 desktop native binaries (UWP was already supported) to make the Speech SDK more useful on more machine types.

- **Java:** `DialogServiceConnector` now has a `setSpeechActivityTemplate()` method that was unintentionally excluded from the language previously. This is equivalent to setting the `Conversation_Speech_Activity_Template` property and will request that all future Bot Framework activities originated by the Direct Line Speech service merge the provided content into their JSON payloads.
- **Java:** Improved low-level debugging. The `Connection` class now has a `MessageReceived` event, similar to other programming languages (C++, C#). This event provides low-level access to incoming data from the service and can be useful for diagnostics and debugging.
- **JavaScript:** Easier setup for Voice Assistants and bots through `BotFrameworkConfig`, which now has `fromHost()` and `fromEndpoint()` factory methods that simplify the use of custom service locations versus manually setting properties. We also standardized optional specification of `botId` to use a non-default bot across the configuration factories.
- **JavaScript:** Improved on device performance through added string control property for websocket compression. For performance reasons, we disabled websocket compression by default. This can be reenabled for low-bandwidth scenarios. More details [here](#). This addresses [GitHub issue #242](#).
- **JavaScript:** Added support for IPronunciation Assessment to enable evaluation of speech pronunciation. See the quickstart [here](#).

Bug fixes

- **All** (except JavaScript): Fixed a regression in version 1.14, in which too much memory was allocated by the recognizer.
- **C++:** Fixed a garbage collection issue with `DialogServiceConnector`, addressing [GitHub issue #794](#).
- **C#:** Fixed an issue with thread shutdown that caused objects to block for about a second when disposed.
- **C++/C#/Java:** Fixed an exception preventing an application from setting speech authorization token or activity template more than once on a `DialogServiceConnector`.
- **C++/C#/Java:** Fixed a recognizer crash due to a race condition in teardown.
- **JavaScript:** `DialogServiceConnector` didn't previously honor the optional `botId` parameter specified in `BotFrameworkConfig`'s factories. This made it necessary to set the `botId` query string parameter manually to use a non-default bot. The bug has been corrected and `botId` values provided to `BotFrameworkConfig`'s factories will be honored and used, including the new

`fromHost()` and `fromEndpoint()` additions. This also applies to the `applicationId` parameter for `CustomCommandsConfig`.

- **JavaScript:** Fixed [GitHub issue #881](#), allowing recognizer object reusage.
- **JavaScript:** Fixed an issue where the SKD was sending `speech.config` multiple times in one TTS session, wasting bandwidth.
- **JavaScript:** Simplified error handling on microphone authorization, allowing more descriptive message to bubble up when user hasn't allowed microphone input on their browser.
- **JavaScript:** Fixed [GitHub issue #249](#) where type errors in `ConversationTranslator` and `ConversationTranscriber` caused a compilation error for TypeScript users.
- **Objective-C:** Fixed an issue where GStreamer build failed for iOS on Xcode 11.4, addressing [GitHub issue #911](#).
- **Python:** Fixed [GitHub issue #870](#), removing "DeprecationWarning: the `imp` module is deprecated in favor of `importlib`".

Samples

- [From-file sample for JavaScript browser](#) now uses files for speech recognition. This addresses [GitHub issue #884](#).

Speech SDK 1.14.0: 2020-October release

ⓘ Note

The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019. Download it [here](#).

New features

- **Linux:** Added support for Debian 10 and Ubuntu 20.04 LTS.
- **Python/Objective-C:** Added support for the `KeywordRecognizer` API. Documentation will be [here](#).
- **C++/Java/C#:** Added support to set any `HTTPHeader` key/value via `ServicePropertyChannel::HTTPHeader`.
- **JavaScript:** Added support for the `ConversationTranscriber` API. Read documentation [here](#).
- **C++/C#:** Added new `AudioDataStream FromWavFileInput` method (to read .WAV files) [here \(C++\)](#) and [here \(C#\)](#).

- **C++/C#/Java/Python/Objective-C/Swift:** Added a `stopSpeakingAsync()` method to stop Text-to-Speech synthesis. Read the Reference documentation [here \(C++\)](#), [here \(C#\)](#), [here \(Java\)](#), [here \(Python\)](#), and [here \(Objective-C/Swift\)](#).
- **C#, C++, Java:** Added a `FromDialogServiceConnector()` function to the `Connection` class that can be used to monitor connection and disconnection events for `DialogServiceConnector`. Read the Reference documentation [here \(C#\)](#), [here \(C++\)](#), and [here \(Java\)](#).
- **C++/C#/Java/Python/Objective-C/Swift:** Added support for Pronunciation Assessment, which evaluates speech pronunciation and gives speakers feedback on the accuracy and fluency of spoken audio. Read the documentation [here](#).

Breaking change

- **JavaScript:** `PullAudioOutputStream.read()` has a return type change from an internal Promise to a Native JavaScript Promise.

Bug fixes

- **All:** Fixed 1.13 regression in `SetServiceProperty` where values with certain special characters were ignored.
- **C#:** Fixed Windows console samples on Visual Studio 2019 failing to find native DLLs.
- **C#:** Fixed crash with memory management if stream is used as `KeywordRecognizer` input.
- **ObjectiveC/Swift:** Fixed crash with memory management if stream is used as recognizer input.
- **Windows:** Fixed coexistence issue with BT HFP/A2DP on UWP.
- **JavaScript:** Fixed mapping of session IDs to improve logging and aid in internal debug/service correlations.
- **JavaScript:** Added fix for `DialogServiceConnector` disabling `ListenOnce` calls after the first call is made.
- **JavaScript:** Fixed issue where result output would only ever be "simple".
- **JavaScript:** Fixed continuous recognition issue in Safari on macOS.
- **JavaScript:** CPU load mitigation for high request throughput scenario.
- **JavaScript:** Allow access to details of Voice Profile Enrollment result.
- **JavaScript:** Added fix for continuous recognition in `IntentRecognizer`.
- **C++/C#/Java/Python/Swift/ObjectiveC:** Fixed incorrect url for australiaeast and brazilsouth in `IntentRecognizer`.

- **C++/C#**: Added `VoiceProfileType` as an argument when creating a `VoiceProfile` object.
- **C++/C#/Java/Python/Swift/ObjectiveC**: Fixed potential `SPX_INVALID_ARG` when trying to read `AudioDataStream` from a given position.
- **iOS**: Fixed crash with speech recognition on Unity

Samples

- **ObjectiveC**: Added sample for keyword recognition [here ↗](#).
- **C#/JavaScript**: Added quickstart for conversation transcription [here \(C#\) ↗](#) and [here \(JavaScript\) ↗](#).
- **C++/C#/Java/Python/Swift/ObjectiveC**: Added sample for Pronunciation Assessment [here ↗](#)
- **Xamarin**: Updated quickstart to latest Visual Studio template [here ↗](#).

Known Issue

- DigiCert Global Root G2 certificate isn't supported by default in HoloLens 2 and Android 4.4 (KitKat) and needs to be added to the system to make the Speech SDK functional. The certificate will be added to HoloLens 2 OS images in the near future. Android 4.4 customers need to add the updated the certificate to the system.

COVID-19 abridged testing

Due to working remotely over the last few weeks, we couldn't do as much manual verification testing as we normally do. We haven't made any changes we think could have broken anything, and our automated tests all passed. In the unlikely event that we missed something, please let us know on [GitHub ↗](#).
Stay healthy!

Speech SDK 1.13.0: 2020-July release

ⓘ Note

The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019. Download and install it from [here ↗](#).

New features

- **C#**: Added support for asynchronous conversation transcription. See documentation [here](#).
- **JavaScript**: Added Speaker Recognition support for both [browser](#) and [Node.js](#).
- **JavaScript**: Added support for Language Identification/language ID. See documentation [here](#).
- **Objective-C**: Added support for [multi-device conversation](#) and [conversation transcription](#).
- **Python**: Added compressed audio support for Python on Windows and Linux. See documentation [here](#).

Bug fixes

- **All**: Fixed an issue that caused the KeywordRecognizer to not move forward the streams after a recognition.
- **All**: Fixed an issue that caused the stream obtained from a KeywordRecognitionResult to not contain the keyword.
- **All**: Fixed an issue that the SendMessageAsync doesn't really send the message over the wire after the users finish waiting for it.
- **All**: Fixed a crash in Speaker Recognition APIs when users call VoiceProfileClient::SpeakerRecEnrollProfileAsync method multiple times and didn't wait for the calls to finish.
- **All**: Fixed enable file logging in VoiceProfileClient and SpeakerRecognizer classes.
- **JavaScript**: Fixed an [issue](#) with throttling when browser is minimized.
- **JavaScript**: Fixed an [issue](#) with a memory leak on streams.
- **JavaScript**: Added caching for OCSP responses from NodeJS.
- **Java**: Fixed an issue that was causing BigInteger fields to always return 0.
- **iOS**: Fixed an [issue](#) with publishing Speech SDK-based apps in the iOS App Store.

Samples

- **C++**: Added sample code for Speaker Recognition [here](#).

COVID-19 abridged testing

Due to working remotely over the last few weeks, we couldn't do as much manual verification testing as we normally do. We haven't made any changes we think

could have broken anything, and our automated tests all passed. In the unlikely event that we missed something, please let us know on [GitHub ↗](#). Stay healthy!

Speech SDK 1.12.1: 2020-June release

New features

- **C#, C++:** Speaker Recognition Preview: This feature enables speaker identification (who is speaking?) and speaker verification (is the speaker who they claim to be?). Start with an [overview](#), read the [Speaker Recognition basics article](#), or the [API reference docs](#).

Bug fixes

- **C#, C++:** Fixed microphone recording wasn't working in 1.12 in Speaker Recognition.
- **JavaScript:** Fixes for Text-To-Speech in Firefox, and Safari on macOS and iOS.
- Fix for Windows application verifier access violation crash on conversation transcription when using eight-channel stream.
- Fix for Windows application verifier access violation crash on multi-device conversation translation.

Samples

- **C#:** [Code sample ↗](#) for Speaker Recognition.
- **C++:** [Code sample ↗](#) for Speaker Recognition.
- **Java:** [Code sample ↗](#) for intent recognition on Android.

COVID-19 abridged testing

Due to working remotely over the last few weeks, we couldn't do as much manual verification testing as we normally do. We haven't made any changes we think could have broken anything, and our automated tests all passed. In the unlikely event that we missed something, please let us know on [GitHub ↗](#).

Stay healthy!

Speech SDK 1.12.0: 2020-May release

New features

- **Go:** New Go language support for [Speech Recognition](#) and [custom voice assistant](#). Set up your dev environment [here](#). For sample code, see the Samples section below.
- **JavaScript:** Added Browser support for Text-To-Speech. See documentation [here](#).
- **C++, C#, Java:** New `KeywordRecognizer` object and APIs supported on Windows, Android, Linux & iOS platforms. Read the documentation [here](#). For sample code, see the Samples section below.
- **Java:** Added multi-device conversation with translation support. See the reference doc [here](#).

Improvements & Optimizations

- **JavaScript:** Optimized browser microphone implementation improving speech recognition accuracy.
- **Java:** Refactored bindings using direct JNI implementation without SWIG. This change reduces by 10x the bindings size for all Java packages used for Windows, Android, Linux, and Mac and eases further development of the Speech SDK Java implementation.
- **Linux:** Updated support [documentation](#) with the latest RHEL 7 specific notes.
- Improved connection logic to attempt connecting multiple times when service and network errors occur.
- Updated the [portal.azure.com](#)  Speech Quickstart page to help developers take the next step in the Azure Speech journey.

Bug fixes

- **C#, Java:** Fixed an [issue](#)  with loading SDK libraries on Linux ARM (both 32 bit and 64 bit).
- **C#:** Fixed explicit disposal of native handles for TranslationRecognizer, IntentRecognizer, and Connection objects.
- **C#:** Fixed audio input lifetime management for ConversationTranscriber object.
- Fixed an issue where `IntentRecognizer` result reason wasn't set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset wasn't set correctly.
- Fixed a race condition where SDK was trying to send a network message before opening the websocket connection. Was reproducible for

`TranslationRecognizer` while adding participants.

- Fixed memory leaks in the keyword recognizer engine.

Samples

- **Go:** Added quickstarts for [speech recognition](#) and [custom voice assistant](#). Find sample code [here ↗](#).
- **JavaScript:** Added quickstarts for [Text-to-Speech](#), [Translation](#), and [Intent Recognition](#).
- Keyword recognition samples for [C# ↗](#) and [Java ↗](#) (Android).

COVID-19 abridged testing

Due to working remotely over the last few weeks, we couldn't do as much manual verification testing as we normally do. We haven't made any changes we think could have broken anything, and our automated tests all passed. If we missed something, please let us know on [GitHub ↗](#).

Stay healthy!

Speech SDK 1.11.0: 2020-March release

New features

- Linux: Added support for Red Hat Enterprise Linux (RHEL)/CentOS 7 x64 with [instructions](#) on how to configure the system for Speech SDK.
- Linux: Added support for .NET Core C# on Linux ARM32 and ARM64. Read more [here](#).
- C#, C++: Added `utteranceId` in `ConversationTranscriptionResult`, a consistent ID across all the intermediates and final speech recognition result. Details for [C#](#), [C++](#).
- Python: Added support for `Language ID`. See `speech_sample.py` in [GitHub repo ↗](#).
- Windows: Added compressed audio input format support on Windows platform for all the win32 console applications. Details [here](#).
- JavaScript: Support speech synthesis (Text-to-Speech) in NodeJS. Learn more [here ↗](#).
- JavaScript: Add new APIs to enable inspection of all send and received messages. Learn more [here ↗](#).

Bug fixes

- C#, C++: Fixed an issue so `SendMessageAsync` now sends binary message as binary type. Details for [C#, C++](#).
- C#, C++: Fixed an issue where using `Connection MessageReceived` event may cause crash if `Recognizer` is disposed before `Connection` object. Details for [C#, C++](#).
- Android: Audio buffer size from microphone decreased from 800 ms to 100 ms to improve latency.
- Android: Fixed an [issue ↗](#) with x86 Android emulator in Android Studio.
- JavaScript: Added support for Regions in China with the `fromSubscription` API. Details [here](#).
- JavaScript: Add more error information for connection failures from NodeJS.

Samples

- Unity: Intent recognition public sample is fixed, where LUIS json import was failing. Details [here ↗](#).
- Python: Sample added for `Language ID`. Details [here ↗](#).

Covid19 abridged testing: Due to working remotely over the last few weeks, we couldn't do as much manual device verification testing as we normally do. For example, we couldn't test microphone input and speaker output on Linux, iOS, and macOS. We haven't made any changes we think could have broken anything on these platforms, and our automated tests all passed. In the unlikely event that we missed something, let us know on [GitHub ↗](#).

Thank you for your continued support. As always, please post questions or feedback on [GitHub ↗](#) or [Stack Overflow ↗](#).

Stay healthy!

Speech SDK 1.10.0: 2020-February release

New features

- Added Python packages to support the new 3.8 release of Python.
- Red Hat Enterprise Linux (RHEL)/CentOS 8 x64 support (C++, C#, Java, Python).

 **Note**

Customers must configure OpenSSL according to these instructions.

- Linux ARM32 support for Debian and Ubuntu.
- DialogServiceConnector now supports an optional "bot ID" parameter on BotFrameworkConfig. This parameter allows the use of multiple Direct Line Speech bots with a single Azure speech resource. Without the parameter specified, the default bot (as determined by the Direct Line Speech channel configuration page) will be used.
- DialogServiceConnector now has a SpeechActivityTemplate property. The contents of this JSON string will be used by Direct Line Speech to pre-populate a wide variety of supported fields in all activities that reach a Direct Line Speech bot, including activities automatically generated in response to events like speech recognition.
- TTS now uses subscription key for authentication, reducing the first byte latency of the first synthesis result after creating a synthesizer.
- Updated speech recognition models for 19 locales for an average word error rate reduction of 18.6% (es-ES, es-MX, fr-CA, fr-FR, it-IT, ja-JP, ko-KR, pt-BR, zh-CN, zh-HK, nb-NO, fi-FL, ru-RU, pl-PL, ca-ES, zh-TW, th-TH, pt-PT, tr-TR). The new models bring significant improvements across multiple domains including Dictation, Call-Center Transcription, and Video Indexing scenarios.

Bug fixes

- Fixed bug where Conversation Transcriber didn't await properly in JAVA APIs
- Android x86 emulator fix for Xamarin [GitHub issue](#)
- Add missing (Get|Set)Property methods to AudioConfig
- Fix a TTS bug where the audioDataStream couldn't be stopped when connection fails
- Using an endpoint without a region would cause USP failures for conversation translator
- ID generation in Universal Windows Applications now uses an appropriately unique GUID algorithm; it previously and unintentionally defaulted to a stubbed implementation that often produced collisions over large sets of interactions.

Samples

- Unity sample for using Speech SDK with [Unity microphone and push mode streaming](#)

Other changes

- OpenSSL configuration documentation updated for Linux

Speech SDK 1.9.0: 2020-January release

New features

- Multi-device conversation: connect multiple devices to the same speech or text-based conversation, and optionally translate messages sent between them. Learn more in [this article](#).
- Keyword recognition support added for Android `.aar` package and added support for x86 and x64 flavors.
- Objective-C: `SendMessage` and `SetMessageProperty` methods added to `Connection` object. See documentation [here](#).
- TTS C++ api now supports `std::wstring` as synthesis text input, removing the need to convert a wstring to string before passing it to the SDK. See details [here](#).
- C#: `Language ID` and `source language config` are now available.
- JavaScript: Added a feature to `Connection` object to pass through custom messages from the Speech service as callback `receivedServiceMessage`.
- JavaScript: Added support for `FromHost API` to ease use with on-prem containers and sovereign clouds. See documentation [here](#).
- JavaScript: We now honor `NODE_TLS_REJECT_UNAUTHORIZED` thanks to a contribution from [orgads](#). See details [here](#).

Breaking changes

- `OpenSSL` has been updated to version 1.1.1b and is statically linked to the Speech SDK core library for Linux. This may cause a break if your inbox `OpenSSL` hasn't been installed to the `/usr/lib/ssl` directory in the system. Check [our documentation](#) under Speech SDK docs to work around the issue.
- We've changed the data type returned for C# `WordLevelTimingResult.Offset` from `int` to `long` to allow for access to `WordLevelTimingResults` when speech data is longer than 2 minutes.
- `PushAudioInputStream` and `PullAudioInputStream` now send wav header information to the Speech service based on `AudioStreamFormat`, optionally specified when they were created. Customers must now use the [supported audio input format](#). Any other formats will get suboptimal recognition results or may cause other issues.

Bug fixes

- See the `OpenSSL` update under Breaking changes above. We fixed both an intermittent crash and a performance issue (lock contention under high load) in Linux and Java.
- Java: Made improvements to object closure in high concurrency scenarios.
- Restructured our NuGet package. We removed the three copies of `Microsoft.CognitiveServices.Speech.core.dll` and `Microsoft.CognitiveServices.Speech.extension.kws.dll` under lib folders, making the NuGet package smaller and faster to download, and we added headers needed to compile some C++ native apps.
- Fixed quickstart samples [here](#). These were exiting without displaying "microphone not found" exception on Linux, macOS, Windows.
- Fixed SDK crash with long speech recognition results on certain code paths like [this sample](#).
- Fixed SDK deployment error in Azure Web App environment to address [this customer issue](#).
- Fixed a TTS error while using multi `<voice>` tag or `<audio>` tag to address [this customer issue](#).
- Fixed a TTS 401 error when the SDK is recovered from suspended.
- JavaScript: Fixed a circular import of audio data thanks to a contribution from [euirim](#).
- JavaScript: added support for setting service properties, as added in 1.7.
- JavaScript: fixed an issue where a connection error could result in continuous, unsuccessful websocket reconnect attempts.

Samples

- Added keyword recognition sample for Android [here](#).
- Added TTS sample for the server scenario [here](#).
- Added Multi-device conversation quickstarts for C# and C++ [here](#).

Other changes

- Optimized SDK core library size on Android.
- SDK in 1.9.0 and onwards supports both `int` and `string` types in the voice signature version field for Conversation Transcriber.

Speech SDK 1.8.0: 2019-November release

New features

- Added a `FromHost()` API, to ease use with on-premises containers and sovereign clouds.
- Added Source Language Identification 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`.
- Parts 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 Identification.

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 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 isn't 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 isn't 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's 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 couldn't 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 isn't affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input isn't 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 isn't 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 weren't handled correctly. With this release, if you set proxy username and proxy password to an empty string, they won't be submitted when connecting to the proxy.
- SessionId's created by the SDK weren't 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 didn't 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 isn't 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 haven't 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 isn't 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 when unset properties occur.
- 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 that UWP apps built with the Speech SDK don't 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.
- Fire 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.

Additional resources

Documentation

[azure.cognitiveservices.speech.SpeechConfig class](#)

Class that defines configurations for speech / intent recognition and speech synthesis. The configuration can be initialized in different ways: from subscription: pass a subscription key and a region from endpoint: pass an endpoint. Subscription key or authorization token are optional. from...

[azure.cognitiveservices.speech package](#)

Microsoft Speech SDK for Python

[Configure the Speech CLI datastore - Speech service - Azure Cognitive Services](#)

Learn how to configure the Speech CLI datastore.

[Speech SDK audio input stream concepts - Azure Cognitive Services](#)

An overview of the capabilities of the Speech SDK audio input stream API.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[microsoft-cognitiveservices-speech-sdk package](#)

[Get batch transcription results - Speech service - Azure Cognitive Services](#)

With batch transcription, the Speech service transcribes the audio data and stores the results in a storage container. You can then retrieve the results from the storage container.

[SpeechRecognizer.StartContinuousRecognitionAsync Method \(Microsoft.CognitiveServices.Speech\) - Azure for .NET Developers](#)

Starts speech recognition on a continuous audio stream as an asynchronous operation, until StopContinuousRecognitionAsync() is called. You must subscribe to events to receive recognition results.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

Language and voice support for the Speech service

Article • 02/01/2023 • 28 minutes to read

The following tables summarize language support for [speech-to-text](#), [text-to-speech](#), [pronunciation assessment](#), [speech translation](#), [speaker recognition](#), and additional service features.

You can also get a list of locales and voices supported for each specific region or endpoint through the [Speech SDK](#), [Speech-to-text REST API](#), [Speech-to-text REST API for short audio](#) and [Text-to-speech REST API](#).

Supported languages

Language support varies by Speech service functionality.

Choose a Speech feature

Speech-to-text

The table in this section summarizes the locales and voices supported for Speech-to-text. Please see the table footnotes for more details.

Additional remarks for Speech-to-text locales are included in the [Custom Speech](#) section below.

Tip

Try out the [Real-time Speech-to-text tool](#) without having to use any code.

Locale (BCP-47)	Language	Custom Speech support
af-ZA	Afrikaans (South Africa)	Plain text
am-ET	Amharic (Ethiopia)	Plain text
ar-AE	Arabic (United Arab Emirates)	Plain text

Locale (BCP-47)	Language	Custom Speech support
ar-BH	Arabic (Bahrain)	Audio + human-labeled transcript
		Plain text
ar-DZ	Arabic (Algeria)	Audio + human-labeled transcript
		Plain text
ar-EG	Arabic (Egypt)	Audio + human-labeled transcript
		Plain text
ar-IL	Arabic (Israel)	Plain text
ar-IQ	Arabic (Iraq)	Plain text
ar-JO	Arabic (Jordan)	Plain text
ar-KW	Arabic (Kuwait)	Plain text
ar-LB	Arabic (Lebanon)	Plain text
ar-LY	Arabic (Libya)	Plain text
ar-MA	Arabic (Morocco)	Audio + human-labeled transcript
		Plain text
ar-OM	Arabic (Oman)	Plain text
ar-PS	Arabic (Palestinian Authority)	Plain text
ar-QA	Arabic (Qatar)	Plain text
ar-SA	Arabic (Saudi Arabia)	Audio + human-labeled transcript
		Plain text
ar-SY	Arabic (Syria)	Plain text

Locale (BCP-47)	Language	Custom Speech support
ar-TN	Arabic (Tunisia)	Audio + human-labeled transcript Plain text
ar-YE	Arabic (Yemen)	Audio + human-labeled transcript Plain text
az-AZ	Azerbaijani (Latin, Azerbaijan)	Plain text
bg-BG	Bulgarian (Bulgaria)	Plain text
bn-IN	Bengali (India)	Plain text
bs-BA	Bosnian (Bosnia and Herzegovina)	Plain text
ca-ES	Catalan	Plain text Pronunciation
cs-CZ	Czech (Czechia)	Plain text Pronunciation
cy-GB	Welsh (United Kingdom)	Plain text
da-DK	Danish (Denmark)	Audio + human-labeled transcript Plain text Pronunciation
de-AT	German (Austria)	Plain text Structured text Pronunciation
de-CH	German (Switzerland)	Audio + human-labeled transcript Plain text Pronunciation

Locale (BCP-47)	Language	Custom Speech support
de-DE	German (Germany)	Audio + human-labeled transcript Plain text Structured text Pronunciation Phrase list
el-GR	Greek (Greece)	Plain text
en-AU	English (Australia)	Audio + human-labeled transcript Audio Plain text Structured text Pronunciation Phrase list
en-CA	English (Canada)	Audio + human-labeled transcript Audio Plain text Structured text Pronunciation Phrase list

Locale (BCP-47)	Language	Custom Speech support
en-GB	English (United Kingdom)	<p>Audio + human-labeled transcript</p> <p>Audio</p> <p>Plain text</p> <p>Structured text</p> <p>Pronunciation</p> <p>Phrase list</p>
en-GH	English (Ghana)	<p>Audio + human-labeled transcript</p> <p>Audio</p> <p>Plain text</p> <p>Structured text</p> <p>Pronunciation</p>
en-HK	English (Hong Kong SAR)	<p>Audio + human-labeled transcript</p> <p>Audio</p> <p>Plain text</p> <p>Pronunciation</p>
en-IE	English (Ireland)	<p>Audio + human-labeled transcript</p> <p>Audio</p> <p>Plain text</p> <p>Pronunciation</p>

Locale (BCP-47)	Language	Custom Speech support
en-IN	English (India)	Audio + human-labeled transcript
		Plain text
		Structured text
		Pronunciation
		Phrase list
en-KE	English (Kenya)	Audio + human-labeled transcript
		Audio
		Plain text
		Structured text
		Pronunciation
en-NG	English (Nigeria)	Audio + human-labeled transcript
		Audio
		Plain text
		Pronunciation
en-NZ	English (New Zealand)	Audio + human-labeled transcript
		Audio
		Plain text
		Pronunciation

Locale (BCP-47)	Language	Custom Speech support
en-PH	English (Philippines)	Audio + human-labeled transcript Audio Plain text Pronunciation
en-SG	English (Singapore)	Audio + human-labeled transcript Audio Plain text Pronunciation
en-TZ	English (Tanzania)	Audio + human-labeled transcript Audio Plain text Structured text Pronunciation
en-US	English (United States)	Audio + human-labeled transcript Audio Plain text Structured text Pronunciation Phrase list

Locale (BCP-47)	Language	Custom Speech support
en-ZA	English (South Africa)	Audio + human-labeled transcript Audio Plain text Pronunciation
es-AR	Spanish (Argentina)	Plain text Pronunciation
es-BO	Spanish (Bolivia)	Plain text Pronunciation
es-CL	Spanish (Chile)	Plain text Pronunciation
es-CO	Spanish (Colombia)	Plain text Pronunciation
es-CR	Spanish (Costa Rica)	Plain text Pronunciation
es-CU	Spanish (Cuba)	Plain text Pronunciation
es-DO	Spanish (Dominican Republic)	Plain text Pronunciation
es-EC	Spanish (Ecuador)	Plain text Pronunciation

Locale (BCP-47)	Language	Custom Speech support
es-ES	Spanish (Spain)	Audio + human-labeled transcript Plain text Structured text Pronunciation Phrase list
es-GQ	Spanish (Equatorial Guinea)	Plain text
es-GT	Spanish (Guatemala)	Plain text Pronunciation
es-HN	Spanish (Honduras)	Plain text Pronunciation
es-MX	Spanish (Mexico)	Audio + human-labeled transcript Plain text Structured text Pronunciation Phrase list
es-NI	Spanish (Nicaragua)	Plain text Pronunciation
es-PA	Spanish (Panama)	Plain text Pronunciation
es-PE	Spanish (Peru)	Plain text Pronunciation
es-PR	Spanish (Puerto Rico)	Plain text Pronunciation

Locale (BCP-47)	Language	Custom Speech support
es-PY	Spanish (Paraguay)	Plain text
		Pronunciation
es-SV	Spanish (El Salvador)	Plain text
		Pronunciation
es-US	Spanish (United States)	Plain text
		Pronunciation
es-UY	Spanish (Uruguay)	Plain text
		Pronunciation
es-VE	Spanish (Venezuela)	Plain text
		Pronunciation
et-EE	Estonian (Estonia)	Plain text
		Pronunciation
eu-ES	Basque	Plain text
fa-IR	Persian (Iran)	Plain text
fi-FI	Finnish (Finland)	Plain text
		Pronunciation
fil-PH	Filipino (Philippines)	Plain text
		Pronunciation
fr-BE	French (Belgium)	Audio + human-labeled transcript
		Plain text

Locale (BCP-47)	Language	Custom Speech support
fr-CA	French (Canada)	Audio + human-labeled transcript
		Plain text
		Structured text
		Pronunciation
		Phrase list
fr-CH	French (Switzerland)	Plain text
		Pronunciation
fr-FR	French (France)	Audio + human-labeled transcript
		Plain text
		Structured text
		Pronunciation
		Phrase list
ga-IE	Irish (Ireland)	Plain text
		Pronunciation
gl-ES	Galician	Plain text
gu-IN	Gujarati (India)	Plain text
he-IL	Hebrew (Israel)	Plain text
hi-IN	Hindi (India)	Audio + human-labeled transcript
		Plain text
		Phrase list
hr-HR	Croatian (Croatia)	Plain text
		Pronunciation

Locale (BCP-47)	Language	Custom Speech support
hu-HU	Hungarian (Hungary)	Audio + human-labeled transcript
		Plain text
		Pronunciation
hy-AM	Armenian (Armenia)	Plain text
id-ID	Indonesian (Indonesia)	Plain text
		Pronunciation
is-IS	Icelandic (Iceland)	Plain text
it-CH	Italian (Switzerland)	Plain text
it-IT	Italian (Italy)	Audio + human-labeled transcript
		Plain text
		Structured text
		Pronunciation
		Phrase list
ja-JP	Japanese (Japan)	Audio + human-labeled transcript
		Plain text
		Structured text
		Phrase list
jv-ID	Javanese (Latin, Indonesia)	Plain text
ka-GE	Georgian (Georgia)	Plain text
kk-KZ	Kazakh (Kazakhstan)	Plain text
km-KH	Khmer (Cambodia)	Plain text
kn-IN	Kannada (India)	Plain text

Locale (BCP-47)	Language	Custom Speech support
ko-KR	Korean (Korea)	Audio + human-labeled transcript
		Plain text
		Structured text
		Phrase list
lo-LA	Lao (Laos)	Plain text
lt-LT	Lithuanian (Lithuania)	Plain text
		Pronunciation
lv-LV	Latvian (Latvia)	Plain text
		Pronunciation
mk-MK	Macedonian (North Macedonia)	Plain text
ml-IN	Malayalam (India)	Plain text
mn-MN	Mongolian (Mongolia)	Plain text
mr-IN	Marathi (India)	Plain text
ms-MY	Malay (Malaysia)	Audio + human-labeled transcript
		Plain text
mt-MT	Maltese (Malta)	Plain text
my-MM	Burmese (Myanmar)	Plain text
nb-NO	Norwegian (Bokmål, Norway)	Audio + human-labeled transcript
		Plain text
ne-NP	Nepali (Nepal)	Plain text
nl-BE	Dutch (Belgium)	Plain text

Locale (BCP-47)	Language	Custom Speech support
nl-NL	Dutch (Netherlands)	Audio + human-labeled transcript
		Plain text
		Pronunciation
pl-PL	Polish (Poland)	Plain text
		Pronunciation
ps-AF	Pashto (Afghanistan)	Plain text
		Audio + human-labeled transcript
		Plain text
pt-BR	Portuguese (Brazil)	Audio + human-labeled transcript
		Plain text
		Structured text
		Pronunciation
		Phrase list
pt-PT	Portuguese (Portugal)	Plain text
		Pronunciation
ro-RO	Romanian (Romania)	Plain text
		Pronunciation
ru-RU	Russian (Russia)	Audio + human-labeled transcript
		Plain text
si-LK	Sinhala (Sri Lanka)	Plain text
sk-SK	Slovak (Slovakia)	Plain text
		Pronunciation
sl-SI	Slovenian (Slovenia)	Plain text
		Pronunciation
so-SO	Somali (Somalia)	Plain text

Locale (BCP-47)	Language	Custom Speech support
sq-AL	Albanian (Albania)	Plain text
sr-RS	Serbian (Cyrillic, Serbia)	Plain text
sv-SE	Swedish (Sweden)	Audio + human-labeled transcript
		Plain text
		Pronunciation
sw-KE	Swahili (Kenya)	Audio + human-labeled transcript
		Plain text
sw-TZ	Swahili (Tanzania)	Audio + human-labeled transcript
		Plain text
ta-IN	Tamil (India)	Plain text
te-IN	Telugu (India)	Plain text
th-TH	Thai (Thailand)	Audio + human-labeled transcript
		Plain text
tr-TR	Turkish (Turkey)	Audio + human-labeled transcript
		Plain text
uk-UA	Ukrainian (Ukraine)	Plain text
uz-UZ	Uzbek (Latin, Uzbekistan)	Plain text
vi-VN	Vietnamese (Vietnam)	Plain text
wuu-CN	Chinese (Wu, Simplified)	Audio + human-labeled transcript
		Plain text
yue-CN	Chinese (Cantonese, Simplified)	Plain text

Locale (BCP-47)	Language	Custom Speech support
zh-CN	Chinese (Mandarin, Simplified)	Audio + human-labeled transcript
		Plain text
		Structured text
		Phrase list
zh-CN-sichuan	Chinese (Southwestern Mandarin, Simplified)	Plain text
zh-HK	Chinese (Cantonese, Traditional)	Plain text
zh-TW	Chinese (Taiwanese Mandarin, Traditional)	Plain text
zu-ZA	Zulu (South Africa)	Plain text

Custom Speech

To improve Speech-to-text recognition accuracy, customization is available for some languages and base models. Depending on the locale, you can upload audio + human-labeled transcripts, plain text, structured text, and pronunciation data. By default, plain text customization is supported for all available base models. To learn more about customization, see [Custom Speech](#).

Next steps

- [Region support](#)

Additional resources

Documentation

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Speech-to-text REST API - Speech service - Azure Cognitive Services](#)

Get reference documentation for Speech-to-text REST API.

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[Speech-to-text overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the speech-to-text feature of the Speech Service.

[azure.cognitiveservices.speech.SpeechRecognizer class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, language, source_language_config or auto_detect_source_language_config.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[Show 5 more](#)

Training

Learning paths and modules

[Create speech-enabled apps with the Speech service - Training](#)

Create speech-enabled apps with the Speech service

Speech service supported regions

Article • 12/14/2022 • 3 minutes to read

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. You can perform custom configurations to your speech experience, for all regions, at the [Speech Studio](#).

Keep in mind the following points:

- If your application uses a [Speech SDK](#), you provide the region identifier, such as `westus`, when you create a `SpeechConfig`. Make sure the region matches the region of your subscription.
- If your application uses one of the Speech service 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. If you attempt to use them with other regions, you get authentication errors.

ⓘ Note

Speech service doesn't store or process customer data outside the region the customer deploys the service instance in.

Speech service

The following regions are supported for Speech service features such as speech-to-text, text-to-speech, pronunciation assessment, and translation. The geographies are listed in alphabetical order.

Geography	Region	Region identifier
Africa	South Africa North	<code>southafricanorth</code> ⁶
Asia Pacific	East Asia	<code>eastasia</code> ⁵
Asia Pacific	Southeast Asia	<code>southeastasia</code> ^{1,2,3,4,5}
Asia Pacific	Australia East	<code>australiaeast</code> ^{1,2,3,4}
Asia Pacific	Central India	<code>centralindia</code> ^{1,2,3,4,5}
Asia Pacific	Japan East	<code>japaneast</code> ^{2,5}

Geography	Region	Region identifier
Asia Pacific	Japan West	japanwest
Asia Pacific	Korea Central	koreacentral ²
Canada	Canada Central	canadacentral ¹
Europe	North Europe	norteurope ^{1,2,4,5}
Europe	West Europe	westeurope ^{1,2,3,4,5}
Europe	France Central	francecentral
Europe	Germany West Central	germanywestcentral
Europe	Norway East	norwayeast
Europe	Switzerland North	switzerlandnorth ⁶
Europe	Switzerland West	switzerlandwest
Europe	UK South	uksouth ^{1,2,3,4}
Middle East	UAE North	uaenorth ⁶
South America	Brazil South	brazilsouth ⁶
US	Central US	centralus
US	East US	eastus ^{1,2,3,4,5}
US	East US 2	eastus2 ^{1,2,4,5}
US	North Central US	northcentralus ^{4,6}
US	South Central US	southcentralus ^{1,2,3,4,5,6}
US	West Central US	westcentralus ⁵
US	West US	westus ^{2,5}
US	West US 2	westus2 ^{1,2,4,5}
US	West US 3	westus3

¹ The region has dedicated hardware for Custom Speech training. If you plan to train a custom model with audio data, use one of the regions with dedicated hardware for faster training. Then you can [copy the trained model](#) to another region.

² The region is available for Custom Neural Voice training. You can copy a trained neural voice model to other regions for deployment.

³ The Long Audio API is available in the region.

⁴ The region supports custom keyword advanced models.

⁵ The region supports keyword verification.

⁶ The region does not support Speaker Recognition.

Intent recognition

Available regions for intent recognition via the Speech SDK are in the following table.

Global region	Region	Region identifier
Asia	East Asia	eastasia
Asia	Southeast Asia	southeastasia
Australia	Australia East	australiaeast
Europe	North Europe	northeurope
Europe	West Europe	westeurope
North America	East US	eastus
North America	East US 2	eastus2
North America	South Central US	southcentralus
North America	West Central US	westcentralus
North America	West US	westus
North America	West US 2	westus2
South America	Brazil South	brazilsouth

This is a subset of the publishing regions supported by the [Language Understanding service \(LUIS\)](#).

Voice assistants

The [Speech SDK](#) supports voice assistant capabilities through [Direct Line Speech](#) for regions in the following table.

Global region	Region	Region identifier
North America	West US	<code>westus</code>
North America	West US 2	<code>westus2</code>
North America	East US	<code>eastus</code>
North America	East US 2	<code>eastus2</code>
North America	West Central US	<code>westcentralus</code>
North America	South Central US	<code>southcentralus</code>
Europe	West Europe	<code>westeurope</code>
Europe	North Europe	<code>northeurope</code>
Asia	East Asia	<code>eastasia</code>
Asia	Southeast Asia	<code>southeastasia</code>
India	Central India	<code>centralindia</code>

Additional resources

Documentation

[Speech Studio overview - Speech service - Azure Cognitive Services](#)

Speech Studio is a set of UI-based tools for building and integrating features from Azure Speech service in your applications.

[Keyword recognition overview - Speech service - Azure Cognitive Services](#)

An overview of the features, capabilities, and restrictions for keyword recognition by using the Speech Software Development Kit (SDK).

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats,

and configuration options for speech recognition.

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[Asynchronous Conversation Transcription - Speech service - Azure Cognitive Services](#)

Learn how to use asynchronous Conversation Transcription using the Speech service. Available for Java and C# only.

[Show 5 more](#)

Speech service quotas and limits

Article • 01/18/2023 • 10 minutes to read

This article contains a quick reference and a detailed description of the quotas and limits for the Speech service in Azure Cognitive Services. The information applies to all [pricing tiers](#) of the service. It also contains some best practices to avoid request throttling.

Quotas and limits reference

The following sections provide you with a quick guide to the quotas and limits that apply to Speech service.

Speech-to-text quotas and limits per resource

In the following tables, the parameters without the **Adjustable** row aren't adjustable for all price tiers.

Online transcription

You can use online transcription with the [Speech SDK](#) or the [speech-to-text REST API](#) for short audio.

Quota	Free (F0) ¹	Standard (S0)
Concurrent request limit - base model endpoint	1	100 (default value)
Adjustable	No ²	Yes ²
Concurrent request limit - custom endpoint	1	100 (default value)
Adjustable	No ²	Yes ²

Batch transcription

Quota	Free (F0) ¹	Standard (S0)
Speech-to-text REST API limit	Not available for F0	300 requests per minute
Max audio input file size	N/A	1 GB

Quota	Free (F0)¹	Standard (S0)
Max input blob size (for example, can contain more than one file in a zip archive). Note the file size limit from the preceding row.	N/A	2.5 GB
Max blob container size	N/A	5 GB
Max number of blobs per container	N/A	10000
Max number of files per transcription request (when you're using multiple content URLs as input).	N/A	1000

Model customization

Quota	Free (F0)¹	Standard (S0)
REST API limit	300 requests per minute	300 requests per minute
Max number of speech datasets	2	500
Max acoustic dataset file size for data import	2 GB	2 GB
Max language dataset file size for data import	200 MB	1.5 GB
Max pronunciation dataset file size for data import	1 KB	1 MB
Max text size when you're using the <code>text</code> parameter in the Models_Create API request	200 KB	500 KB

¹ For the free (F0) pricing tier, see also the monthly allowances at the [pricing page](#).

² See [additional explanations](#), [best practices](#), and [adjustment instructions](#).

Text-to-speech quotas and limits per Speech resource

In the following tables, the parameters without the **Adjustable** row aren't adjustable for all price tiers.

General

Quota	Free (F0)³	Standard (S0)
Max number of transactions per certain time period		

Quota	Free (F0)³	Standard (S0)
Real-time API. Prebuilt neural voices and custom neural voices.	20 transactions per 60 seconds	200 transactions per second (TPS) (default value)
Adjustable	No ⁴	Yes ⁵ , up to 1000 TPS
HTTP-specific quotas		
Max audio length produced per request	10 min	10 min
Max total number of distinct <voice> and <audio> tags in SSML	50	50
Websocket specific quotas		
Max audio length produced per turn	10 min	10 min
Max total number of distinct <voice> and <audio> tags in SSML	50	50
Max SSML message size per turn	64 KB	64 KB

Custom Neural Voice

Quota	Free (F0)³	Standard (S0)
Max number of transactions per second (TPS)	Not available for F0	See General
Max number of datasets	N/A	500
Max number of simultaneous dataset uploads	N/A	5
Max data file size for data import per dataset	N/A	2 GB
Upload of long audios or audios without script	N/A	Yes
Max number of simultaneous model trainings	N/A	3
Max number of custom endpoints	N/A	50

Audio Content Creation tool

Quota	Free (F0)	Standard (S0)
File size	3,000 characters per file	20,000 characters per file
Export to audio library	1 concurrent task	N/A

³ For the free (F0) pricing tier, see also the monthly allowances at the [pricing page](#).

⁴ See [additional explanations](#) and [best practices](#).

⁵ See [additional explanations](#), [best practices](#), and [adjustment instructions](#).

Detailed description, quota adjustment, and best practices

Before requesting a quota increase (where applicable), ensure that it's necessary. Speech service uses autoscaling technologies to bring the required computational resources in on-demand mode. At the same time, Speech service tries to keep your costs low by not maintaining an excessive amount of hardware capacity.

Let's look at an example. Suppose that your application receives response code 429, which indicates that there are too many requests. Your application receives this response even though your workload is within the limits defined by the [Quotas and limits reference](#). The most likely explanation is that Speech service is scaling up to your demand and didn't reach the required scale yet. Therefore the service doesn't immediately have enough resources to serve the request. In most cases, this throttled state is transient.

General best practices to mitigate throttling during autoscaling

To minimize issues related to throttling, it's a good idea to use the following techniques:

- Implement retry logic in your application.
- Avoid sharp changes in the workload. Increase the workload gradually. For example, let's say your application is using text-to-speech, and your current workload is 5 TPS. The next second, you increase the load to 20 TPS (that is, four times more). Speech service immediately starts scaling up to fulfill the new load, but is unable to scale as needed within one second. Some of the requests will get response code 429 (too many requests).
- Test different load increase patterns. For more information, see the [workload pattern example](#).
- Create additional Speech service resources in *different* regions, and distribute the workload among them. (Creating multiple Speech service resources in the same region will not affect the performance, because all resources will be served by the same backend cluster).

The next sections describe specific cases of adjusting quotas.

Speech-to-text: increase online transcription concurrent request limit

By default, the number of concurrent requests is limited to 100 per resource in the base model, and 100 per custom endpoint in the custom model. For the standard pricing tier, you can increase this amount. Before submitting the request, ensure that you're familiar with the material discussed earlier in this article, such as the best practices to mitigate throttling.

Note

If you use custom models, be aware that one Speech service resource might be associated with many custom endpoints hosting many custom model deployments. Each custom endpoint has the default limit of concurrent requests (100) set by creation. If you need to adjust it, you need to make the adjustment of each custom endpoint *separately*. Note also that the value of the limit of concurrent requests for the base model of a resource has *no* effect to the custom endpoints associated with this resource.

Increasing the limit of concurrent requests doesn't directly affect your costs. Speech service uses a payment model that requires that you pay only for what you use. The limit defines how high the service can scale before it starts throttle your requests.

Concurrent request limits for base and custom models need to be adjusted separately.

You aren't able to see the existing value of the concurrent request limit parameter in the Azure portal, the command-line tools, or API requests. To verify the existing value, create an Azure support request.

Note

Speech containers don't require increases of the concurrent request limit, because containers are constrained only by the CPUs of the hardware they are hosted on. Speech containers do, however, have their own capacity limitations that should be taken into account. For more information, see the [Speech containers FAQ](#).

Have the required information ready

- For the base model:
 - Speech resource ID

- Region
- For the custom model:
 - Region
 - Custom endpoint ID

How to get information for the base model:

1. Go to the [Azure portal](#).
2. Select the Speech service resource for which you would like to increase the concurrency request limit.
3. From the **Resource Management** group, select **Properties**.
4. Copy and save the values of the following fields:
 - **Resource ID**
 - **Location** (your endpoint region)

How to get information for the custom model:

1. Go to the [Speech Studio](#) portal.
2. Sign in if necessary, and go to **Custom Speech**.
3. Select your project, and go to **Deployment**.
4. Select the required endpoint.
5. Copy and save the values of the following fields:
 - **Service Region** (your endpoint region)
 - **Endpoint ID**

Create and submit a support request

Initiate the increase of the limit for concurrent requests for your resource, or if necessary check the current limit, by submitting a support request. Here's how:

1. Ensure you have the required information listed in the previous section.
2. Go to the [Azure portal](#).
3. Select the Speech service resource for which you would like to increase (or to check) the concurrency request limit.
4. In the **Support + troubleshooting** group, select **New support request**. A new window will appear, with auto-populated information about your Azure subscription and Azure resource.
5. In **Summary**, describe what you want (for example, "Increase speech-to-text concurrency request limit").
6. In **Problem type**, select **Quota or Subscription issues**.
7. In **Problem subtype**, select either:

- Quota or concurrent requests increase for an increase request.
- Quota or usage validation to check the existing limit.

8. Select **Next: Solutions**. Proceed further with the request creation.

9. On the **Details** tab, in the **Description** field, enter the following:

- A note that the request is about the speech-to-text quota.
- Choose either the base or custom model.
- The Azure resource information you [collected previously](#).
- Any other required information.

10. On the **Review + create** tab, select **Create**.

11. Note the support request number in Azure portal notifications. You'll be contacted shortly about your request.

Example of a workload pattern best practice

Here's a general example of a good approach to take. It's meant only as a template that you can adjust as necessary for your own use.

Suppose that a Speech service resource has the concurrent request limit set to 300. Start the workload from 20 concurrent connections, and increase the load by 20 concurrent connections every 90-120 seconds. Control the service responses, and implement the logic that falls back (reduces the load) if you get too many requests (response code 429). Then, retry the load increase in one minute, and if it still doesn't work, try again in two minutes. Use a pattern of 1-2-4-4 minutes for the intervals.

Generally, it's a very good idea to test the workload and the workload patterns before going to production.

Text-to-speech: increase concurrent request limit

For the standard pricing tier, you can increase this amount. Before submitting the request, ensure that you're familiar with the material discussed earlier in this article, such as the best practices to mitigate throttling.

Increasing the limit of concurrent requests doesn't directly affect your costs. Speech service uses a payment model that requires that you pay only for what you use. The limit defines how high the service can scale before it starts throttle your requests.

You aren't able to see the existing value of the concurrent request limit parameter in the Azure portal, the command-line tools, or API requests. To verify the existing value, create an Azure support request.

Note

Speech containers don't require increases of the concurrent request limit, because containers are constrained only by the CPUs of the hardware they are hosted on.

Prepare the required information

To create an increase request, you need to provide your information.

- For the prebuilt voice:
 - Speech resource ID
 - Region
- For the custom voice:
 - Deployment region
 - Custom endpoint ID

How to get information for the prebuilt voice:

1. Go to the [Azure portal](#).
2. Select the Speech service resource for which you would like to increase the concurrency request limit.
3. From the **Resource Management** group, select **Properties**.
4. Copy and save the values of the following fields:
 - **Resource ID**
 - **Location** (your endpoint region)

How to get information for the custom voice:

1. Go to the [Speech Studio](#) portal.
2. Sign in if necessary, and go to **Custom Voice**.
3. Select your project, and go to **Deploy model**.
4. Select the required endpoint.
5. Copy and save the values of the following fields:
 - **Service Region** (your endpoint region)
 - **Endpoint ID**

Create and submit a support request

Initiate the increase of the limit for concurrent requests for your resource, or if necessary check the current limit, by submitting a support request. Here's how:

1. Ensure you have the required information listed in the previous section.
 2. Go to the [Azure portal](#).
 3. Select the Speech service resource for which you would like to increase (or to check) the concurrency request limit.
 4. In the **Support + troubleshooting** group, select **New support request**. A new window will appear, with auto-populated information about your Azure subscription and Azure resource.
 5. In **Summary**, describe what you want (for example, "Increase text-to-speech concurrency request limit").
 6. In **Problem type**, select **Quota or Subscription issues**.
 7. In **Problem subtype**, select either:
 - **Quota or concurrent requests increase** for an increase request.
 - **Quota or usage validation** to check the existing limit.
 8. On the **Recommended solution** tab, select **Next**.
 9. On the **Additional details** tab, fill in all the required items. And in the **Details** field, enter the following:
 - A note that the request is about the text-to-speech quota.
 - Choose either the prebuilt voice or custom voice.
 - The Azure resource information you [collected previously](#).
 - Any other required information.
 10. On the **Review + create** tab, select **Create**.
 11. Note the support request number in Azure portal notifications. You'll be contacted shortly about your request.
-

Additional resources

Documentation

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Custom Speech overview - Speech service - Azure Cognitive Services](#)

Custom Speech is a set of online tools that allows you to evaluate and improve the Microsoft speech-to-text accuracy for your applications, tools, and products.

[Conversation transcription overview - Speech service - Azure Cognitive Services](#)

You use the conversation transcription feature for meetings. It combines recognition, speaker ID, and diarization to provide transcription of any conversation.

[Speech-to-text FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions about the speech-to-text service.

[Real-time Conversation Transcription quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, learn how to transcribe meetings and other conversations. You can add, remove, and identify multiple participants by streaming audio to the Speech service.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Speaker Recognition quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you use speaker recognition to confirm who is speaking. Learn about common design patterns for working with speaker verification and identification.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Show 5 more](#)

What is Speech Studio?

Article • 02/02/2023 • 2 minutes to read

[Speech Studio](#) is a set of UI-based tools for building and integrating features from Azure Cognitive Services Speech service in your applications. You create projects in Speech Studio by using a no-code approach, and then reference those assets in your applications by using the [Speech SDK](#), the [Speech CLI](#), or the REST APIs.

Tip

You can try speech-to-text and text-to-speech in [Speech Studio](#) without signing up or writing any code.

Speech Studio scenarios

Explore, try out, and view sample code for some of common use cases.

- [Captioning](#): Choose a sample video clip to see real-time or offline processed captioning results. Learn how to synchronize captions with your input audio, apply profanity filters, get partial results, apply customizations, and identify spoken languages for multilingual scenarios. For more information, see the [captioning quickstart](#).
- [Call Center](#): View a demonstration on how to use the Language and Speech services to analyze call center conversations. Transcribe calls in real-time or process a batch of calls, redact personally identifying information, and extract insights such as sentiment to help with your call center use case. For more information, see the [call center quickstart](#).

Speech Studio features

In Speech Studio, the following Speech service features are available as project types:

- [Real-time speech-to-text](#): Quickly test speech-to-text by dragging audio files here without having to use any code. This is a demo tool for seeing how speech-to-text works on your audio samples. To explore the full functionality, see [What is speech-to-text?](#).
- [Custom Speech](#): Create speech recognition models that are tailored to specific vocabulary sets and styles of speaking. In contrast to the base speech recognition

model, Custom Speech models become part of your unique competitive advantage because they're not publicly accessible. To get started with uploading sample audio to create a Custom Speech model, see [Upload training and testing datasets](#).

- [Pronunciation assessment](#): Evaluate speech pronunciation and give speakers feedback on the accuracy and fluency of spoken audio. Speech Studio provides a sandbox for testing this feature quickly, without code. To use the feature with the Speech SDK in your applications, see the [Pronunciation assessment](#) article.
- [Voice Gallery](#): Build apps and services that speak naturally. Choose from a broad portfolio of [languages, voices, and variants](#). Bring your scenarios to life with highly expressive and human-like neural voices.
- [Custom Voice](#): Create custom, one-of-a-kind voices for text-to-speech. You supply audio files and create matching transcriptions in Speech Studio, and then use the custom voices in your applications. To create and use custom voices via endpoints, see [Create and use your voice model](#).
- [Audio Content Creation](#): A no-code approach for text-to-speech synthesis. You can use the output audio as-is, or as a starting point for further customization. You can build highly natural audio content for a variety of scenarios, such as audiobooks, news broadcasts, video narrations, and chat bots. For more information, see the [Audio Content Creation](#) documentation.
- [Custom Keyword](#): A custom keyword is a word or short phrase that you can use to voice-activate a product. You create a custom keyword in Speech Studio, and then generate a binary file to [use with the Speech SDK](#) in your applications.
- [Custom Commands](#): Easily build rich, voice-command apps that are optimized for voice-first interaction experiences. Custom Commands provides a code-free authoring experience in Speech Studio, an automatic hosting model, and relatively lower complexity. The feature helps you focus on building the best solution for your voice-command scenarios. For more information, see the [Develop Custom Commands applications](#) guide. Also see [Integrate with a client application by using the Speech SDK](#).

Next steps

- [Explore Speech Studio](#)

Additional resources

Documentation

[Custom Speech overview - Speech service - Azure Cognitive Services](#)

Custom Speech is a set of online tools that allows you to evaluate and improve the Microsoft speech-to-text accuracy for your applications, tools, and products.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[Speech-to-text overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the speech-to-text feature of the Speech Service.

[Train a Custom Speech model - Speech service - Azure Cognitive Services](#)

Learn how to train Custom Speech models. Training a speech-to-text model can improve recognition accuracy for the Microsoft base model or a custom model.

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Quickstart: The Speech CLI - Speech service - Azure Cognitive Services](#)

In this Azure Speech CLI quickstart, you interact with speech-to-text, text-to-speech, and speech translation without having to write code.

[Speech-to-text REST API - Speech service - Azure Cognitive Services](#)

Get reference documentation for Speech-to-text REST API.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

What is the Speech CLI?

Article • 12/01/2022 • 2 minutes to read

The Speech CLI is a command-line tool for using Speech service without having to write any code. The Speech CLI requires minimal setup. You can easily use it to experiment with key features of Speech service and see how it works with your use cases. Within minutes, you can run simple test workflows, such as batch speech-recognition from a directory of files or text-to-speech on a collection of strings from a file. Beyond simple workflows, the Speech CLI is production-ready, and you can scale it up to run larger processes by using automated `.bat` or shell scripts.

Most features in the Speech SDK are available in the Speech CLI, and some advanced features and customizations are simplified in the Speech CLI. As you're deciding when to use the Speech CLI or the Speech SDK, consider the following guidance.

Use the Speech CLI when:

- You want to experiment with Speech service features with minimal setup and without having to write code.
- You have relatively simple requirements for a production application that uses Speech service.

Use the Speech SDK when:

- You want to integrate Speech service functionality within a specific language or platform (for example, C#, Python, or C++).
- You have complex requirements that might require advanced service requests.
- You're developing custom behavior, including response streaming.

Core features

- **Speech recognition:** Convert speech to text either from audio files or directly from a microphone, or transcribe a recorded conversation.
- **Speech synthesis:** Convert text to speech either by using input from text files or by inputting directly from the command line. Customize speech output characteristics by using [Speech Synthesis Markup Language \(SSML\) configurations](#).
- **Speech translation:** Translate audio in a source language to text or audio in a target language.

- **Run on Azure compute resources:** Send Speech CLI commands to run on an Azure remote compute resource by using `spx webjob`.

Get started

To get started with the Speech CLI, see the [quickstart](#). This article shows you how to run some basic commands. It also gives you slightly more advanced commands for running batch operations for speech-to-text and text-to-speech. After you've read the basics article, you should understand the syntax well enough to start writing some custom commands or automate simple Speech service operations.

Next steps

- [Get started with the Azure Speech CLI](#)
- [Speech CLI configuration options](#)
- [Speech CLI batch operations](#)

Quickstart: Get started with the Azure Speech CLI

Article • 01/13/2023 • 8 minutes to read

In this article, you'll learn how to use the Azure Speech CLI (also called SPX) to access Speech services such as speech-to-text, text-to-speech, and speech translation, without having to write any code. The Speech CLI is production ready, and you can use it to automate simple workflows in the Speech service by using `.bat` or shell scripts.

This article assumes that you have working knowledge of the Command Prompt window, terminal, or PowerShell.

ⓘ Note

In PowerShell, the stop-parsing token (`--%`) should follow `spx`. For example, run `spx --% config @region` to view the current region config value.

Download and install

Windows

Follow these steps to install the Speech CLI on Windows:

1. Install the [Microsoft Visual C++ Redistributable for Visual Studio 2019](#) for your platform. Installing it for the first time might require a restart.
2. Install [.NET 6](#).
3. Install the Speech CLI via the .NET CLI by entering this command:

.NET CLI

```
dotnet tool install --global Microsoft.CognitiveServices.Speech.CLI
```

To update the Speech CLI, enter this command:

.NET CLI

```
dotnet tool update --global Microsoft.CognitiveServices.Speech.CLI
```

Enter `spx` or `spx help` to see help for the Speech CLI.

Font limitations

On Windows, the Speech CLI can show only fonts that are available to the command prompt on the local computer. [Windows Terminal](#) supports all fonts that the Speech CLI produces interactively.

If you output to a file, a text editor like Notepad or a web browser like Microsoft Edge can also show all fonts.

Create a resource configuration

Terminal

To get started, you need a Speech resource key and region identifier (for example, `eastus`, `westus`). Create a Speech resource on the [Azure portal](#). For more information, see [Create a new Azure Cognitive Services resource](#).

To configure your resource key and region identifier, run the following commands:

Console

```
spx config @key --set SPEECH-KEY  
spx config @region --set SPEECH-REGION
```

The key and region are stored for future Speech CLI commands. To view the current configuration, run the following commands:

Console

```
spx config @key  
spx config @region
```

As needed, include the `clear` option to remove either stored value:

Console

```
spx config @key --clear  
spx config @region --clear
```

Basic usage

This section shows a few basic SPX commands that are often useful for first-time testing and experimentation. Start by viewing the help that's built into the tool by running the following command:

```
Console
```

```
spx
```

You can search help topics by keyword. For example, to see a list of Speech CLI usage examples, run the following command:

```
Console
```

```
spx help find --topics "examples"
```

To see options for the recognize command, run the following command:

```
Console
```

```
spx help recognize
```

Additional help commands are listed in the console output. You can enter these commands to get detailed help about subcommands.

Speech-to-text (speech recognition)

To convert speech to text (speech recognition) by using your system's default microphone, run the following command:

```
Console
```

```
spx recognize --microphone
```

After you run the command, SPX begins listening for audio on the current active input device. It stops listening when you select **Enter**. The spoken audio is then recognized and converted to text in the console output.

With the Speech CLI, you can also recognize speech from an audio file. Run the following command:

```
Console
```

```
spx recognize --file /path/to/file.wav
```

ⓘ Note

If you're using a Docker container, `--microphone` will not work.

If you're recognizing speech from an audio file in a Docker container, make sure that the audio file is located in the directory that you mounted previously.

💡 Tip

If you get stuck or want to learn more about the Speech CLI recognition options, you can run `spx help recognize`.

Text-to-speech (speech synthesis)

The following command takes text as input and then outputs the synthesized speech to the current active output device (for example, your computer speakers).

Console

```
spx synthesize --text "Testing synthesis using the Speech CLI" --speakers
```

You can also save the synthesized output to a file. In this example, let's create a file named *my-sample.wav* in the directory where you're running the command.

Console

```
spx synthesize --text "Enjoy using the Speech CLI." --audio output my-sample.wav
```

These examples presume that you're testing in English. However, Speech service supports speech synthesis in many languages. You can pull down a full list of voices either by running the following command or by visiting the [language support page](#).

Console

```
spx synthesize --voices
```

Here's a command for using one of the voices you've discovered.

Console

```
spx synthesize --text "Bienvenue chez moi." --voice fr-FR-AlainNeural --speakers
```

💡 Tip

If you get stuck or want to learn more about the Speech CLI recognition options, you can run `spx help synthesize`.

Speech-to-text translation

With the Speech CLI, you can also do speech-to-text translation. Run the following command to capture audio from your default microphone and output the translation as text. Keep in mind that you need to supply the `source` and `target` language with the `translate` command.

Console

```
spx translate --microphone --source en-US --target ru-RU
```

When you're translating into multiple languages, separate the language codes with a semicolon (;).

Console

```
spx translate --microphone --source en-US --target ru-RU;fr-FR;es-ES
```

If you want to save the output of your translation, use the `--output` flag. In this example, you'll also read from a file.

Console

```
spx translate --file /some/file/path/input.wav --source en-US --target ru-RU --output file /some/file/path/russian_translation.txt
```

ⓘ Note

For a list of all supported languages and their corresponding locale codes, see [Language and voice support for the Speech service](#).

💡 Tip

If you get stuck or want to learn more about the Speech CLI recognition options, you can run `spx help translate`.

Next steps

- [Install GStreamer to use the Speech CLI with MP3 and other formats](#)
 - [Configuration options for the Speech CLI](#)
 - [Batch operations with the Speech CLI](#)
-

Additional resources

Documentation

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[How to lower speech synthesis latency using Speech SDK - Azure Cognitive Services](#)

How to lower speech synthesis latency using Speech SDK, including streaming, pre-connection, and so on.

[azure.cognitiveservices.speech.SpeechConfig class](#)

Class that defines configurations for speech / intent recognition and speech synthesis. The configuration can be initialized in different ways: from subscription: pass a subscription key and a region from endpoint: pass an endpoint. Subscription key or authorization token are optional. from...

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[Install the Speech SDK - Azure Cognitive Services](#)

In this quickstart, you'll learn how to install the Speech SDK for your preferred programming language.

[Speech Studio overview - Speech service - Azure Cognitive Services](#)

Speech Studio is a set of UI-based tools for building and integrating features from Azure Speech service in your applications.

[SpeechRecognizer class](#)

Performs speech recognition from microphone, file, or other audio input streams, and gets transcribed text as result.

[Show 5 more](#)

Configure the Speech CLI datastore

Article • 05/08/2022 • 2 minutes to read

The [Speech CLI](#) can rely on settings in configuration files, which you can refer to using a @ symbol. The Speech CLI saves a new setting in a new ./spx/data subdirectory that is created in the current working directory for the Speech CLI. When looking for a configuration value, the Speech CLI searches your current working directory, then in the datastore at ./spx/data, and then in other datastores, including a final read-only datastore in the spx binary.

In the [Speech CLI quickstart](#), you used the datastore to save your @key and @region values, so you did not need to specify them with each spx command. Keep in mind, that you can use configuration files to store your own configuration settings, or even use them to pass URLs or other dynamic content generated at runtime.

For more details about datastore files, including use of default configuration files (@spx.default, @default.config, and @*.default.config for command-specific default settings), enter this command:

```
Console  
spx help advanced setup
```

nodefaults

The following example clears the @my.defaults configuration file, adds key-value pairs for key and region in the file, and uses the configuration in a call to spx recognize.

```
Console  
spx config @my.defaults --clear  
spx config @my.defaults --add key 000072626F6E20697320636F6F6C0000  
spx config @my.defaults --add region westus  
  
spx config @my.defaults  
  
spx recognize --nodefaults @my.defaults --file hello.wav
```

Dynamic configuration

You can also write dynamic content to a configuration file using the --output option.

For example, the following command creates a custom speech model and stores the URL of the new model in a configuration file. The next command waits until the model at that URL is ready for use before returning.

Console

```
spx csr model create --name "Example 4" --datasets @my.datasets.txt --output url @my.model.txt  
spx csr model status --model @my.model.txt --wait
```

The following example writes two URLs to the `@my.datasets.txt` configuration file. In this scenario, `--output` can include an optional `add` keyword to create a configuration file or append to the existing one.

Console

```
spx csr dataset create --name "LM" --kind Language --content https://crbn.us/data.txt --output url @my.datasets.txt  
spx csr dataset create --name "AM" --kind Acoustic --content https://crbn.us/audio.zip --output add url @my.datasets.txt  
  
spx config @my.datasets.txt
```

SPX config add

For readability, flexibility, and convenience, you can use a preset configuration with select output options.

For example, you might have the following requirements for [captioning](#):

- Recognize from the input file `caption.this.mp4`.
- Output WebVTT and SRT captions to the files `caption.vtt` and `caption.srt` respectively.
- Output the `offset`, `duration`, `resultid`, and `text` of each recognizing event to the file `each.result.tsv`.

You can create a preset configuration named `@caption.defaults` as shown here:

Console

```
spx config @caption.defaults --clear  
spx config @caption.defaults --add  
output.each.recognizing.result.offset=true  
spx config @caption.defaults --add  
output.each.recognizing.result.duration=true
```

```
spx config @caption.defaults --add
output.each.recognizing.result.resultid=true
spx config @caption.defaults --add output.each.recognizing.result.text=true
spx config @caption.defaults --add output.each.file.name=each.result.tsv
spx config @caption.defaults --add output.srt.file.name=caption.srt
spx config @caption.defaults --add output.vtt.file.name=caption.vtt
```

The settings are saved to the current directory in a file named `caption.defaults`. Here are the file contents:

```
output.each.recognizing.result.offset=true
output.each.recognizing.result.duration=true
output.each.recognizing.result.resultid=true
output.each.recognizing.result.text=true
output.all.file.name=output.result.tsv
output.each.file.name=each.result.tsv
output.srt.file.name=caption.srt
output.vtt.file.name=caption.vtt
```

Then, to generate [captions](#), you can run this command that imports settings from the `@caption.defaults` preset configuration:

Console

```
spx recognize --file caption.this.mp4 --format any --output vtt --output srt
@caption.defaults
```

Using the preset configuration as shown previously is similar to running the following command:

Console

```
spx recognize --file caption.this.mp4 --format any --output vtt file
caption.vtt --output srt file caption.srt --output each file each.result.tsv
--output all file output.result.tsv --output each recognizer recognizing
result offset --output each recognizer recognizing duration --output each
recognizer recognizing result resultid --output each recognizer recognizing
text
```

Next steps

- [Batch operations with the Speech CLI](#)

Configure the Speech CLI output options

Article • 09/20/2022 • 2 minutes to read

The [Speech CLI](#) output can be written to standard output or specified files.

For contextual help in the Speech CLI, you can run any of the following commands:

Console

```
spx help recognize output examples  
spx help synthesize output examples  
spx help translate output examples  
spx help intent output examples
```

Standard output

If the file argument is a hyphen (-), the results are written to standard output as shown in the following example.

Console

```
spx recognize --file caption.this.mp4 --format any --output vtt file - --  
output srt file - --output each file - @output.each.detailed --property  
SpeechServiceResponse_StablePartialResultThreshold=0 --profanity masked
```

Default file output

If you omit the `file` option, output is written to default files in the current directory.

For example, run the following command to write WebVTT and SRT [captions](#) to their own default files:

Console

```
spx recognize --file caption.this.mp4 --format any --output vtt --output srt  
--output each text --output all duration
```

The default file names are as follows, where the `<EPOCH_TIME>` is replaced at run time.

- The default SRT file name includes the input file name and the local operating system epoch time: `output.caption.this.<EPOCH_TIME>.srt`
- The default Web VTT file name includes the input file name and the local operating system epoch time: `output.caption.this.<EPOCH_TIME>.vtt`
- The default `output each` file name, `each.<EPOCH_TIME>.tsv`, includes the local operating system epoch time. This file is not created by default, unless you specify the `--output each` option.
- The default `output all` file name, `output.<EPOCH_TIME>.tsv`, includes the local operating system epoch time. This file is created by default.

Output to specific files

For output to files that you specify instead of the [default files](#), set the `file` option to the file name.

For example, to output both WebVTT and SRT [captions](#) to files that you specify, run the following command:

```
Console
spx recognize --file caption.this.mp4 --format any --output vtt file
caption.vtt --output srt file caption.srt --output each text --output each
file each.result.tsv --output all file output.result.tsv
```

The preceding command also outputs the `each` and `all` results to the specified files.

Output to multiple files

For translations with `spx translate`, separate files are created for the source language (such as `--source en-US`) and each target language (such as `--target de;fr;zh-Hant`).

For example, to output translated SRT and WebVTT captions, run the following command:

```
Console
spx translate --source en-US --target de;fr;zh-Hant --file caption.this.mp4
--format any --output vtt file caption.vtt --output srt file caption.srt
```

Captions should then be written to the following files: `caption.srt`, `caption.vtt`, `caption.de.srt`, `caption.de.vtt`, `caption.fr.srt`, `caption.fr.vtt`, `caption.zh-Hant.srt`, and `caption.zh-Hant.vtt`.

Suppress header

You can suppress the header line in the output file by setting the `has_header false` option:

```
spx recognize --nodefaults @my.defaults --file audio.wav --output recognized  
text --output file has_header false
```

See [Configure the Speech CLI datastore](#) for more information about `--nodefaults`.

Next steps

- [Captioning quickstart](#)

Run batch operations with the Speech CLI

Article • 09/20/2022 • 2 minutes to read

Common tasks when using Azure Speech services, are batch operations. In this article, you'll learn how to do batch speech to text (speech recognition), batch text to speech (speech synthesis) with the Speech CLI. Specifically, you'll learn how to:

- Run batch speech recognition on a directory of audio files
- Run batch speech synthesis by iterating over a `.tsv` file

Batch speech to text (speech recognition)

The Speech service is often used to recognize speech from audio files. In this example, you'll learn how to iterate over a directory using the Speech CLI to capture the recognition output for each `.wav` file. The `--files` flag is used to point at the directory where audio files are stored, and the wildcard `*.wav` is used to tell the Speech CLI to run recognition on every file with the extension `.wav`. The output for each recognition file is written as a tab separated value in `speech_output.tsv`.

Note

The `--threads` argument can be also used in the next section for `spx synthesize` commands, and the available threads will depend on the CPU and its current load percentage.

Console

```
spx recognize --files C:\your_wav_file_dir\*.wav --output file  
C:\output_dir\speech_output.tsv --threads 10
```

The following is an example of the output file structure.

Output

<code>audio.input.id</code>	<code>recognizer.session.started.sessionid</code>
<code>recognizer.recognized.result.text</code>	
<code>sample_1</code>	<code>07baa2f8d9fd4fbcb9faea451ce05475</code>
	A sample wave file.
<code>sample_2</code>	<code>8f9b378f6d0b42f99522f1173492f013</code>
	Sample text synthesized.

Batch text to speech (speech synthesis)

The easiest way to run batch text-to-speech is to create a new `.tsv` (tab-separated-value) file, and use the `--foreach` command in the Speech CLI. You can create a `.tsv` file using your favorite text editor, for this example, let's call it `text_synthesis.tsv`:

ⓘ Important

When copying the contents of this text file, make sure that your file has a **tab** not spaces between the file location and the text. Sometimes, when copying the contents from this example, tabs are converted to spaces causing the `spx` command to fail when run.

Input

```
audio.output      text
C:\batch_wav_output\wav_1.wav    Sample text to synthesize.
C:\batch_wav_output\wav_2.wav    Using the Speech CLI to run batch-synthesis.
C:\batch_wav_output\wav_3.wav    Some more text to test capabilities.
```

Next, you run a command to point to `text_synthesis.tsv`, perform synthesis on each `text` field, and write the result to the corresponding `audio.output` path as a `.wav` file.

Console

```
spx synthesize --foreach in @C:\your\path\to\text_synthesis.tsv
```

This command is the equivalent of running `spx synthesize --text "Sample text to synthesize" --audio output C:\batch_wav_output\wav_1.wav` for each record in the `.tsv` file.

A couple things to note:

- The column headers, `audio.output` and `text`, correspond to the command-line arguments `--audio output` and `--text`, respectively. Multi-part command-line arguments like `--audio output` should be formatted in the file with no spaces, no leading dashes, and periods separating strings, for example, `audio.output`. Any other existing command-line arguments can be added to the file as additional columns using this pattern.
- When the file is formatted in this way, no additional arguments are required to be passed to `--foreach`.

- Ensure to separate each value in the `.tsv` with a **tab**.

However, if you have a `.tsv` file like the following example, with column headers that **do not match** command-line arguments:

Input

```
wav_path      str_text
C:\batch_wav_output\wav_1.wav    Sample text to synthesize.
C:\batch_wav_output\wav_2.wav    Using the Speech CLI to run batch-synthesis.
C:\batch_wav_output\wav_3.wav    Some more text to test capabilities.
```

You can override these field names to the correct arguments using the following syntax in the `--foreach` call. This is the same call as above.

Console

```
spx synthesize --foreach audio.output;text in
@C:\your\path\to\text_synthesis.tsv
```

Next steps

- [Speech CLI overview](#)
- [Speech CLI quickstart](#)

What is the Speech SDK?

Article • 01/05/2023 • 2 minutes to read

The Speech SDK (software development kit) exposes many of the [Speech service capabilities](#), so you can develop speech-enabled applications. The Speech SDK is available [in many programming languages](#) and across platforms. The Speech SDK is ideal for both real-time and non-real-time scenarios, by using local devices, files, Azure Blob Storage, and input and output streams.

In some cases, you can't or shouldn't use the [Speech SDK](#). In those cases, you can use REST APIs to access the Speech service. For example, use the [Speech-to-text REST API](#) for [batch transcription](#) and [custom speech](#).

Supported languages

The Speech SDK supports the following languages and platforms:

Programming language	Reference	Platform support
C# ¹	.NET	Windows, Linux, macOS, Mono, Xamarin.iOS, Xamarin.Mac, Xamarin.Android, UWP, Unity
C++ ²	C++	Windows, Linux, macOS
Go	Go ↗	Linux
Java	Java	Android, Windows, Linux, macOS
JavaScript	JavaScript	Browser, Node.js
Objective-C	Objective-C	iOS, macOS
Python	Python	Windows, Linux, macOS
Swift	Objective-C ³	iOS, macOS

¹ C# code samples are available in the documentation. The Speech SDK for C# is based on .NET Standard 2.0, so it supports many platforms and programming languages. For more information, see [.NET implementation support](#).

² C isn't a supported programming language for the Speech SDK.

³ The Speech SDK for Swift shares client libraries and reference documentation with the Speech SDK for Objective-C.

Important

By downloading any of the Azure Cognitive Services Speech SDKs, you acknowledge its license. For more information, see:

- [Microsoft software license terms for the Speech SDK ↗](#)
- [Third-party software notices ↗](#)

Speech SDK demo

The following video shows how to install the [Speech SDK for C#](#) and write a simple .NET console application for speech-to-text.

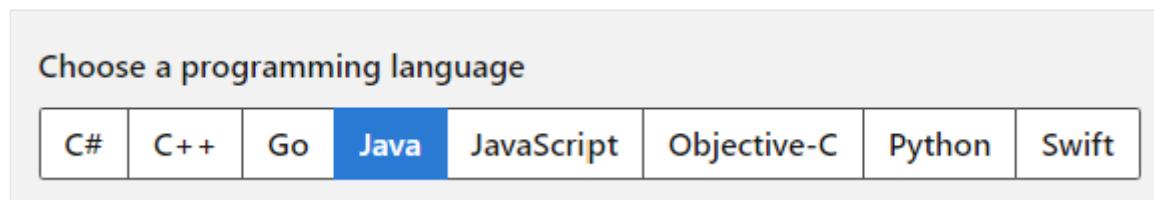
<https://learn-video.azurefd.net/vod/player?id=c20d3b0c-e96a-4154-9299-155e27db7117&locale=en-us&embedUrl=%2Fazure%2Fcognitive-services%2Fspeech-service%2Fspeech-sdk> ↗

Code samples

Speech SDK code samples are available in the documentation and GitHub.

Docs samples

At the top of documentation pages that contain samples, options to select include C#, C++, Go, Java, JavaScript, Objective-C, Python, or Swift.



If a sample is not available in your preferred programming language, you can select another programming language to get started and learn about the concepts, or see the reference and samples linked from the beginning of the article.

GitHub samples

In depth samples are available in the [Azure-Samples/cognitive-services-speech-sdk](#) ↗ repository on GitHub. There are samples for C# (including UWP, Unity, and Xamarin), C++, Java, JavaScript (including Browser and Node.js), Objective-C, Python, and Swift.

Code samples for Go are available in the [Microsoft/cognitive-services-speech-sdk-go](#) repository on GitHub.

Help options

The Microsoft Q&A and [Stack Overflow](#) forums are available for the developer community to ask and answer questions about Azure Cognitive Speech and other services. Microsoft monitors the forums and replies to questions that the community has not yet answered. To make sure that we see your question, tag it with 'azure-speech'.

You can suggest an idea or report a bug by creating an issue on GitHub:

- [Azure-Samples/cognitive-services-speech-sdk](#)
- [Microsoft/cognitive-services-speech-sdk-go](#)
- [Microsoft/cognitive-services-speech-sdk-js](#)

See also [Azure Cognitive Services support and help options](#) to get support, stay up-to-date, give feedback, and report bugs for Cognitive Services.

Next steps

- [Install the SDK](#)
 - [Try the speech to text quickstart](#)
-

Additional resources

Documentation

[How to lower speech synthesis latency using Speech SDK - Azure Cognitive Services](#)

How to lower speech synthesis latency using Speech SDK, including streaming, pre-connection, and so on.

[SpeechRecognizer class](#)

Performs speech recognition from microphone, file, or other audio input streams, and gets transcribed text as result.

[Troubleshoot the Speech SDK - Speech service - Azure Cognitive Services](#)

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

[Asynchronous Conversation Transcription - Speech service - Azure Cognitive Services](#)

Learn how to use asynchronous Conversation Transcription using the Speech service. Available for Java and C# only.

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Show 5 more](#)

Install the Speech SDK

Article • 09/20/2022 • 38 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) ↗ | [Additional Samples on GitHub](#) ↗

This guide shows how to install the [Speech SDK](#) for C#.

Code samples in the documentation are written in C# 8 and run on .NET standard 2.0.

Platform requirements

The Speech SDK for C# is compatible with Windows, Linux, and macOS.

Windows

On Windows, you must use the 64-bit target architecture.

You must install the [Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, 2019, and 2022](#) for your platform. Installing this package for the first time might require a restart.

Install the Speech SDK for C#

The Speech SDK for C# is available as a NuGet package and implements .NET Standard 2.0. For more information, see [Microsoft.CognitiveServices.Speech](#) ↗.

The Speech SDK for C# can be installed from the .NET CLI with the following `dotnet add` command:

.NET CLI

```
dotnet add package Microsoft.CognitiveServices.Speech
```

The Speech SDK for C# can be installed with the following `Install-Package` command:

PowerShell

```
Install-Package Microsoft.CognitiveServices.Speech
```

You can follow a guide below for additional options.

Choose your target environment

.NET

This guide shows how to install the [Speech SDK](#) for a .NET Framework (Windows) console app.

Prerequisites

This guide requires:

- [Microsoft Visual C++ Redistributable for Visual Studio 2019](#) for the Windows platform. Installing it for the first time might require a restart.
- [Visual Studio 2019](#).

Create a Visual Studio project and install the Speech SDK

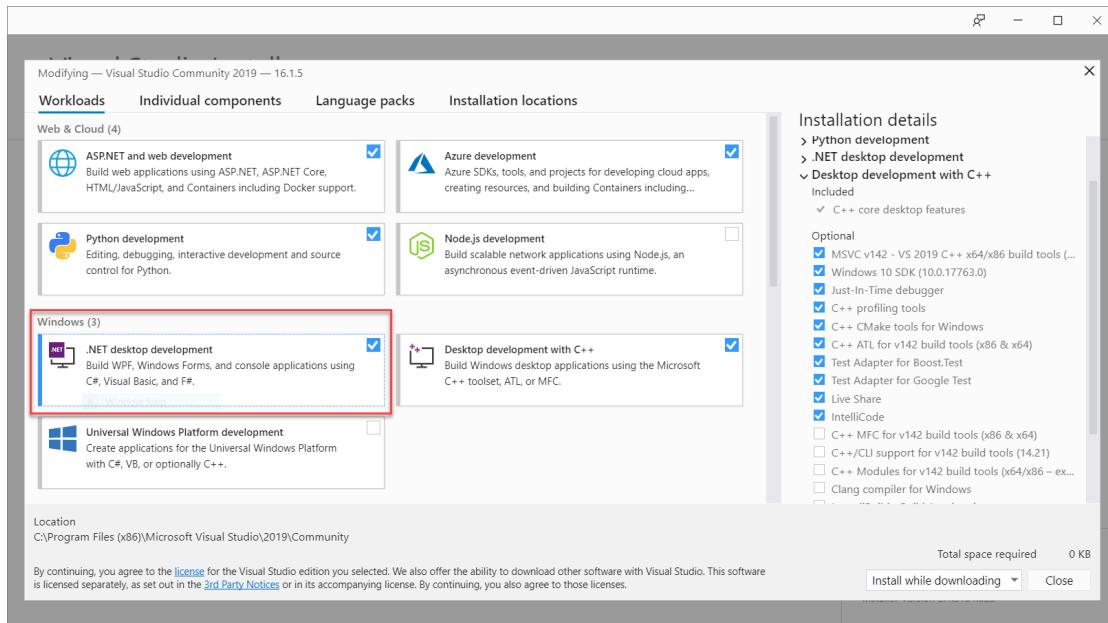
You need to install the [Speech SDK NuGet package](#) so you can reference it in your code. To do that, you might first need to create a **helloworld** project. If you already have a project with the **.NET desktop development** workload available, you can use that project and skip to [Use NuGet Package Manager to install the Speech SDK](#).

Create a 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**. This step opens Visual Studio Installer and displays the **Modifying** dialog.
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 might take a few minutes to download and install.

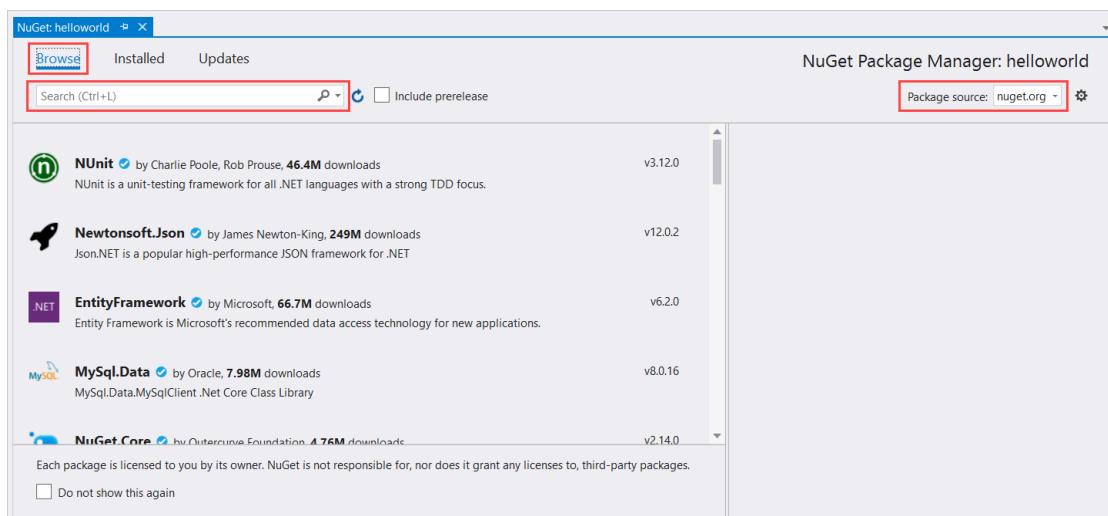
If the check box next to **.NET desktop development** is already selected, select **Close** to close the dialog.



7. Close Visual Studio Installer.

Use NuGet Package Manager to install the Speech SDK

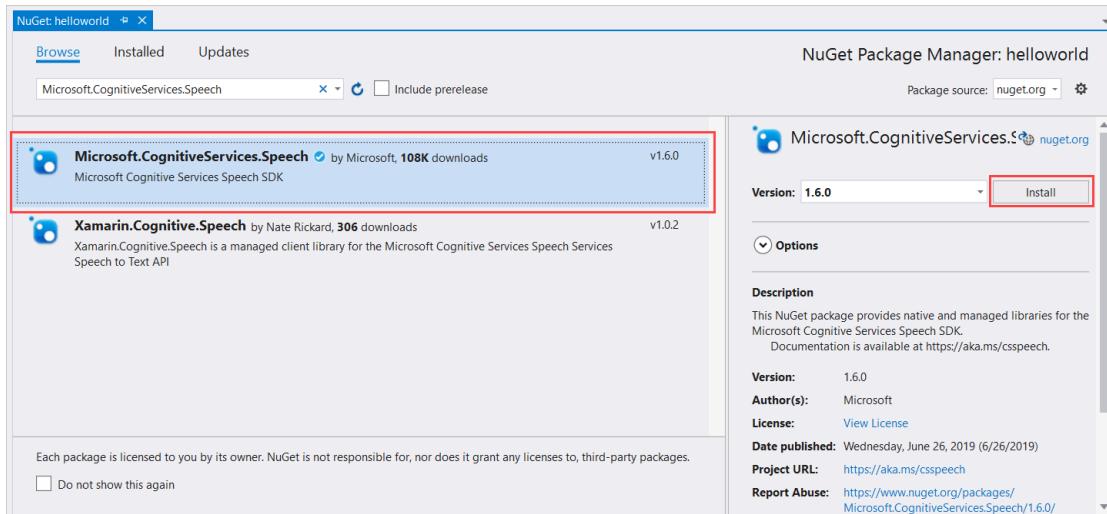
1. In Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show NuGet Package Manager.
2. In the upper-right corner, find the **Package Source** drop-down box, and make sure that **nuget.org** is selected.



3. In the upper-left corner, select **Browse**.

4. In the search box, type **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.



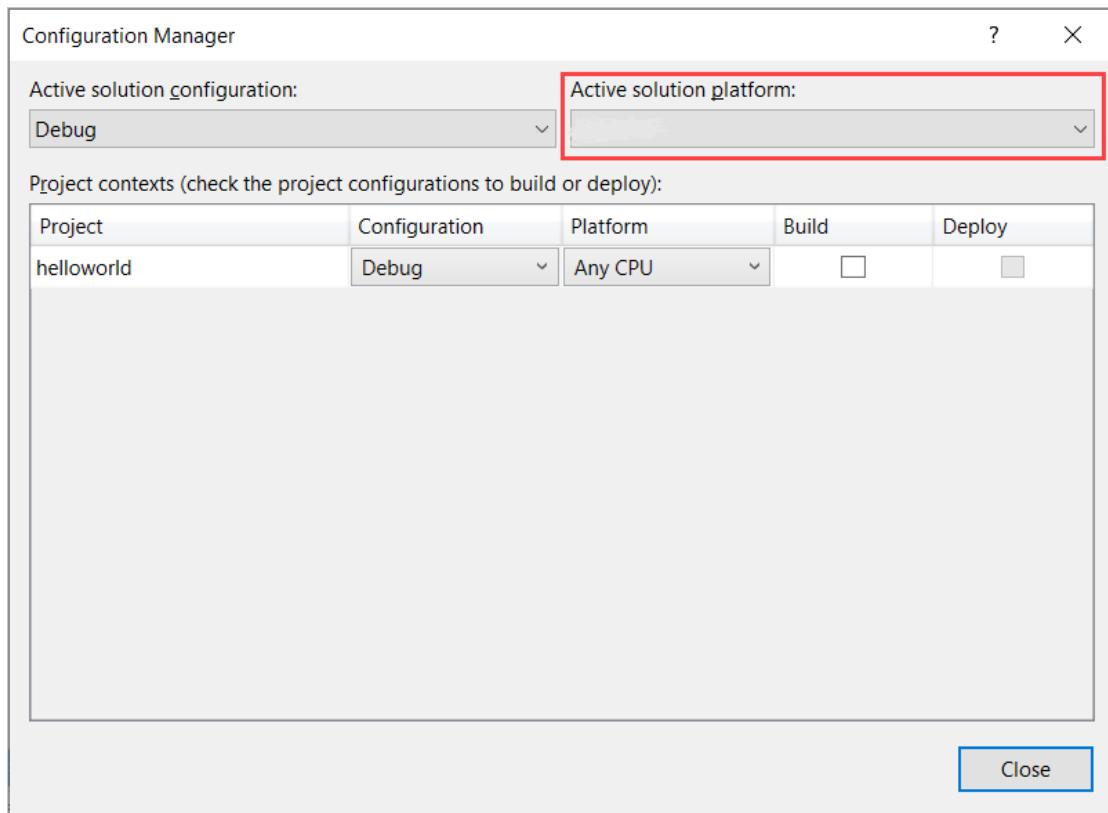
6. Accept all agreements and licenses to start the installation.

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

Choose target architecture

To build and run the console application, create a platform configuration that matches your computer's architecture.

1. From the menu bar, select **Build > Configuration Manager**. The **Configuration Manager** dialog appears.
2. In the **Active solution platform** drop-down box, select **New**. The **New Solution Platform** dialog 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**.

Next steps

- [Speech-to-text quickstart](#)
- [Text-to-speech quickstart](#)
- [Speech translation quickstart](#)

Speech-to-text documentation

Speech-to-text from the Speech service, also known as speech recognition, enables real-time and batch transcription of audio streams into text. With additional reference text input, it also enables real-time pronunciation assessment and gives speakers feedback on the accuracy and fluency of spoken audio.

About speech-to-text

OVERVIEW

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

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

[What is Custom Speech?](#)

[Use the Speech CLI for speech-to-text with no code](#)

QUICKSTART

[Get started with speech-to-text](#)

Develop with speech-to-text

HOW-TO GUIDE

[Choose speech recognition mode](#)

[Improve accuracy with Custom Speech](#)

[Use compressed audio input formats](#)

[Migrate from v3.0 to v3.1](#)

CONCEPT

[Training and testing datasets](#)

[Train a model for Custom Speech](#)

[Create human-labeled transcriptions](#)

 REFERENCE

[Language support](#)

[Speech-to-text FAQ](#)

[Speech-to-text pricing ↗](#)

Help and feedback

 REFERENCE

[Support and help options](#)

What is speech-to-text?

Article • 01/11/2023 • 2 minutes to read

In this overview, you learn about the benefits and capabilities of the speech-to-text feature of the Speech service, which is part of Azure Cognitive Services.

Speech-to-text, also known as speech recognition, enables real-time or offline transcription of audio streams into text. For a full list of available speech-to-text languages, see [Language and voice support for the Speech service](#).

ⓘ Note

Microsoft uses the same recognition technology for Windows and Office products.

Get started

To get started, try the [speech-to-text quickstart](#). Speech-to-text is available via the [Speech SDK](#), the [REST API](#), and the [Speech CLI](#).

In depth samples are available in the [Azure-Samples/cognitive-services-speech-sdk](#) repository on GitHub. There are samples for C# (including UWP, Unity, and Xamarin), C++, Java, JavaScript (including Browser and Node.js), Objective-C, Python, and Swift. Code samples for Go are available in the [Microsoft/cognitive-services-speech-sdk-go](#) repository on GitHub.

Batch transcription

Batch transcription is a set of [Speech-to-text REST API](#) operations that enable you to transcribe a large amount of audio in storage. You can point to audio files with a shared access signature (SAS) URI and asynchronously receive transcription results. For more information on how to use the batch transcription API, see [How to use batch transcription](#) and [Batch transcription samples \(REST\)](#).

Custom Speech

The Azure speech-to-text service analyzes audio in real-time or batch to transcribe the spoken word into text. Out of the box, speech to text utilizes a Universal Language Model as a base model that is trained with Microsoft-owned data and reflects commonly used spoken language. This base model is pre-trained with dialects and

phonetics representing a variety of common domains. The base model works well in most scenarios.

The base model may not be sufficient if the audio contains ambient noise or includes a lot of industry and domain-specific jargon. In these cases, building a custom speech model makes sense by training with additional data associated with that specific domain. You can create and train custom acoustic, language, and pronunciation models. For more information, see [Custom Speech](#) and [Speech-to-text REST API](#).

Customization options vary by language or locale. To verify support, see [Language and voice support for the Speech service](#).

Next steps

- [Get started with speech-to-text](#)
- [Get the Speech SDK](#)

Quickstart: Recognize and convert speech to text

Article • 09/20/2022 • 43 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this quickstart, you run an application to recognize and transcribe human speech (often called speech-to-text).

💡 Tip

You can try speech-to-text in [Speech Studio](#) without signing up or writing any code.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

[I ran into an issue](#)

Set up the environment

The Speech SDK is available as a [NuGet package](#) and implements .NET Standard 2.0. You install the Speech SDK later in this guide, but first check the [SDK installation guide](#) for any more requirements.

Set environment variables

Your application must be authenticated to access Cognitive Services resources. For production, use a secure way of storing and accessing your credentials. For example, after you [get a key for your Speech resource](#), write it to a new environment variable on the local machine running the application.

💡 Tip

Don't include the key directly in your code, and never post it publicly. See the Cognitive Services [security](#) article for more authentication options like [Azure Key Vault](#).

To set the environment variable for your Speech resource key, open a console window, and follow the instructions for your operating system and development environment. To set the `SPEECH_KEY` environment variable, replace `your-key` with one of the keys for your resource.

Windows

Console

```
setx SPEECH_KEY your-key
```

 **Note**

If you only need to access the environment variable in the current running console, you can set the environment variable with `set` instead of `setx`.

After you add the environment variable, you may need to restart any running programs that will need to read the environment variable, including the console window. For example, if you are using Visual Studio as your editor, restart Visual Studio before running the example.

To set the environment variable for your Speech resource region, follow the same steps. Set `SPEECH_REGION` to the region of your resource. For example, `westus`.

I ran into an issue

Recognize speech from a microphone

Follow these steps to create a new console application and install the Speech SDK.

1. Open a command prompt where you want the new project, and create a console application with the .NET CLI. The `Program.cs` file should be created in the project directory.

.NET CLI

```
dotnet new console
```

2. Install the Speech SDK in your new project with the .NET CLI.

NET CLI

```
dotnet add package Microsoft.CognitiveServices.Speech
```

3. Replace the contents of `Program.cs` with the following code.

C#

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

class Program
{
    // This example requires environment variables named "SPEECH_KEY"
    and "SPEECH_REGION"
    static string speechKey =
    Environment.GetEnvironmentVariable("SPEECH_KEY");
    static string speechRegion =
    Environment.GetEnvironmentVariable("SPEECH_REGION");

    static void OutputSpeechRecognitionResult(SpeechRecognitionResult
speechRecognitionResult)
    {
        switch (speechRecognitionResult.Reason)
        {
            case ResultReason.RecognizedSpeech:
                Console.WriteLine($"RECOGNIZED: Text=
{speechRecognitionResult.Text}");
                break;
            case ResultReason.NoMatch:
                Console.WriteLine($"NOMATCH: Speech could not be
recognized.");
                break;
            case ResultReason.Canceled:
                var cancellation =
                CancellationDetails.FromResult(speechRecognitionResult);
                Console.WriteLine($"CANCELED: Reason=
{cancellation.Reason}");

                if (cancellation.Reason == CancellationReason.Error)
                {
                    Console.WriteLine($"CANCELED: ErrorCode=
{cancellation.ErrorCode}");
                    Console.WriteLine($"CANCELED: ErrorDetails=
{cancellation.ErrorDetails}");
                }
        }
    }
}
```

```
{cancellation.ErrorDetails});  
                Console.WriteLine($"CANCELED: Did you set the  
speech resource key and region values?");  
            }  
            break;  
        }  
  
    async static Task Main(string[] args)  
    {  
        var speechConfig = SpeechConfig.FromSubscription(speechKey,  
speechRegion);  
        speechConfig.SpeechRecognitionLanguage = "en-US";  
  
        using var audioConfig =  
AudioConfig.FromDefaultMicrophoneInput();  
        using var speechRecognizer = new SpeechRecognizer(speechConfig,  
audioConfig);  
  
        Console.WriteLine("Speak into your microphone.");  
        var speechRecognitionResult = await  
speechRecognizer.RecognizeOnceAsync();  
        OutputSpeechRecognitionResult(speechRecognitionResult);  
    }  
}
```

4. To change the speech recognition language, replace `en-US` with another [supported language](#). For example, `es-ES` for Spanish (Spain). The default language is `en-US` if you don't specify a language. For details about how to identify one of multiple languages that might be spoken, see [language identification](#).

Run your new console application to start speech recognition from a microphone:

```
Console
```

```
dotnet run
```

ⓘ Important

Make sure that you set the `SPEECH_KEY` and `SPEECH_REGION` environment variables as described [above](#). If you don't set these variables, the sample will fail with an error message.

Speak into your microphone when prompted. What you speak should be output as text:

```
Console
```

Speak into your microphone.
RECOGNIZED: Text=I'm excited to try speech to text.

I ran into an issue

Remarks

Now that you've completed the quickstart, here are some additional considerations:

- This example uses the `RecognizeOnceAsync` operation to transcribe utterances of up to 30 seconds, or until silence is detected. For information about continuous recognition for longer audio, including multi-lingual conversations, see [How to recognize speech](#).
- To recognize speech from an audio file, use `FromWavFileInput` instead of `FromDefaultMicrophoneInput`:

C#

```
using var audioConfig =
    AudioConfig.FromWavFileInput("YourAudioFile.wav");
```

- For compressed audio files such as MP4, install GStreamer and use `PullAudioInputStream` or `PushAudioInputStream`. For more information, see [How to use compressed input audio](#).

Clean up resources

You can use the [Azure portal](#) or [Azure Command Line Interface \(CLI\)](#) to remove the Speech resource you created.

Next steps

[Learn more about speech recognition](#)

How to recognize speech

Article • 09/20/2022 • 34 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this how-to guide, you learn how to recognize and transcribe human speech (often called speech-to-text).

Create a speech configuration

To call the Speech service by using the Speech SDK, you need to create a `SpeechConfig` instance. This class includes information about your subscription, like your key and associated location/region, endpoint, host, or authorization token.

Create a `SpeechConfig` instance by using your key and location/region. Create a Speech resource on the [Azure portal](#). For more information, see [Create a new Azure Cognitive Services resource](#).

C#

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

class Program
{
    async static Task Main(string[] args)
    {
        var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
    }
}
```

You can initialize `SpeechConfig` in a few other ways:

- With an endpoint: pass in a Speech service endpoint. A key or authorization token is optional.
- With a host: pass in a host address. A key or authorization token is optional.
- With an authorization token: pass in an authorization token and the associated region/location.

 Note

Regardless of whether you're performing speech recognition, speech synthesis, translation, or intent recognition, you'll always create a configuration.

Recognize speech from a microphone

To recognize speech by using your device microphone, create an [AudioConfig](#) instance by using `FromDefaultMicrophoneInput()`. Then initialize [SpeechRecognizer](#) by passing `audioConfig` and `speechConfig`.

C#

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

class Program
{
    async static Task FromMic(SpeechConfig speechConfig)
    {
        using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();
        using var recognizer = new SpeechRecognizer(speechConfig,
audioConfig);

        Console.WriteLine("Speak into your microphone.");
        var result = await recognizer.RecognizeOnceAsync();
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    }

    async static Task Main(string[] args)
    {
        var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
        await FromMic(speechConfig);
    }
}
```

If you want to use a *specific* audio input device, you need to specify the device ID in `AudioConfig`. Learn [how to get the device ID](#) for your audio input device.

Recognize speech from a file

If you want to recognize speech from an audio file instead of a microphone, you still need to create an `AudioConfig` instance. But instead of calling `FromDefaultMicrophoneInput()`, you call `FromWavFileInput()` and pass the file path:

C#

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

class Program
{
    async static Task FromFile(SpeechConfig speechConfig)
    {
        using var audioConfig =
    AudioConfig.FromWavFileInput("PathToFile.wav");
        using var recognizer = new SpeechRecognizer(speechConfig,
audioConfig);

        var result = await recognizer.RecognizeOnceAsync();
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    }

    async static Task Main(string[] args)
    {
        var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
        await FromFile(speechConfig);
    }
}
```

Recognize speech from an in-memory stream

For many use cases, it's likely that your audio data will come from blob storage or will otherwise already be in memory as a `byte[]` instance or a similar raw data structure. The following example uses `PushAudioInputStream` to recognize speech, which is essentially an abstracted memory stream. The sample code does the following:

- Writes raw audio data (PCM) to `PushAudioInputStream` by using the `Write()` function, which accepts a `byte[]` instance.
- Reads a `.wav` file by using `FileReader` for demonstration purposes. If you already have audio data in a `byte[]` instance, you can skip directly to writing the content to the input stream.
- The default format is 16-bit, 16-KHz mono PCM. To customize the format, you can pass an `AudioStreamFormat` object to `CreatePushStream()` by using the static function `AudioStreamFormat.GetWaveFormatPCM(sampleRate, (byte)bitRate, (byte)channels)`.

C#

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

class Program
{
    async static Task FromStream(SpeechConfig speechConfig)
    {
        var reader = new BinaryReader(File.OpenRead("PathToFile.wav"));
        using var audioInputStream = AudioInputStream.CreatePushStream();
        using var audioConfig =
            AudioConfig.FromStreamInput(audioInputStream);
        using var recognizer = new SpeechRecognizer(speechConfig,
            audioConfig);

        byte[] readBytes;
        do
        {
            readBytes = reader.ReadBytes(1024);
            audioInputStream.Write(readBytes, readBytes.Length);
        } while (readBytes.Length > 0);

        var result = await recognizer.RecognizeOnceAsync();
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    }

    async static Task Main(string[] args)
    {
        var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
            "YourSpeechRegion");
        await FromStream(speechConfig);
    }
}
```

Using a push stream as input assumes that the audio data is a raw PCM and skips any headers. The API will still work in certain cases if the header has not been skipped. But for the best results, consider implementing logic to read off the headers so that `byte[]` starts at the *start of the audio data*.

Handle errors

The previous examples simply get the recognized text from `result.text`. To handle errors and other responses, you need to write some code to handle the result. The following code evaluates the `result.Reason` property and:

- Prints the recognition result: `ResultReason.RecognizedSpeech`.
- If there is no recognition match, informs the user: `ResultReason.NoMatch`.
- If an error is encountered, prints the error message: `ResultReason.Canceled`.

C#

```
switch (result.Reason)
{
    case ResultReason.RecognizedSpeech:
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        break;
    case ResultReason.NoMatch:
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
        break;
    case 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 set the speech resource
key and region values?");
        }
        break;
}
```

Use continuous recognition

The previous examples use single-shot recognition, which recognizes a single utterance. 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.

In contrast, you use continuous recognition when you want to control when to stop recognizing. It requires you to subscribe to the `Recognizing`, `Recognized`, and `Canceled` events to get the recognition results. To stop recognition, you must call `StopContinuousRecognitionAsync`. Here's an example of how continuous recognition is performed on an audio input file.

Start by defining the input and initializing `SpeechRecognizer`:

C#

```
using var audioConfig = AudioConfig.FromWavFileInput("YourAudioFile.wav");
using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
```

Then create a `TaskCompletionSource<int>` instance to manage the state of speech recognition:

C#

```
var stopRecognition = new TaskCompletionSource<int>();
```

Next, subscribe to the events that `SpeechRecognizer` sends:

- **Recognizing**: Signal for events that contain intermediate recognition results.
- **Recognized**: Signal for events that contain final recognition results, which indicate a successful recognition attempt.
- **SessionStopped**: Signal for events that indicate the end of a recognition session (operation).
- **Canceled**: Signal for events that contain canceled recognition results. These results indicate a recognition attempt that was canceled as a result or a direct cancellation request. Alternatively, they indicate a transport or protocol failure.

C#

```
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)
    {
        Console.WriteLine($"CANCELED: ErrorCode={e.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={e.ErrorDetails}");
    }
};
```

```
        Console.WriteLine($"CANCELED: Did you set the speech resource key  
and region values?");  
    }  
  
    stopRecognition.TrySetResult(0);  
};  
  
recognizer.SessionStopped += (s, e) =>  
{  
    Console.WriteLine("\n      Session stopped event.");  
    stopRecognition.TrySetResult(0);  
};
```

With everything set up, call `StartContinuousRecognitionAsync` to start recognizing:

C#

```
await recognizer.StartContinuousRecognitionAsync();  
  
// Waits for completion. Use Task.WaitAny to keep the task rooted.  
Task.WaitAny(new[] { stopRecognition.Task });  
  
// Make the following call at some point to stop recognition:  
// await recognizer.StopContinuousRecognitionAsync();
```

Change the source language

A common task for speech recognition is specifying the input (or source) language. The following example shows how you would change the input language to Italian. In your code, find your `SpeechConfig` instance and add this line directly below it:

C#

```
speechConfig.SpeechRecognitionLanguage = "it-IT";
```

The `SpeechRecognitionLanguage` property expects a language-locale format string. Refer to the [list of supported speech-to-text locales](#).

Use a custom endpoint

With [Custom Speech](#), you can upload your own data, test and train a custom model, compare accuracy between models, and deploy a model to a custom endpoint. The following example shows how to set a custom endpoint.

C#

```
var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey",  
    "YourServiceRegion");  
speechConfig.EndpointId = "YourEndpointId";  
var speechRecognizer = new SpeechRecognizer(speechConfig);
```

Change how silence is handled

If a user is expected to speak faster or slower than usual, the default behaviors for non-speech silence in input audio may not result in what you expect. Common problems with silence handling include:

- Fast speech chaining many sentences together into a single recognition result instead of breaking sentences into individual results
- Slow speech separating parts of a single sentence into multiple results
- A single-shot recognition ending too quickly while waiting for speech to begin

These problems can be addressed by setting one of two *timeout properties* on the `SpeechConfig` used to create a `SpeechRecognizer`:

- **Segmentation silence timeout** adjusts how much non-speech audio is allowed within a phrase that's currently being spoken before that phrase is considered "done."
 - *Higher* values generally make results longer and allow longer pauses from the speaker within a phrase, but will make results take longer to arrive can also make separate phrases combine together into a single result when set too high
 - *Lower* values generally make results shorter and ensure more prompt and frequent breaks between phrases, but can also cause single phrases to separate into multiple results when set too low
 - This timeout can be set to integer values between 100 and 5000, in milliseconds, with 500 a typical default
- **Initial silence timeout** adjusts how much non-speech audio is allowed *before* a phrase before the recognition attempt ends in a "no match" result.
 - *Higher* values give speakers more time to react and start speaking, but can also result in slow responsiveness when nothing is spoken
 - *Lower* values ensure a prompt "no match" for faster user experience and more controlled audio handling, but may cut a speaker off too quickly when set too low
 - Because continuous recognition generates many results, this value determines how often "no match" results will arrive but doesn't otherwise affect the content of recognition results

- This timeout can be set to any non-negative integer value, in milliseconds, or set to 0 to disable it entirely; 5000 is a typical default for single-shot recognition while 15000 is a typical default for continuous recognition

As there are tradeoffs when modifying these timeouts, it's only recommended to change the settings when a problem related to silence handling is observed. Default values optimally handle the majority of spoken audio and only uncommon scenarios should encounter problems.

Example: users speaking a serial number like "ABC-123-4567" pause between character groups long enough for the serial number to be broken into multiple results. In this case, setting the segmentation silence timeout to a higher value like 2000ms could help:

```
C#
```

```
speechConfig SetProperty(PropertyId.Speech_SegmentationSilenceTimeoutMs,  
"2000");
```

Example: a recorded presenter's speech is fast enough that several sentences in a row get combined, with big recognition results only arriving once or twice per minute. In this case, setting the segmentation silence timeout to a lower value like 300ms could help:

```
C#
```

```
speechConfig SetProperty(PropertyId.Speech_SegmentationSilenceTimeoutMs,  
"300");
```

Example: a single-shot recognition asking a speaker to find and read a serial number ends too quickly while the number is being found. In this case, a longer initial silence timeout like 10000ms could help:

```
C#
```

```
speechConfig SetProperty(PropertyId.SpeechServiceConnection_InitialSilenceTi  
meoutMs, "10000");
```

Next steps

- [Try the speech to text quickstart](#)
- [Improve recognition accuracy with custom speech](#)
- [Use batch transcription](#)

Get speech recognition results

Article • 06/13/2022 • 28 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this how-to guide, you learn about how you can use speech recognition results.

Speech synchronization

You might want to synchronize transcriptions with an audio track, whether it's done in real time or with a prerecording.

The Speech service returns the offset and duration of the recognized speech.

- **Offset:** The offset into the audio stream being recognized, expressed as duration. Offset is measured in ticks, starting from 0 (zero) tick, associated with the first audio byte processed by the SDK. For example, the offset begins when you start recognition, since that's when the SDK starts processing the audio stream. One tick represents one hundred nanoseconds or one ten-millionth of a second.
- **Duration:** Duration of the utterance that is being recognized. The duration in ticks doesn't include trailing or leading silence.

The end of a single utterance is determined by listening for silence at the end. You won't get the final recognition result until an utterance has completed. Recognizing events will provide intermediate results that are subject to change while an audio stream is being processed. Recognized events will provide the final transcribed text once processing of an utterance is completed.

Recognizing offset and duration

With the `Recognizing` event, you can get the offset and duration of the speech being recognized. Offset and duration per word are not available while recognition is in progress. Each `Recognizing` event comes with a textual estimate of the speech recognized so far.

This code snippet shows how to get the offset and duration from a `Recognizing` event.

C#

```
speechRecognizer.Recognizing += (object sender, SpeechRecognitionEventArgs e) =>
{
    {
        // Your code here
    }
}
```

```

        if (e.Result.Reason == ResultReason.RecognizingSpeech)
    {
        Console.WriteLine(String.Format ("RECOGNIZING: {0}",
e.Result.Text));
        Console.WriteLine(String.Format ("Offset in Ticks: {0}",
e.Result.OffsetInTicks));
        Console.WriteLine(String.Format ("Duration in Ticks: {0}",
e.Result.Duration.Ticks));
    }
};

```

Recognized offset and duration

Once an utterance has been recognized, you can get the offset and duration of the recognized speech. With the `Recognized` event, you can also get the offset and duration per word. To request the offset and duration per word, first you must set the corresponding `SpeechConfig` property as shown here:

```
C#
speechConfig.RequestWordLevelTimestamps();
```

This code snippet shows how to get the offset and duration from a `Recognized` event.

```
C#
speechRecognizer.Recognized += (object sender, SpeechRecognitionEventArgs e) =>
{
    if (ResultReason.RecognizedSpeech == e.Result.Reason && e.Result.Text.Length > 0)
    {
        Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
        Console.WriteLine(String.Format ("Offset in Ticks: {0}",
e.Result.OffsetInTicks));
        Console.WriteLine(String.Format ("Duration in Ticks: {0}",
e.Result.Duration.Ticks));

        var detailedResults = speechRecognitionResult.Best();
        if(detailedResults != null && detailedResults.Any())
        {
            // The first item in detailedResults corresponds to the
            // recognized text.
            // This is not necessarily the item with the highest
            // confidence number.
            var bestResults = detailedResults?.ToList()[0];
            Console.WriteLine(String.Format ("\tConfidence: {0}\n\tText:
{1}\n\tLexicalForm: {2}\n\tNormalizedForm: {3}\n\tMaskedNormalizedForm:
{4}",
```

```

                bestResults.Confidence, bestResults.Text,
bestResults.LexicalForm, bestResults.NormalizedForm,
bestResults.MaskedNormalizedForm));
        // You must set speechConfig.RequestWordLevelTimestamps() to
        // get word-level timestamps.
        Console.WriteLine($"\\tWord-level timing:");
        Console.WriteLine($"\\t\\tWord | Offset | Duration");
        Console.WriteLine($"\\t\\t----- | ----- | ----- ");

        foreach (var word in bestResults.Words)
        {
            Console.WriteLine($"\\t\\t{word.Word} | {word.Offset} |
{word.Duration}");
        }
    }
};


```

Example offset and duration

The following table shows potential offset and duration in ticks when a speaker says "Welcome to Applied Mathematics course 201." In this example, the offset doesn't change throughout the `Recognizing` and `Recognized` events. However, don't rely on the offset to remain the same between the `Recognizing` and `Recognized` events, since the final result could be different.

Event	Text	Offset (in ticks)	Duration (in ticks)
RECOGNIZING	welcome	17000000	5000000
RECOGNIZING	welcome to	17000000	6400000
RECOGNIZING	welcome to applied math	17000000	13600000
RECOGNIZING	welcome to applied mathematics	17000000	17200000
RECOGNIZING	welcome to applied mathematics course	17000000	23700000
RECOGNIZING	welcome to applied mathematics course 2	17000000	26700000
RECOGNIZING	welcome to applied mathematics course 201	17000000	33400000
RECOGNIZED	Welcome to applied Mathematics course 201.	17000000	34500000

The total duration of the first utterance was 3.45 seconds. It was recognized at 1.7 to 5.15 seconds offset from the start of the audio stream being recognized (00:00:01.700 --

> 00:00:05.150).

If the speaker continues then to say "Let's get started," a new offset is calculated from the start of the audio stream being recognized, to the start of the new utterance. The following table shows potential offset and duration for an utterance that started two seconds after the previous utterance ended.

Event	Text	Offset (in ticks)	Duration (in ticks)
RECOGNIZING	OK	71500000	3100000
RECOGNIZING	OK now	71500000	10300000
RECOGNIZING	OK, now let's	71500000	14700000
RECOGNIZING	OK, now let's get started.	71500000	18500000
RECOGNIZED	OK, now let's get started.	71500000	20600000

The total duration of the second utterance was 2.06 seconds. It was recognized at 7.15 to 9.21 seconds offset from the start of the audio stream being recognized (00:00:07.150 --> 00:00:09.210).

Next steps

- [Try the speech to text quickstart](#)
- [Improve recognition accuracy with custom speech](#)
- [Use batch transcription](#)

Display text formatting with speech to text

Article • 09/22/2022 • 3 minutes to read

Speech-to-text offers an array of formatting features to ensure that the transcribed text is clear and legible. Below is an overview of these features and how each one is used to improve the overall clarity of the final text output.

ITN

Inverse Text Normalization (ITN) is a process that converts spoken words into their written form. For example, the spoken word "four" is converted to the written form "4". This process is performed by the speech-to-text service and isn't configurable. Some of the supported text formats include dates, times, decimals, currencies, addresses, emails, and phone numbers. You can speak naturally, and the service formats text as expected. The following table shows the ITN rules that are applied to the text output.

Recognized speech	Display text
that will cost nine hundred dollars	That will cost \$900.
my phone number is one eight hundred, four five six, eight nine ten	My phone number is 1-800-456-8910.
the time is six forty five p m	The time is 6:45 PM.
I live on thirty five lexington avenue	I live on 35 Lexington Ave.
the answer is six point five	The answer is 6.5.
send it to support at help dot com	Send it to support@help.com.

Capitalization

Speech-to-text models recognize words that should be capitalized to improve readability, accuracy, and grammar. For example, the Speech service will automatically capitalize proper nouns and words at the beginning of a sentence. Some examples are shown in this table.

Recognized speech	Display text
-------------------	--------------

Recognized speech	Display text
i got an x l t shirt	I got an XL t-shirt.
my name is jennifer smith	My name is Jennifer Smith.
i want to visit new york city	I want to visit New York City.

Disfluency removal

When speaking, it's common for someone to stutter, duplicate words, and say filler words like "uhm" or "uh". Speech-to-text can recognize such disfluencies and remove them from the display text. Disfluency removal is great for transcribing live unscripted speeches to read them back later. Some examples are shown in this table.

Recognized speech	Display text
i uh said that we can go to the uhmm movies	I said that we can go to the movies.
its its not that big of uhm a deal	It's not that big of a deal.
umm i think tomorrow should work	I think tomorrow should work.

Punctuation

Speech-to-text automatically punctuates your text to improve clarity. Punctuation is helpful for reading back call or conversation transcriptions. Some examples are shown in this table.

Recognized speech	Display text
how are you	How are you?
we can go to the mall park or beach	We can go to the mall, park, or beach.

When you're using speech-to-text with continuous recognition, you can configure the Speech service to recognize explicit punctuation marks. Then you can speak punctuation aloud in order to make your text more legible. This is especially useful in a situation where you want to use complex punctuation without having to merge it later. Some examples are shown in this table.

Recognized speech	Display text
they entered the room dot dot dot	They entered the room...

Recognized speech	Display text
i heart emoji you period	I <3 you.
the options are apple forward slash banana forward slash orange period	The options are apple/banana/orange.
are you sure question mark	Are you sure?

Use the Speech SDK to enable dictation mode when you're using speech-to-text with continuous recognition. This mode will cause the speech configuration instance to interpret word descriptions of sentence structures such as punctuation.

```
C#
```

```
speechConfig.EnableDictation();
```

Profanity filter

You can specify whether to mask, remove, or show profanity in the final transcribed text. Masking replaces profane words with asterisk (*) characters so that you can keep the original sentiment of your text while making it more appropriate for certain situations

 **Note**

Microsoft also reserves the right to mask or remove any word that is deemed inappropriate. Such words will not be returned by the Speech service, whether or not you enabled profanity filtering.

The profanity filter options are:

- **Masked**: Replaces letters in profane words with asterisk (*) characters. Masked is the default option.
- **Raw**: Include the profane words verbatim.
- **Removed**: Removes profane words.

For example, to remove profane words from the speech recognition result, set the profanity filter to **Removed** as shown here:

```
C#
```

```
speechConfig.SetProfanity(ProfanityOption.Removed);
```

Profanity filter is applied to the result `Text` and `MaskedNormalizedForm` properties.

Profanity filter isn't applied to the result `LexicalForm` and `NormalizedForm` properties.

Neither is the filter applied to the word level results.

Next steps

- [Speech-to-text quickstart](#)
- [Get speech recognition results](#)

What is batch transcription?

Article • 10/23/2022 • 2 minutes to read

Batch transcription is used to transcribe a large amount of audio data in storage. Both the [Speech-to-text REST API](#) and [Speech CLI](#) support batch transcription.

ⓘ Note

To use batch transcription, you need a standard Speech resource (S0) in your subscription. Free resources (F0) aren't supported. For more information, see [pricing and limits ↗](#).

You should provide multiple files per request or point to an Azure Blob Storage container with the audio files to transcribe. The batch transcription service can handle a large number of submitted transcriptions. The service transcribes the files concurrently, which reduces the turnaround time.

How does it work?

With batch transcriptions, you submit the audio data, and then retrieve transcription results asynchronously. The service transcribes the audio data and stores the results in a storage container. You can then retrieve the results from the storage container.

To get started with batch transcription, refer to the following how-to guides:

1. [Locate audio files for batch transcription](#) - You can upload your own data or use existing audio files via public URI or [shared access signature \(SAS\)](#) URI.
2. [Create a batch transcription](#) - Submit the transcription job with parameters such as the audio files, the transcription language, and the transcription model.
3. [Get batch transcription results](#) - Check transcription status and retrieve transcription results asynchronously.

Batch transcription jobs are scheduled on a best-effort basis. You can't estimate when a job will change into the running state, but it should happen within minutes under normal system load. When the job is in the running state, the transcription occurs faster than the audio runtime playback speed.

Next steps

- [Locate audio files for batch transcription](#)

Locate audio files for batch transcription

Article • 11/15/2022 • 9 minutes to read

Batch transcription is used to transcribe a large amount of audio in storage. Batch transcription can access audio files from inside or outside of Azure.

When source audio files are stored outside of Azure, they can be accessed via a public URI (such as "<https://crbn.us/hello.wav>"). Files should be directly accessible; URLs that require authentication or that invoke interactive scripts before the file can be accessed aren't supported.

Audio files that are stored in Azure Blob storage can be accessed via one of two methods:

- [Trusted Azure services security mechanism](#)
- [Shared access signature \(SAS\) URI](#).

You can specify one or multiple audio files when creating a transcription. We recommend that you provide multiple files per request or point to an Azure Blob storage container with the audio files to transcribe. The batch transcription service can handle a large number of submitted transcriptions. The service transcribes the files concurrently, which reduces the turnaround time.

Supported audio formats

The batch transcription API supports the following formats:

Format	Codec	Bits per sample	Sample rate
WAV	PCM	16-bit	8 kHz or 16 kHz, mono or stereo
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. A JSON result file is created for each input audio file. To create an ordered final transcript, use the timestamps that are generated per utterance.

Azure Blob Storage upload

When audio files are located in an [Azure Blob Storage](#) account, you can request transcription of individual audio files or an entire Azure Blob Storage container. You can also [write transcription results](#) to a Blob container.

 **Note**

For blob and container limits, see [batch transcription quotas and limits](#).

Azure portal

Follow these steps to create a storage account and upload wav files from your local directory to a new container.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. [Create a Storage account resource](#) in the Azure portal. Use the same subscription and resource group as your Speech resource.
3. Select the Storage account.
4. In the **Data storage** group in the left pane, select **Containers**.
5. Select **+ Container**.
6. Enter a name for the new container and select **Create**.
7. Select the new container.
8. Select **Upload**.
9. Choose the files to upload and select **Upload**.

Trusted Azure services security mechanism

This section explains how to set up and limit access to your batch transcription source audio files in an Azure Storage account using the [trusted Azure services security mechanism](#).

 **Note**

With the trusted Azure services security mechanism, you need to use [Azure Blob storage](#) to store audio files. Usage of [Azure Files](#) is not supported.

If you perform all actions in this section, your Storage account will be in the following configuration:

- Access to all external network traffic is prohibited.

- Access to Storage account using Storage account key is prohibited.
- Access to Storage account blob storage using [shared access signatures \(SAS\)](#) is prohibited.
- Access to the selected Speech resource is allowed using the [resource system assigned managed identity](#).

So in effect your Storage account becomes completely "locked" and can't be used in any scenario apart from transcribing audio files that were already present by the time the new configuration was applied. You should consider this configuration as a model as far as the security of your audio data is concerned and customize it according to your needs.

For example, you may allow traffic from selected public IP addresses and Azure Virtual networks. You may also set up access to your Storage account using [private endpoints](#) (see as well [this tutorial](#)), re-enable access using Storage account key, allow access to other Azure trusted services, etc.

Note

Using [private endpoints for Speech](#) isn't required to secure the storage account. You can use a private endpoint for batch transcription API requests, while separately accessing the source audio files from a secure storage account, or the other way around.

By following the steps below, you'll severely restrict access to the storage account. Then you'll assign the minimum required permissions for Speech resource managed identity to access the Storage account.

Enable system assigned managed identity for the Speech resource

Follow these steps to enable system assigned managed identity for the Speech resource that you will use for batch transcription.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Speech resource.
3. In the **Resource Management** group in the left pane, select **Identity**.
4. On the **System assigned** tab, select **On** for the status.

ⓘ Important

User assigned managed identity won't meet requirements for the batch transcription storage account scenario. Be sure to enable system assigned managed identity.

5. Select Save

Now the managed identity for your Speech resource can be granted access to your storage account.

Restrict access to the storage account

Follow these steps to restrict access to the storage account.

ⓘ Important

Upload audio files in a Blob container before locking down the storage account access.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Storage account.
3. In the **Settings** group in the left pane, select **Configuration**.
4. Select **Disabled** for **Allow Blob public access**.
5. Select **Disabled** for **Allow storage account key access**
6. Select **Save**.

For more information, see [Prevent anonymous public read access to containers and blobs](#) and [Prevent Shared Key authorization for an Azure Storage account](#).

Configure Azure Storage firewall

Having restricted access to the Storage account, you need to grant access to specific managed identities. Follow these steps to add access for the Speech resource.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Storage account.
3. In the **Security + networking** group in the left pane, select **Networking**.

4. In the Firewalls and virtual networks tab, select **Enabled** from selected virtual networks and IP addresses.
5. Deselect all check boxes.
6. Make sure **Microsoft network routing** is selected.
7. Under the **Resource instances** section, select **Microsoft.CognitiveServices/accounts** as the resource type and select your Speech resource as the instance name.
8. Select **Save**.

 **Note**

It may take up to 5 min for the network changes to propagate.

Although by now the network access is permitted, the Speech resource can't yet access the data in the Storage account. You need to assign a specific access role for Speech resource managed identity.

Assign resource access role

Follow these steps to assign the **Storage Blob Data Reader** role to the managed identity of your Speech resource.

 **Important**

You need to be assigned the *Owner* role of the Storage account or higher scope (like Subscription) to perform the operation in the next steps. This is because only the *Owner* role can assign roles to others. See details [here](#).

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Storage account.
3. Select **Access Control (IAM)** menu in the left pane.
4. Select **Add role assignment** in the **Grant access to this resource** tile.
5. Select **Storage Blob Data Reader** under **Role** and then select **Next**.
6. Select **Managed identity** under **Members > Assign access to**.

7. Assign the managed identity of your Speech resource and then select **Review + assign**.

The screenshot shows the 'Add role assignment' page in the Azure portal. At the top, there's a breadcrumb navigation: Home > contosostorage | Access Control (IAM) >. Below it, the title 'Add role assignment' is followed by a '... More options' button. A 'Got feedback?' link is also present. The main area has tabs: Role, Members, Conditions (optional), and Review + assign (which is underlined).

The 'Role' section shows 'Storage Blob Data Reader'.
The 'Scope' section shows '/subscriptions/<redacted>/resourcegroups/speech-rg/providers/Microsoft.Storage/storageAccounts/contosostorage'.
The 'Members' section contains a table:

	Name	Object ID	Type
	contoso-speech	<redacted>	Speech service ⓘ

The 'Description' section has 'No description'.
The 'Condition' section has 'None'.

At the bottom, there are two buttons: 'Review + assign' (in blue) and 'Previous'.

8. After confirming the settings, select **Review + assign**

Now the Speech resource managed identity has access to the Storage account and can access the audio files for batch transcription.

With system assigned managed identity, you'll use a plain Storage Account URL (no SAS or other additions) when you [create a batch transcription](#) request. For example:

```
JSON

{
    "contentContainerUrl": "https://<storage_account_name>.blob.core.windows.net/<container_name>"
}
```

You could otherwise specify individual files in the container. For example:

```
JSON

{
    "contentUrls": [
        "https://<storage_account_name>.blob.core.windows.net/<container_name>/<file_name_1>",
        "https://<storage_account_name>.blob.core.windows.net/<container_name>/<file_name_2>"
    ]
}
```



SAS URL for batch transcription

A shared access signature (SAS) is a URI that grants restricted access to an Azure Storage container. Use it when you want to grant access to your batch transcription files for a specific time range without sharing your storage account key.

Tip

If the container with batch transcription source files should only be accessed by your Speech resource, use the **trusted Azure services security mechanism** instead.

Azure portal

Follow these steps to generate a SAS URL that you can use for batch transcriptions.

1. Complete the steps in [Azure Blob Storage upload](#) to create a Storage account and upload audio files to a new container.
2. Select the new container.
3. In the **Settings** group in the left pane, select **Shared access tokens**.
4. Select **+ Container**.
5. Select **Read** and **List** for Permissions.
6. Enter the start and expiry times for the SAS URI, or leave the defaults.

The screenshot shows the Azure Storage container settings page under the 'Shared access tokens' tab. It includes fields for 'Signing method' (set to 'Account key'), 'Signing key' (set to 'Key 1'), 'Stored access policy' (set to 'None'), and 'Permissions' (set to '2 selected' with 'Read' and 'List' checked). Below these are two time range fields: 'Start' (3:06:36 PM, US & Canada) and 'End' (11:06:36 PM, US & Canada). At the bottom, there's a note about allowed IP addresses and a choice between 'HTTPS only' and 'HTTPS and HTTP', with 'HTTPS and HTTP' selected. A 'Generate SAS token and URL' button is at the bottom right.

7. Select Generate SAS token and URL.

You will use the SAS URL when you [create a batch transcription](#) request. For example:

JSON

```
{  
    "contentContainerUrl":  
        "https://<storage_account_name>.blob.core.windows.net/<container_name>?  
        SAS_TOKEN"  
}
```

You could otherwise specify individual files in the container. You must generate and use a different SAS URL with read (r) permissions for each file. For example:

JSON

```
{  
    "contentUrls": [  
  
        "https://<storage_account_name>.blob.core.windows.net/<container_name>/<file  
        _name_1>?SAS_TOKEN_1",  
  
        "https://<storage_account_name>.blob.core.windows.net/<container_name>/<file  
        _name_2>?SAS_TOKEN_2"  
    ]  
}
```

Next steps

- [Batch transcription overview](#)
- [Create a batch transcription](#)
- [Get batch transcription results](#)

Create a batch transcription

Article • 12/08/2022 • 7 minutes to read

With batch transcriptions, you submit the [audio data](#), and then retrieve transcription results asynchronously. The service transcribes the audio data and stores the results in a storage container. You can then [retrieve the results](#) from the storage container.

Create a transcription job

To create a transcription, use the `spx batch transcription create` command. Construct the request parameters according to the following instructions:

- Set the required `content` parameter. You can specify either a semi-colon delimited list of individual files, or the URL for an entire container. For more information about Azure blob storage for batch transcription, see [Locate audio files for batch transcription](#).
- Set the required `language` property. This should match the expected locale of the audio data to transcribe. The locale can't be changed later. The Speech CLI `language` parameter corresponds to the `locale` property in the JSON request and response.
- Set the required `name` property. Choose a transcription name that you can refer to later. The transcription name doesn't have to be unique and can be changed later. The Speech CLI `name` parameter corresponds to the `displayName` property in the JSON request and response.

Here's an example Speech CLI command that creates a transcription job:

Azure CLI

```
spx batch transcription create --api-version v3.1 --name "My Transcription"  
--language "en-US" --content  
https://crbn.us/hello.wav;https://crbn.us/whatstheweatherlike.wav
```

You should receive a response body in the following format:

JSON

```
{  
  "self":  
    "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions/  
    /7f4232d5-9873-47a7-a6f7-4a3f00d00dc0",  
  "model": {
```

```
    "self":  
      "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/models/base/13  
      fb305e-09ad-4bce-b3a1-938c9124dda3"  
    },  
    "links": {  
      "files":  
        "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions  
        /7f4232d5-9873-47a7-a6f7-4a3f00d00dc0/files"  
    },  
    "properties": {  
      "diarizationEnabled": false,  
      "wordLevelTimestampsEnabled": false,  
      "channels": [  
        0,  
        1  
      ],  
      "punctuationMode": "DictatedAndAutomatic",  
      "profanityFilterMode": "Masked"  
    },  
    "lastActionDateTime": "2022-10-21T14:21:59Z",  
    "status": "NotStarted",  
    "createdDateTime": "2022-10-21T14:21:59Z",  
    "locale": "en-US",  
    "displayName": "My Transcription",  
    "description": ""  
  }  
}
```

The top-level `self` property in the response body is the transcription's URI. Use this URI to get details such as the URI of the transcriptions and transcription report files. You also use this URI to update or delete a transcription.

For Speech CLI help with transcriptions, run the following command:

Azure CLI

```
spx help batch transcription
```

Request configuration options

For Speech CLI help with transcription configuration options, run the following command:

Azure CLI

```
spx help batch transcription create advanced
```

Using custom models

Batch transcription uses the default base model for the locale that you specify. You don't need to set any properties to use the default base model.

Optionally, you can modify the previous [create transcription example](#) by setting the `model` property to use a specific base model or [Custom Speech](#) model.

Azure CLI

```
spx batch transcription create --api-version v3.1 --name "My Transcription"
--language "en-US" --content
https://crbn.us/hello.wav;https://crbn.us/whatstheweatherlike.wav --model
"https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/models/base/1a
ae1070-7972-47e9-a977-87e3b05c457d"
```

To use a Custom Speech model for batch transcription, you need the model's URI. You can retrieve the model location when you create or get a model. The top-level `self` property in the response body is the model's URI. For an example, see the JSON response example in the [Create a model](#) guide. A [custom model deployment endpoint](#) isn't needed for the batch transcription service.

Batch transcription requests for expired models will fail with a 4xx error. You'll want to set the `model` property to a base model or custom model that hasn't yet expired.

Otherwise don't include the `model` property to always use the latest base model. For more information, see [Choose a model](#) and [Custom Speech model lifecycle](#).

Destination container URL

The transcription result can be stored in an Azure container. If you don't specify a container, the Speech service stores the results in a container managed by Microsoft. In that case, when the transcription job is deleted, the transcription result data is also deleted.

You can store the results of a batch transcription to a writable Azure Blob storage container using option `destinationContainerUrl` in the [batch transcription creation request](#). Note however that this option is only using [ad hoc SAS](#) URI and doesn't support [Trusted Azure services security mechanism](#). The Storage account resource of the destination container must allow all external traffic.

The [Trusted Azure services security mechanism](#) is not supported for storing transcription results from a Speech resource. If you would like to store the transcription results in an Azure Blob storage container via the [Trusted Azure services security mechanism](#), then

you should consider using [Bring-your-own-storage \(BYOS\)](#). You can secure access to BYOS-associated Storage account exactly as described in the [Trusted Azure services security mechanism](#) guide, except that the BYOS Speech resource would need **Storage Blob Data Contributor** role assignment. The results of batch transcription performed by the BYOS Speech resource will be automatically stored in the **TranscriptionData** folder of the **customspeech-artifacts** blob container.

Next steps

- [Batch transcription overview](#)
 - [Locate audio files for batch transcription](#)
 - [Get batch transcription results](#)
-

Additional resources

Documentation

[Get batch transcription results - Speech service - Azure Cognitive Services](#)

With batch transcription, the Speech service transcribes the audio data and stores the results in a storage container. You can then retrieve the results from the storage container.

[Locate audio files for batch transcription - Speech service - Azure Cognitive Services](#)

Batch transcription is used to transcribe a large amount of audio in storage. You should provide multiple files per request or point to an Azure Blob Storage container with the audio files to transcribe.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Human-labeled transcriptions guidelines - Speech service - Azure Cognitive Services](#)

You use human-labeled transcriptions with your audio data to improve speech recognition accuracy. This is especially helpful when words are deleted or incorrectly replaced.

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Migrate from v3.0 to v3.1 REST API - Speech service - Azure Cognitive Services](#)

This document helps developers migrate code from v3.0 to v3.1 of the Speech to text REST API.

Get batch transcription results

Article • 11/30/2022 • 6 minutes to read

To get transcription results, first check the [status](#) of the transcription job. If the job is completed, you can [retrieve](#) the transcriptions and transcription report.

Get transcription status

To get the status of the transcription job, use the `spx batch transcription status` command. Construct the request parameters according to the following instructions:

- Set the `transcription` parameter to the ID of the transcription that you want to get.

Here's an example Speech CLI command to get the transcription status:

Azure CLI

```
spx batch transcription status --api-version v3.1 --transcription  
YourTranscriptionId
```

You should receive a response body in the following format:

JSON

```
{  
  "self":  
    "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions/  
    /637d9333-6559-47a6-b8de-c7d732c1ddf3",  
  "model": {  
    "self":  
      "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/models/base/aa  
      a321e9-5a4e-4db1-88a2-f251bbe7b555"  
  },  
  "links": {  
    "files":  
      "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions/  
      /637d9333-6559-47a6-b8de-c7d732c1ddf3/files"  
  },  
  "properties": {  
    "diarizationEnabled": false,  
    "wordLevelTimestampsEnabled": true,  
    "displayFormWordLevelTimestampsEnabled": false,  
    "channels": [  
      0,  
      1  
    ],
```

```
        "punctuationMode": "DictatedAndAutomatic",
        "profanityFilterMode": "Masked",
        "duration": "PT3S"
    },
    "lastActionDateTime": "2022-09-10T18:39:09Z",
    "status": "Succeeded",
    "createdDateTime": "2022-09-10T18:39:07Z",
    "locale": "en-US",
    "displayName": "My Transcription"
}
```

The `status` property indicates the current status of the transcriptions. The transcriptions and transcription report will be available when the transcription status is `Succeeded`.

For Speech CLI help with transcriptions, run the following command:

Azure CLI

```
spx help batch transcription
```

Get transcription results

The `spx batch transcription list` command returns a list of result files for a transcription. A [transcription report](#) file is provided for each submitted batch transcription job. In addition, one [transcription](#) file (the end result) is provided for each successfully transcribed audio file.

- Set the required `files` flag.
- Set the required `transcription` parameter to the ID of the transcription that you want to get logs.

Here's an example Speech CLI command that gets a list of result files for a transcription:

Azure CLI

```
spx batch transcription list --api-version v3.1 --files --transcription
YourTranscriptionId
```

You should receive a response body in the following format:

JSON

```
{
  "values": [
    {
      "self": "
```

```
"https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions
/637d9333-6559-47a6-b8de-c7d732c1ddf3/files/2dd180a1-434e-4368-a1ac-
37350700284f",
  "name": "contenturl_0.json",
  "kind": "Transcription",
  "properties": {
    "size": 3407
  },
  "createdDateTime": "2022-09-10T18:39:09Z",
  "links": {
    "contentUrl": "https://spsvcprodeus.blob.core.windows.net/bestor-
c6e3ae79-1b48-41bf-92ff-940bea3e5c2d/TranscriptionData/637d9333-6559-47a6-
b8de-c7d732c1ddf3_0_0.json?sv=2021-08-06&st=2022-09-
10T18%3A36%3A01Z&se=2022-09-
11T06%3A41%3A01Z&sr=b&sp=rl&sig=Aobsq09DH9CIOuGC5iffH3QpkQay6PjHiWn5G87FcIg%
3D"
  }
},
{
  "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions
/637d9333-6559-47a6-b8de-c7d732c1ddf3/files/c027c6a9-2436-4303-b64b-
e98e3c9fc2e3",
  "name": "contenturl_1.json",
  "kind": "Transcription",
  "properties": {
    "size": 8233
  },
  "createdDateTime": "2022-09-10T18:39:09Z",
  "links": {
    "contentUrl": "https://spsvcprodeus.blob.core.windows.net/bestor-
c6e3ae79-1b48-41bf-92ff-940bea3e5c2d/TranscriptionData/637d9333-6559-47a6-
b8de-c7d732c1ddf3_1_0.json?sv=2021-08-06&st=2022-09-
10T18%3A36%3A01Z&se=2022-09-
11T06%3A41%3A01Z&sr=b&sp=rl&sig=w03VxbhLK4PhT3rwLpJXBYHYQi5EQqyl%2Fp1lgjNvfh
0%3D"
  }
},
{
  "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions
/637d9333-6559-47a6-b8de-c7d732c1ddf3/files/faea9a41-c95c-4d91-96ff-
e39225def642",
  "name": "report.json",
  "kind": "TranscriptionReport",
  "properties": {
    "size": 279
  },
  "createdDateTime": "2022-09-10T18:39:09Z",
  "links": {
    "contentUrl": "https://spsvcprodeus.blob.core.windows.net/bestor-
c6e3ae79-1b48-41bf-92ff-940bea3e5c2d/TranscriptionData/637d9333-6559-47a6-
b8de-c7d732c1ddf3_report.json?sv=2021-08-06&st=2022-09-
10T18%3A36%3A01Z&se=2022-09-
11T06%3A41%3A01Z&sr=b&sp=rl&sig=gk1k%2Ft5qa1TpmM45tPommx%2F2%2Bc%2FUUfsYTX5F
```

```
oSa1u%2FY%3D"
    }
}
]
}
```

The location of each transcription and transcription report files with more details are returned in the response body. The `contentUrl` property contains the URL to the transcription ("kind": "Transcription") or transcription report ("kind": "TranscriptionReport") file.

By default, the results are stored in a container managed by Microsoft. When the transcription job is deleted, the transcription result data is also deleted.

Transcription report file

One transcription report file is provided for each submitted batch transcription job.

The contents of each transcription result file are formatted as JSON, as shown in this example.

JSON

```
{
  "successfulTranscriptionsCount": 2,
  "failedTranscriptionsCount": 0,
  "details": [
    {
      "source": "https://crbn.us/hello.wav",
      "status": "Succeeded"
    },
    {
      "source": "https://crbn.us/whatstheweatherlike.wav",
      "status": "Succeeded"
    }
  ]
}
```

Transcription result file

One transcription result file is provided for each successfully transcribed audio file.

The contents of each transcription result file are formatted as JSON, as shown in this example.

JSON

```
{  
  "source": "...",  
  "timestamp": "2022-09-16T09:30:21Z",  
  "durationInTicks": 41200000,  
  "duration": "PT4.12S",  
  "combinedRecognizedPhrases": [  
    {  
      "channel": 0,  
      "lexical": "hello world",  
      "itn": "hello world",  
      "maskedITN": "hello world",  
      "display": "Hello world."  
    }  
  ],  
  "recognizedPhrases": [  
    {  
      "recognitionStatus": "Success",  
      "speaker": 1,  
      "channel": 0,  
      "offset": "PT0.07S",  
      "duration": "PT1.59S",  
      "offsetInTicks": 700000.0,  
      "durationInTicks": 15900000.0,  
  
      "nBest": [  
        {  
          "confidence": 0.898652852,  
          "lexical": "hello world",  
          "itn": "hello world",  
          "maskedITN": "hello world",  
          "display": "Hello world.",  
  
          "words": [  
            {  
              "word": "hello",  
              "offset": "PT0.09S",  
              "duration": "PT0.48S",  
              "offsetInTicks": 900000.0,  
              "durationInTicks": 4800000.0,  
              "confidence": 0.987572  
            },  
            {  
              "word": "world",  
              "offset": "PT0.59S",  
              "duration": "PT0.16S",  
              "offsetInTicks": 5900000.0,  
              "durationInTicks": 1600000.0,  
              "confidence": 0.906032  
            }  
          ]  
        }  
      ]  
    }  
  ]  
}
```

```
]  
}
```

Depending in part on the request parameters set when you created the transcription job, the transcription file can contain the following result properties.

Property	Description
<code>channel</code>	The channel number of the results. For stereo audio streams, the left and right channels are split during the transcription. A JSON result file is created for each input audio file.
<code>combinedRecognizedPhrases</code>	The concatenated results of all phrases for the channel.
<code>confidence</code>	The confidence value for the recognition.
<code>display</code>	The display form of the recognized text. Added punctuation and capitalization are included.
<code>displayPhraseElements</code>	A list of results with display text for each word of the phrase. The <code>displayFormWordLevelTimestampsEnabled</code> request property must be set to <code>true</code> , otherwise this property is not present. Note: This property is only available with speech-to-text REST API version 3.1.
<code>duration</code>	The audio duration. The value is an ISO 8601 encoded duration.
<code>durationInTicks</code>	The audio duration in ticks (1 tick is 100 nanoseconds).
<code>itn</code>	The inverse text normalized (ITN) form of the recognized text. Abbreviations such as "Doctor Smith" to "Dr Smith", phone numbers, and other transformations are applied.
<code>lexical</code>	The actual words recognized.
<code>locale</code>	The locale identified from the input the audio. The <code>languageIdentification</code> request property must be set to <code>true</code> , otherwise this property is not present. Note: This property is only available with speech-to-text REST API version 3.1.
<code>maskedITN</code>	The ITN form with profanity masking applied.
<code>nBest</code>	A list of possible transcriptions for the current phrase with confidences.
<code>offset</code>	The offset in audio of this phrase. The value is an ISO 8601 encoded duration.

Property	Description
<code>offsetInTicks</code>	The offset in audio of this phrase in ticks (1 tick is 100 nanoseconds).
<code>recognitionStatus</code>	The recognition state. For example: "Success" or "Failure".
<code>recognizedPhrases</code>	The list of results for each phrase.
<code>source</code>	The URL that was provided as the input audio source. The source corresponds to the <code>contentUrls</code> or <code>contentContainerUrl</code> request property. The <code>source</code> property is the only way to confirm the audio input for a transcription.
<code>speaker</code>	The identified speaker. The <code>diarization</code> and <code>diarizationEnabled</code> request properties must be set, otherwise this property is not present.
<code>timestamp</code>	The creation date and time of the transcription. The value is an ISO 8601 encoded timestamp.
<code>words</code>	A list of results with lexical text for each word of the phrase. The <code>wordLevelTimestampsEnabled</code> request property must be set to <code>true</code> , otherwise this property is not present.

Next steps

- [Batch transcription overview](#)
- [Locate audio files for batch transcription](#)
- [Create a batch transcription](#)

What is Custom Speech?

Article • 01/17/2023 • 2 minutes to read

With Custom Speech, you can evaluate and improve the Microsoft speech-to-text accuracy for your applications and products.

Out of the box, speech to text utilizes a Universal Language Model as a base model that is trained with Microsoft-owned data and reflects commonly used spoken language. The base model is pre-trained with dialects and phonetics representing a variety of common domains. When you make a speech recognition request, the most recent base model for each [supported language](#) is used by default. The base model works very well in most speech recognition scenarios.

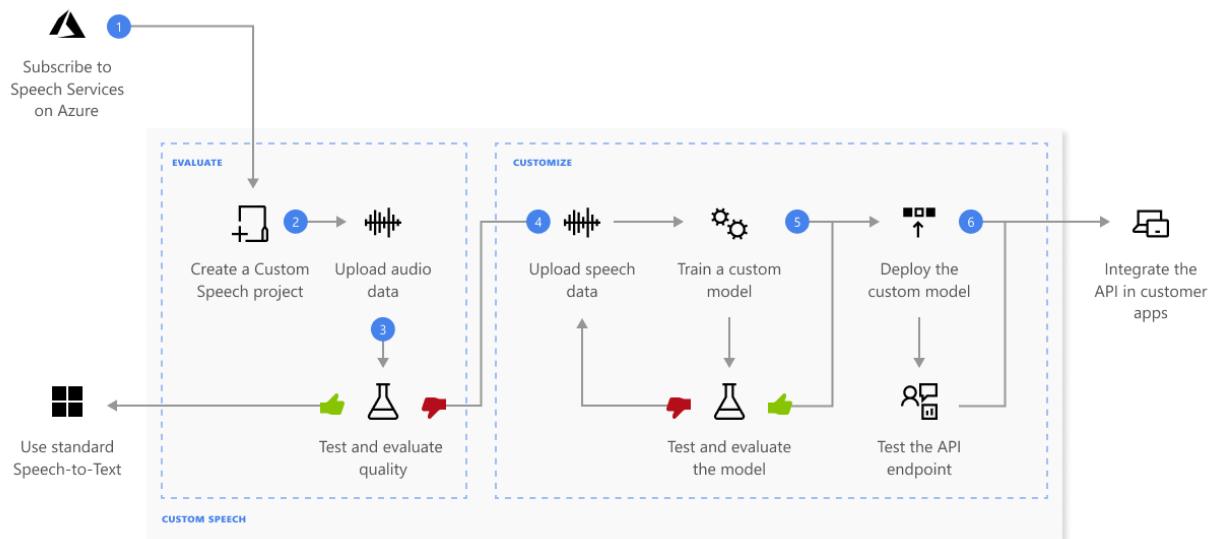
A custom model can be used to augment the base model to improve recognition of domain-specific vocabulary specific to the application by providing text data to train the model. It can also be used to improve recognition based for the specific audio conditions of the application by providing audio data with reference transcriptions.

Note

You pay to use Custom Speech models, but you are not charged for training a model. Usage includes hosting of your deployed custom endpoint in addition to using the endpoint for speech-to-text. For more information, see [Speech service pricing ↗](#).

How does it work?

With Custom Speech, you can upload your own data, test and train a custom model, compare accuracy between models, and deploy a model to a custom endpoint.



Here's more information about the sequence of steps shown in the previous diagram:

1. **Create a project** and choose a model. Use a [Speech resource](#) that you create in the Azure portal. If you will train a custom model with audio data, choose a Speech resource region with dedicated hardware for training audio data. See footnotes in the [regions](#) table for more information.
2. **Upload test data**. Upload test data to evaluate the Microsoft speech-to-text offering for your applications, tools, and products.
3. **Test recognition quality**. Use the [Speech Studio](#) to play back uploaded audio and inspect the speech recognition quality of your test data.
4. **Test model quantitatively**. Evaluate and improve the accuracy of the speech-to-text model. The Speech service provides a quantitative word error rate (WER), which you can use to determine if additional training is required.
5. **Train a model**. Provide written transcripts and related text, along with the corresponding audio data. Testing a model before and after training is optional but recommended.
6. **Deploy a model**. Once you're satisfied with the test results, deploy the model to a custom endpoint. With the exception of [batch transcription](#), you must deploy a custom endpoint to use a Custom Speech model.

Next steps

- [Create a project](#)
- [Upload test data](#)
- [Train a model](#)

Additional resources

Documentation

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Speech-to-text FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions about the speech-to-text service.

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Model lifecycle of Custom Speech - Speech service - Azure Cognitive Services](#)

Custom Speech provides base models for training and lets you create custom models from your data. This article describes the timelines for models and for endpoints that use these models.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Train a Custom Speech model - Speech service - Azure Cognitive Services](#)

Learn how to train Custom Speech models. Training a speech-to-text model can improve recognition accuracy for the Microsoft base model or a custom model.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[`azure.cognitiveservices.speech.SpeechRecognizer` class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, `language`, `source_language_config` or `auto_detect_source_language_config`.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

Create a Custom Speech project

Article • 01/11/2023 • 4 minutes to read

Custom Speech projects contain models, training and testing datasets, and deployment endpoints. Each project is specific to a [locale](#). For example, you might create a project for English in the United States.

Create a project

To create a Custom Speech project, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select the subscription and Speech resource to work with.

Important

If you will train a custom model with audio data, choose a Speech resource region with dedicated hardware for training audio data. See footnotes in the [regions](#) table for more information.

3. Select **Custom speech > Create a new project**.
4. Follow the instructions provided by the wizard to create your project.

Select the new project by name or select **Go to project**. You will see these menu items in the left panel: **Speech datasets**, **Train custom models**, **Test models**, and **Deploy models**.

Choose your model

There are a few approaches to using Custom Speech models:

- The base model provides accurate speech recognition out of the box for a range of [scenarios](#). Base models are updated periodically to improve accuracy and quality. We recommend that if you use base models, use the latest default base models. If a required customization capability is only available with an older model, then you can choose an older base model.
- A custom model augments the base model to include domain-specific vocabulary shared across all areas of the custom domain.
- Multiple custom models can be used when the custom domain has multiple areas, each with a specific vocabulary.

One recommended way to see if the base model will suffice is to analyze the transcription produced from the base model and compare it with a human-generated transcript for the same audio. You can compare the transcripts and obtain a [word error rate \(WER\)](#) score. If the WER score is high, training a custom model to recognize the incorrectly identified words is recommended.

Multiple models are recommended if the vocabulary varies across the domain areas. For instance, Olympic commentators report on various events, each associated with its own vernacular. Because each Olympic event vocabulary differs significantly from others, building a custom model specific to an event increases accuracy by limiting the utterance data relative to that particular event. As a result, the model doesn't need to sift through unrelated data to make a match. Regardless, training still requires a decent variety of training data. Include audio from various commentators who have different accents, gender, age, etcetera.

Model stability and lifecycle

A base model or custom model deployed to an endpoint using Custom Speech is fixed until you decide to update it. The speech recognition accuracy and quality will remain consistent, even when a new base model is released. This allows you to lock in the behavior of a specific model until you decide to use a newer model.

Whether you train your own model or use a snapshot of a base model, you can use the model for a limited time. For more information, see [Model and endpoint lifecycle](#).

Next steps

- [Training and testing datasets](#)
- [Test model quantitatively](#)
- [Train a model](#)

Upload training and testing datasets for Custom Speech

Article • 11/30/2022 • 6 minutes to read

You need audio or text data for testing the accuracy of Microsoft speech recognition or training your custom models. For information about the data types supported for testing or training your model, see [Training and testing datasets](#).

💡 Tip

You can also use the [online transcription editor](#) to create and refine labeled audio datasets.

Upload datasets

To upload your own datasets in Speech Studio, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Speech datasets** > **Upload data**.
3. Select the **Training data** or **Testing data** tab.
4. Select a dataset type, and then select **Next**.
5. Specify the dataset location, and then select **Next**. You can choose a local file or enter a remote location such as Azure Blob URL.

❗ Note

If you use Azure Blob URL, you can ensure maximum security of your dataset files by using trusted Azure services security mechanism. You will use the same techniques as for Batch transcription and plain Storage Account URLs for your dataset files. See details [here](#).

6. Enter the dataset name and description, and then select **Next**.
7. Review your settings, and then select **Save and close**.

After your dataset is uploaded, go to the [Train custom models](#) page to [train a custom model](#).

 **Important**

Connecting a dataset to a Custom Speech project isn't required to train and test a custom model using the REST API or Speech CLI. But if the dataset is not connected to any project, you can't select it for training or testing in the [Speech Studio](#).

Next steps

- [Test recognition quality](#)
- [Test model quantitatively](#)
- [Train a custom model](#)

Test recognition quality of a Custom Speech model

Article • 11/30/2022 • 8 minutes to read

You can inspect the recognition quality of a Custom Speech model in the [Speech Studio](#). You can play back uploaded audio and determine if the provided recognition result is correct. After a test has been successfully created, you can see how a model transcribed the audio dataset, or compare results from two models side by side.

Side-by-side model testing is useful to validate which speech recognition model is best for an application. For an objective measure of accuracy, which requires transcription datasets input, see [Test model quantitatively](#).

Important

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 for [the latest details](#).

Create a test

Follow these instructions to create a test:

1. Sign in to the [Speech Studio](#).
2. Navigate to **Speech Studio > Custom Speech** and select your project name from the list.
3. Select **Test models > Create new test**.
4. Select **Inspect quality (Audio-only data) > Next**.
5. Choose an audio dataset that you'd like to use for testing, and then select **Next**. If there aren't any datasets available, cancel the setup, and then go to the **Speech datasets** menu to [upload datasets](#).

Create new test

Choose testing type

Choose testing data

Choose models to evaluate

Name and description

Review and evaluate

Choose which testing datasets to use

Choose one audio dataset that you'd like to use for testing. Or upload additional from the Speech dataset page.

Name ↓	Type	Quantity
<input checked="" type="checkbox"/> audio	Audio	--

6. Choose one or two models to evaluate and compare accuracy.

7. Enter the test name and description, and then select **Next**.

8. Review your settings, and then select **Save and close**.

Get test results

You should get the test results and [inspect](#) the audio datasets compared to transcription results for each model.

Follow these steps to get test results:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Test models**.
3. Select the link by test name.
4. After the test is complete, as indicated by the status set to *Succeeded*, you should see results that include the WER number for each tested model.

This page lists all the utterances in your dataset and the recognition results, alongside the transcription from the submitted dataset. You can toggle various error types, including insertion, deletion, and substitution. By listening to the audio and comparing recognition results in each column, you can decide which model meets your needs and determine where additional training and improvements are required.

Compare transcription with audio

You can inspect the transcription output by each model tested, against the audio input dataset. If you included two models in the test, you can compare their transcription quality side by side.

To review the quality of transcriptions:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Test models**.
3. Select the link by test name.
4. Play an audio file while the reading the corresponding transcription by a model.

If the test dataset included multiple audio files, you'll see multiple rows in the table. If you included two models in the test, transcriptions are shown in side-by-side columns. Transcription differences between models are shown in blue text font.

Name ↑	Model 1	Model 2
audio1.wav	<p>▶ content like data models tests and improvements are organized into projects in the custom speech portal each project is specific to a domain in country slash language for example you may create a project for call centers that use english in the united states that to create your first project select the speech to text slash 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 for tabs data testing training and deployment use the links provided in next steps to learn how to use each tab</p>	<p>▶ content like data models tests and endpoints are organized into projects in the custom speech portal each project is specific to a domain in country slash language for example you may create a project for call centers that use english in the united states dot to create your first project select the speech to text slash 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</p>
audio2.wav	<p>▶ custom speech provides tools that allow you to visually inspect the recognition quality of model by comparing audio data with the corresponding recognition results from the custom speech portal you can playback uploaded audio and determine if they provided recognition result is correct this tool allows you to quickly inspect quality of microsoft's baseline speech to text models or a trained custom model without having to transcribe any audio data</p>	<p>▶ 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 playback 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</p>

Next steps

- [Test model quantitatively](#)
- [Train your model](#)
- [Deploy your model](#)

Test accuracy of a Custom Speech model

Article • 11/30/2022 • 11 minutes to read

In this article, you learn how to quantitatively measure and improve the accuracy of the Microsoft speech-to-text model or your own custom models. [Audio + human-labeled transcript](#) data is required to test accuracy. You should provide from 30 minutes to 5 hours of representative audio.

Important

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 for [the latest details](#).

Create a test

You can test the accuracy of your custom model by creating a test. A test requires a collection of audio files and their corresponding transcriptions. You can compare a custom model's accuracy a Microsoft speech-to-text base model or another custom model. After you [get](#) the test results, [evaluate](#) the word error rate (WER) compared to speech recognition results.

Follow these steps to create a test:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Test models**.
3. Select **Create new test**.
4. Select **Evaluate accuracy** > **Next**.
5. Select one audio + human-labeled transcription dataset, and then select **Next**. If there aren't any datasets available, cancel the setup, and then go to the **Speech datasets** menu to [upload datasets](#).

Note

It's important to select an acoustic dataset that's different from the one you used with your model. This approach can provide a more realistic sense of the model's performance.

6. Select up to two models to evaluate, and then select **Next**.

7. Enter the test name and description, and then select **Next**.

8. Review the test details, and then select **Save and close**.

Get test results

You should get the test results and [evaluate](#) the word error rate (WER) compared to speech recognition results.

Follow these steps to get test results:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Test models**.
3. Select the link by test name.
4. After the test is complete, as indicated by the status set to *Succeeded*, you should see results that include the WER number for each tested model.

This page lists all the utterances in your dataset and the recognition results, alongside the transcription from the submitted dataset. You can toggle various error types, including insertion, deletion, and substitution. By listening to the audio and comparing recognition results in each column, you can decide which model meets your needs and determine where additional training and improvements are required.

Evaluate word error rate

The industry standard for measuring model accuracy is [word error rate \(WER\)](#). WER counts the number of incorrect words identified during recognition, and divides the sum by the total number of words provided in the human-labeled transcript (N).

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

In the Speech Studio, the quotient is multiplied by 100 and shown as a percentage. The Speech CLI and REST API results aren't multiplied by 100.

$$WER = \frac{I + D + S}{N} \times 100$$

Here's an example that shows incorrectly identified words, when compared to the human-labeled transcript:

Human-labeled Transcript: How are you today John
Speech Recognition Result: How you a today Jones



The speech recognition result erred as follows:

- Insertion (I): Added the word "a"
- Deletion (D): Deleted the word "are"
- Substitution (S): Substituted the word "Jones" for "John"

The word error rate from the previous example is 60%.

If you want to replicate WER measurements locally, you can use the sclite tool from the [NIST Scoring Toolkit \(SCTK\)](#).

Resolve errors and improve WER

You can use the WER calculation from the machine recognition results to evaluate the quality of the model you're 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, but you might 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 need to collect audio data closer to the source. Insertion errors mean that the audio was recorded in a noisy environment and crosstalk might 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.

Example scenario outcomes

Speech recognition scenarios vary by audio quality and language (vocabulary and speaking style). The following table examines four common scenarios:

Scenario	Audio quality	Vocabulary	Speaking style
Call center	Low, 8 kHz, could be two people on one audio channel, could be compressed	Narrow, unique to domain and products	Conversational, loosely structured
Voice assistant, such as Cortana, or a drive-through window	High, 16 kHz	Entity-heavy (song titles, products, locations)	Clearly stated words and phrases
Dictation (instant message, notes, search)	High, 16 kHz	Varied	Note-taking
Video closed captioning	Varied, including varied microphone use, added music	Varied, from meetings, recited speech, musical lyrics	Read, prepared, or loosely structured

Different scenarios produce different quality outcomes. The following table examines how content from these four scenarios rates in the [WER](#). The table shows which error types are most common in each scenario. The insertion, substitution, and deletion error rates help you determine what kind of data to add to improve the model.

Scenario	Speech recognition quality	Insertion errors	Deletion errors	Substitution errors
Call center	Medium (< 30% WER)	Low, except when other people talk in the background	Can be high. Call centers can be noisy, and overlapping speakers can confuse the model	Medium. Products and people's names can cause these errors
Voice assistant	High (can be < 10% WER)	Low	Low	Medium, due to song titles, product names, or locations
Dictation	High (can be < 10% WER)	Low	Low	High

Scenario	Speech recognition quality	Insertion errors	Deletion errors	Substitution errors
Video closed captioning	Depends on video type (can be < 50% WER)	Low	Can be high because of music, noises, microphone quality	Jargon might cause these errors

Next steps

- [Train a model](#)
- [Deploy a model](#)
- [Use the online transcription editor](#)

Train a Custom Speech model

Article • 01/17/2023 • 10 minutes to read

In this article, you'll learn how to train a custom model to improve recognition accuracy from the Microsoft base model. The speech recognition accuracy and quality of a Custom Speech model will remain consistent, even when a new base model is released.

ⓘ Note

You pay to use Custom Speech models, but you are not charged for training a model. Usage includes hosting of your deployed custom endpoint in addition to using the endpoint for speech-to-text. For more information, see [Speech service pricing ↗](#).

Training a model is typically an iterative process. You will first select a base model that is the starting point for a new model. You train a model with [datasets](#) that can include text and audio, and then you test. If the recognition quality or accuracy doesn't meet your requirements, you can create a new model with additional or modified training data, and then test again.

You can use a custom model for a limited time after it's trained. You must periodically recreate and adapt your custom model from the latest base model to take advantage of the improved accuracy and quality. For more information, see [Model and endpoint lifecycle](#).

ⓘ Important

If you will train a custom model with audio data, choose a Speech resource region with dedicated hardware for training audio data. After a model is trained, you can [copy it to a Speech resource](#) in another region as needed.

In regions with dedicated hardware for Custom Speech training, the Speech service will use up to 20 hours of your audio training data, and can process about 10 hours of data per day. In other regions, the Speech service uses up to 8 hours of your audio data, and can process about 1 hour of data per day. See footnotes in the [regions](#) table for more information.

Create a model

After you've uploaded [training datasets](#), follow these instructions to start training your model:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Train custom models**.
3. Select **Train a new model**.
4. On the **Select a baseline model** page, select a base model, and then select **Next**. If you aren't sure, select the most recent model from the top of the list. The name of the base model corresponds to the date when it was released in YYYYMMDD format. The customization capabilities of the base model are listed in parenthesis after the model name in Speech Studio.

 **Important**

Take note of the **Expiration for adaptation** date. This is the last date that you can use the base model for training. For more information, see [Model and endpoint lifecycle](#).

5. On the **Choose data** page, select one or more datasets that you want to use for training. If there aren't any datasets available, cancel the setup, and then go to the **Speech datasets** menu to [upload datasets](#).
6. Enter a name and description for your custom model, and then select **Next**.
7. Optionally, check the **Add test in the next step** box. If you skip this step, you can run the same tests later. For more information, see [Test recognition quality](#) and [Test model quantitatively](#).
8. Select **Save and close** to kick off the build for your custom model.
9. Return to the **Train custom models** page.

 **Important**

Take note of the **Expiration** date. This is the last date that you can use your custom model for speech recognition. For more information, see [Model and endpoint lifecycle](#).

Copy a model

You can copy a model to another project that uses the same locale. For example, after a model is trained with audio data in a [region](#) with dedicated hardware for training, you can copy it to a Speech resource in another region as needed.

Follow these instructions to copy a model to a project in another region:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Train custom models**.
3. Select **Copy to**.
4. On the **Copy speech model** page, select a target region where you want to copy the model.

Copy speech model



Copy the selected custom speech model to another project. Easily deploy the model to a different region or project.

Target region *

West US 2



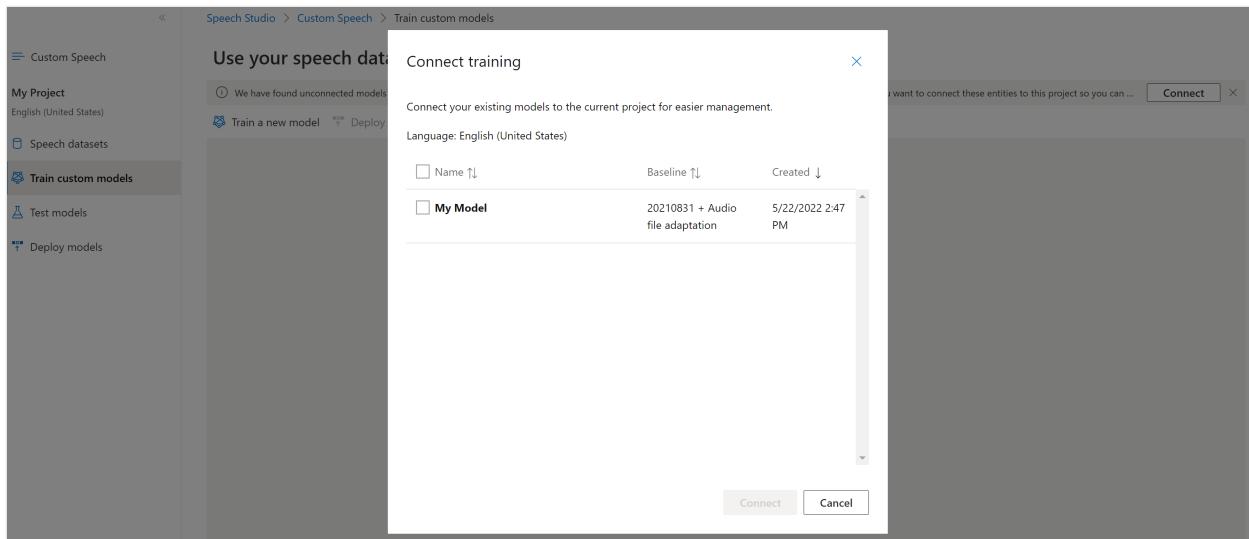
5. Select a Speech resource in the target region, or create a new Speech resource.
6. Select a project where you want to copy the model, or create a new project.
7. Select **Copy**.

After the model is successfully copied, you'll be notified and can view it in the target project.

Connect a model

Models might have been copied from one project using the Speech CLI or REST API, without being connected to another project. Connecting a model is a matter of updating the model with a reference to the project.

If you are prompted in Speech Studio, you can connect them by selecting the **Connect** button.



Next steps

- Test recognition quality
- Test model quantitatively
- Deploy a model

Additional resources

Documentation

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Training and testing datasets - Speech service - Azure Cognitive Services](#)

Learn about types of training and testing data for a Custom Speech project, along with how to use and manage that data.

[Human-labeled transcriptions guidelines - Speech service - Azure Cognitive Services](#)

You use human-labeled transcriptions with your audio data to improve speech recognition accuracy. This is especially helpful when words are deleted or incorrectly replaced.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Speech-to-text FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions about the speech-to-text service.

[Speaker Recognition quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you use speaker recognition to confirm who is speaking. Learn about common design patterns for working with speaker verification and identification.

[Custom Speech overview - Speech service - Azure Cognitive Services](#)

Custom Speech is a set of online tools that allows you to evaluate and improve the Microsoft speech-to-text accuracy for your applications, tools, and products.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

Deploy a Custom Speech model

Article • 01/17/2023 • 8 minutes to read

In this article, you'll learn how to deploy an endpoint for a Custom Speech model. With the exception of [batch transcription](#), you must deploy a custom endpoint to use a Custom Speech model.

ⓘ Note

You pay to use Custom Speech models, but you are not charged for training a model. Usage includes hosting of your deployed custom endpoint in addition to using the endpoint for speech-to-text. For more information, see [Speech service pricing](#).

You can deploy an endpoint for a base or custom model, and then [update](#) the endpoint later to use a better trained model.

ⓘ Note

Endpoints used by  Speech resources are deleted after seven days.

Add a deployment endpoint

To create a custom endpoint, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Deploy models**.

If this is your first endpoint, you'll notice that there are no endpoints listed in the table. After you create an endpoint, you use this page to track each deployed endpoint.

3. Select **Deploy model** to start the new endpoint wizard.
4. On the **New endpoint** page, enter a name and description for your custom endpoint.
5. Select the custom model that you want to associate with the endpoint.

6. Optionally, you can check the box to enable audio and diagnostic [logging](#) of the endpoint's traffic.

New endpoint

[X](#)

Name *

Description

Model *

Expiration for decoding: 7/14/2024 5:00 PM [Learn more about model deprecation policy](#)

* I accept the [terms of use](#) and [the pricing information](#).

Enable content logging (audio and diagnostic information) for this endpoint.

[Add](#) [Cancel](#)

7. Select **Add** to save and deploy the endpoint.

On the main **Deploy models** page, details about the new endpoint are displayed in a table, such as name, description, status, and expiration date. It can take up to 30 minutes to instantiate a new endpoint that uses your custom models. When the status of the deployment changes to **Succeeded**, the endpoint is ready to use.

ⓘ Important

Take note of the model expiration date. This is the last date that you can use your custom model for speech recognition. For more information, see [Model and endpoint lifecycle](#).

Select the endpoint link to view information specific to it, such as the endpoint key, endpoint URL, and sample code.

Change model and redeploy endpoint

An endpoint can be updated to use another model that was created by the same Speech resource. As previously mentioned, you must update the endpoint's model before the [model expires](#).

To use a new model and redeploy the custom endpoint:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Deploy models**.
3. Select the link to an endpoint by name, and then select **Change model**.
4. Select the new model that you want the endpoint to use.
5. Select **Done** to save and redeploy the endpoint.

The redeployment takes several minutes to complete. In the meantime, your endpoint will use the previous model without interruption of service.

View logging data

Logging data is available for export if you configured it while creating the endpoint.

To download the endpoint logs:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Deploy models**.
3. Select the link by endpoint name.
4. Under **Content logging**, select **Download log**.

Logging data is available on Microsoft-owned storage for 30 days, after which it will be removed. If your own storage account is linked to the Cognitive Services subscription, the logging data won't be automatically deleted.

Next steps

- [CI/CD for Custom Speech](#)
- [Custom Speech model lifecycle](#)

Additional resources



[Model lifecycle of Custom Speech - Speech service - Azure Cognitive Services](#)

Custom Speech provides base models for training and lets you create custom models from your

data. This article describes the timelines for models and for endpoints that use these models.

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[Intent recognition overview - Speech service - Azure Cognitive Services](#)

Intent recognition allows you to recognize user objectives you have pre-defined. This article is an overview of the benefits and capabilities of the intent recognition service.

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[azure.cognitiveservices.speech.SpeechRecognizer class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, language, source_language_config or auto_detect_source_language_config.

[Structured text phonetic pronunciation data - Azure Cognitive Services](#)

Use phonemes to customize pronunciation of words in Speech-to-Text.

[Show 5 more](#)

Training and testing datasets

Article • 02/03/2023 • 14 minutes to read

In a Custom Speech project, you can upload datasets for training, qualitative inspection, and quantitative measurement. This article covers the types of training and testing data that you can use for Custom Speech.

Text and audio that you use to test and train a custom model should include samples from a diverse set of speakers and scenarios that you want your model to recognize. Consider these factors when you're gathering data for custom model testing and training:

- Include text and audio data to cover the kinds of verbal statements that your users will make when they're interacting with your model. For example, a model that raises and lowers the temperature needs training on statements that people might make to request such changes.
- Include all speech variances that you want your model to recognize. Many factors can vary speech, including accents, dialects, language-mixing, age, gender, voice pitch, stress level, and time of day.
- Include samples from different environments, for example, indoor, outdoor, and road noise, where your model will be used.
- Record audio with hardware devices that the production system will use. If your model must identify speech recorded on devices of varying quality, the audio data that you provide to train your model must also represent these diverse scenarios.
- Keep the dataset diverse and representative of your project requirements. You can add more data to your model later.
- Only include data that your model needs to transcribe. Including data that isn't within your custom model's recognition requirements can harm recognition quality overall.

Data types

The following 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 for testing	Used for training	Recommended for training

Data type	Used for testing	Recommended for testing	Used for training	Recommended for training
Audio only	Yes (visual inspection)	5+ audio files	Yes (Preview for en-us)	1-20 hours of audio
Audio + human-labeled transcripts	Yes (evaluation of accuracy)	0.5-5 hours of audio	Yes	1-20 hours of audio
Plain text	No	Not applicable	Yes	1-200 MB of related text
Structured text	No	Not applicable	Yes	Up to 10 classes with up to 4,000 items and up to 50,000 training sentences
Pronunciation	No	Not applicable	Yes	1 KB to 1 MB of pronunciation text

Training with plain text or structured text usually finishes within a few minutes.

💡 Tip

Start with plain-text data or structured-text data. This data will improve the recognition of special terms and phrases. Training with text is much faster than training with audio (minutes versus days).

Start with small sets of sample data that match the language, acoustics, and hardware where your model will be used. Small datasets of representative data can expose problems before you invest in gathering larger datasets for training. For sample Custom Speech data, see [this GitHub repository](#).

If you will train a custom model with audio data, choose a Speech resource region with dedicated hardware for training audio data. See footnotes in the [regions](#) table for more information. In regions with dedicated hardware for Custom Speech training, the Speech service will use up to 20 hours of your audio training data, and can process about 10 hours of data per day. In other regions, the Speech service uses up to 8 hours of your audio data, and can process about 1 hour of data per day. After the model is trained, you can copy the model to another region as needed with the [Models_CopyTo](#) REST API.

Consider datasets by scenario

A model that's trained on a subset of scenarios can perform well in only those scenarios. Carefully choose data that represents the full scope of scenarios that you need your custom model to recognize. The following table shows datasets to consider for some speech recognition scenarios:

Scenario	Plain text data and structured text data	Audio + human-labeled transcripts	New words with pronunciation
Call center	Marketing documents, website, product reviews related to call center activity	Call center calls transcribed by humans	Terms that have ambiguous pronunciations (see the <i>Xbox</i> example in the preceding section)
Voice assistant	Lists of sentences that use various combinations of commands and entities	Recorded voices speaking commands into device, transcribed into text	Names (movies, songs, products) that have unique pronunciations
Dictation	Written input, such as instant messages or emails	Similar to preceding examples	Similar to preceding examples
Video closed captioning	TV show scripts, movies, marketing content, video summaries	Exact transcripts of videos	Similar to preceding examples

To help determine which dataset to use to address your problems, refer to the following table:

Use case	Data type
Improve recognition accuracy on industry-specific vocabulary and grammar, such as medical terminology or IT jargon.	Plain text or structured text data
Define the phonetic and displayed form of a word or term that has nonstandard pronunciation, such as product names or acronyms.	Pronunciation data or phonetic pronunciation in structured text
Improve recognition accuracy on speaking styles, accents, or specific background noises.	Audio + human-labeled transcripts

Audio + human-labeled transcript data for training or testing

You can use audio + human-labeled transcript data for both [training](#) and [testing](#) purposes. You must provide human-labeled transcriptions (word by word) for

comparison:

- To improve the acoustic aspects like slight accents, speaking styles, and background noises.
- To measure the accuracy of Microsoft's speech-to-text accuracy when it's processing your audio files.

For a list of base models that support training with audio data, see [Language support](#). Even if a base model does support training with audio data, the service might use only part of the audio. And it will still use all the transcripts.

ⓘ Important

If a base model doesn't support customization with audio data, only the transcription text will be used for training. If you switch to a base model that supports customization with audio data, the training time may increase from several hours to several days. The change in training time would be most noticeable when you switch to a base model in a **region** without dedicated hardware for training. If the audio data is not required, you should remove it to decrease the training time.

Audio with human-labeled transcripts offers the greatest accuracy improvements if the audio comes from the target use case. Samples must cover the full scope of speech. For example, a call center for a retail store would get the most calls about swimwear and sunglasses during summer months. Ensure that your sample includes the full scope of speech that you want to detect.

Consider these details:

- Training with audio will bring the most benefits if the audio is also hard to understand for humans. In most cases, you should start training by using only related text.
- If you use one of the most heavily used languages, such as US English, it's unlikely that you would need to train with audio data. For such languages, the base models already offer very good recognition results in most scenarios, so it's probably enough to train with related text.
- Custom Speech can capture word context only to reduce substitution errors, not insertion or deletion errors.
- Avoid samples that include transcription errors, but do include a diversity of audio quality.
- Avoid sentences that are unrelated to your problem domain. Unrelated sentences can harm your model.

- When the transcript quality varies, you can duplicate exceptionally good sentences, such as excellent transcriptions that include key phrases, to increase their weight.
- The Speech service automatically uses the transcripts to improve the recognition of domain-specific words and phrases, as though they were added as related text.
- It can take several days for a training operation to finish. To improve the speed of training, be sure to create your Speech service subscription in a region that has dedicated hardware for training.

A large training dataset is required to improve recognition. Generally, we recommend that you provide word-by-word transcriptions for 1 to 20 hours of audio. However, even as little as 30 minutes can help improve recognition results. Although creating human-labeled transcription can take time, improvements in recognition will only be as good as the data that you provide. You should upload only high-quality transcripts.

Audio files can have silence at the beginning and end of the recording. If possible, include at least a half-second of silence before and after speech in each sample file. Although audio with low recording volume or disruptive background noise is not helpful, it shouldn't limit or degrade your custom model. Always consider upgrading your microphones and signal processing hardware before gathering audio samples.

 **Important**

For more information about the best practices of preparing human-labeled transcripts, see [Human-labeled transcripts with audio](#).

Custom Speech projects require audio files with these properties:

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) Training with audio has a maximum audio length of 60 seconds per file. For audio files longer than 60 seconds, only the corresponding transcription files will be used for training. If all audio files are longer than 60 seconds, the training will fail.
Sample format	PCM, 16-bit

Property	Value
Archive format	.zip
Maximum zip size	2 GB or 10,000 files

Plain-text data for training

You can add plain text sentences of related text to improve the recognition of domain-specific words and phrases. Related text sentences can reduce substitution errors related to misrecognition of common words and domain-specific words by showing them in context. Domain-specific words can be uncommon or made-up words, but their pronunciation must be straightforward to be recognized.

Provide domain-related sentences in a single text file. Use text data that's close to the expected spoken utterances. Utterances don't need to be complete or grammatically correct, but they must accurately reflect the spoken input that you expect the model to recognize. When possible, try to have one sentence or keyword controlled on a separate line. To increase the weight of a term such as product names, add several sentences that include the term. But don't copy too much - it could affect the overall recognition rate.

ⓘ Note

Avoid related text sentences that include noise such as unrecognizable characters or words.

Use this table to ensure that your plain text dataset file is formatted correctly:

Property	Value
Text encoding	UTF-8 BOM
Number of utterances per line	1
Maximum file size	200 MB

You must also adhere to the following restrictions:

- Avoid repeating characters, words, or groups of words more than three times, as in "aaaa," "yeah yeah yeah yeah," or "that's it that's it that's it that's it." The Speech service might drop lines with too many repetitions.

- Don't use special characters or UTF-8 characters above `U+00A1`.
- URLs will be rejected.
- For some languages such as Japanese or Korean, importing large amounts of text data can take a long time or can time out. Consider dividing the dataset into multiple text files with up to 20,000 lines in each.

Structured-text data for training

Note

Structured-text data for training is in public preview.

Use structured text data when your data follows a particular pattern in particular utterances that differ only by words or phrases from a list. To simplify the creation of training data and to enable better modeling inside the Custom Language model, you can use a structured text in Markdown format to define lists of items and phonetic pronunciation of words. You can then reference these lists inside your training utterances.

Expected utterances often follow a certain pattern. One common pattern is that utterances differ only by words or phrases from a list. Examples of this pattern could be:

- "I have a question about `product`," where `product` is a list of possible products.
- "Make that `object` `color`," where `object` is a list of geometric shapes and `color` is a list of colors.

For a list of supported base models and locales for training with structured text, see [Language support](#). You must use the latest base model for these locales. For locales that don't support training with structured text, the service will take any training sentences that don't reference any classes as part of training with plain-text data.

The structured-text file should have an .md extension. The maximum file size is 200 MB, and the text encoding must be UTF-8 BOM. The syntax of the Markdown is the same as that from the Language Understanding models, in particular list entities and example utterances. For more information about the complete Markdown syntax, see the [Language Understanding Markdown](#).

Here are key details about the supported Markdown format:

Property	Description	Limits
----------	-------------	--------

Property	Description	Limits
@list	A list of items that can be referenced in an example sentence.	Maximum of 20 lists. Maximum of 35,000 items per list.
speech:phoneticlexicon	A list of phonetic pronunciations according to the Universal Phone Set . Pronunciation is adjusted for each instance where the word appears in a list or training sentence. For example, if you have a word that sounds like "cat" and you want to adjust the pronunciation to "k ae t", you would add <code>- cat/k ae t</code> to the speech:phoneticlexicon list.	Maximum of 15,000 entries. Maximum of 2 pronunciations per word.
#ExampleSentences	A pound symbol (#) delimits a section of example sentences. The section heading can only contain letters, digits, and underscores. Example sentences should reflect the range of speech that your model should expect. A training sentence can refer to items under a @list by using surrounding left and right curly braces (<code>{@list name}</code>). You can refer to multiple lists in the same training sentence, or none at all.	Maximum file size of 200MB.
//	Comments follow a double slash (//).	Not applicable

Here's an example structured text file:

markdown

```
// This is a comment because it follows a double slash (^//^).

// Here are three separate lists of items that can be referenced in an
example sentence. You can have up to 10 of these.

@ list food =
- pizza
- burger
- ice cream
- soda

@ list pet =
- cat
- dog
- fish

@ list sports =
- soccer
- tennis
- cricket
- basketball
```

```

- baseball
- football

// List of phonetic pronunciations
@ speech:phoneticlexicon
- cat/k ae t
- fish/f ih sh

// Here are two sections of training sentences.
#TrainingSentences_Section1
- you can include sentences without a class reference
- what {@pet} do you have
- I like eating {@food} and playing {@sports}
- my {@pet} likes {@food}

#TrainingSentences_Section2
- you can include more sentences without a class reference
- or more sentences that have a class reference like {@pet}

```

Pronunciation data for training

Specialized or made up words might have unique pronunciations. These words can be recognized if they can be broken down into smaller words to pronounce them. For example, to recognize "Xbox", pronounce it as "X box". This approach won't increase overall accuracy, but can improve recognition of this and other keywords.

You can provide a custom pronunciation file to improve recognition. Don't use custom pronunciation files to alter the pronunciation of common words. For a list of languages that support custom pronunciation, see [language support](#).

Note

You can use a pronunciation file alongside any other training dataset except structured text training data. To use pronunciation data with structured text, it must be within a structured text file.

The spoken form is the phonetic sequence spelled out. It can be composed of letters, words, syllables, or a combination of all three. This table includes some examples:

Recognized displayed form	Spoken form
3CPO	three c p o
CNTK	c n t k
IEEE	i triple e

You provide pronunciations in a single text file. Include the spoken utterance and a custom pronunciation for each. Each row in the file should begin with the recognized form, then a tab character, and then the space-delimited phonetic sequence.

tsv
3CPO three c p o
CNTK c n t k
IEEE i triple e

Refer to the following table to ensure that your pronunciation dataset files are valid and correctly formatted.

Property	Value
Text encoding	UTF-8 BOM (ANSI is also supported for English)
Number of pronunciations per line	1
Maximum file size	1 MB (1 KB for free tier)

Audio data for training or testing

Audio data is optimal for testing the accuracy of Microsoft's baseline speech-to-text model or a custom model. Keep in mind that audio data is used to inspect the accuracy of speech with regard to a specific model's performance. If you want to quantify the accuracy of a model, use [audio + human-labeled transcripts](#).

Note

Audio only data for training is available in preview for the `en-us` locale. For other locales, to train with audio data you must also provide [human-labeled transcripts](#).

Custom Speech projects require audio files with these properties:

Property	Value
File format	RIFF (WAV)
Sample rate	8,000 Hz or 16,000 Hz
Channels	1 (mono)
Maximum length per audio	2 hours

Property	Value
Sample format	PCM, 16-bit
Archive format	.zip
Maximum archive size	2 GB or 10,000 files

ⓘ Note

When you're uploading training and testing data, the .zip file size can't 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 test from only a *single* dataset.

Use [SoX](#) to verify audio properties or convert existing audio to the appropriate formats. Here are some example SoX commands:

Activity	SoX command
Check the audio file format.	<code>sox --i <filename></code>
Convert the audio file to single channel, 16-bit, 16 KHz.	<code>sox <input> -b 16 -e signed-integer -c 1 -r 16k -t wav <output>.wav</code>

Next steps

- [Upload your data](#)
- [Test model quantitatively](#)
- [Train a custom model](#)

Additional resources

☰ Documentation

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[azure.cognitiveservices.speech.SpeechRecognizer class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, language, source_language_config or auto_detect_source_language_config.

[azure-cognitiveservices-speech package](#)

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Test accuracy of a Custom Speech model - Speech service - Azure Cognitive Services](#)

In this article, you learn how to quantitatively measure and improve the quality of our speech-to-text model or your custom model.

[Train a Custom Speech model - Speech service - Azure Cognitive Services](#)

Learn how to train Custom Speech models. Training a speech-to-text model can improve recognition accuracy for the Microsoft base model or a custom model.

[Show 5 more](#)

How to create human-labeled transcriptions

Article • 05/12/2022 • 6 minutes to read

Human-labeled transcriptions are word-by-word transcriptions of an audio file. You use human-labeled transcriptions to improve recognition accuracy, especially when words are deleted or incorrectly replaced. This guide can help you create high-quality transcriptions.

A large sample of transcription data is required to improve recognition. We suggest providing between 1 and 20 hours of audio data. The Speech service will use up to 20 hours of audio for training. This guide is broken up by locale, with sections for US English, Mandarin Chinese, and German.

The transcriptions for all WAV files are contained in a single plain-text file. Each line of the transcription file contains the name of one of the audio files, followed by the corresponding transcription. The file name and transcription are separated by a tab (\t).

For example:

tsv	
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

The transcriptions are text-normalized so the system can process them. However, you must do some important normalizations before you upload the dataset.

Human-labeled transcriptions for languages other than English and Mandarin Chinese, must be UTF-8 encoded with a byte-order marker. For other locales transcription requirements, see the sections below.

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.
- If you use audio, transcribe numbers as words that match the audio (for example, "101" could be pronounced as "one oh one" or "one hundred and one").
- Avoid repeating characters, words, or groups of words more than three times, such as "yeah yeah yeah yeah". Lines with such repetitions might be dropped by the Speech service.

Here are a few examples of normalization that you should perform on the transcription:

Original text	Text after normalization (human)
Dr. Bruce Banner	Doctor Bruce Banner
James Bond, 007	James Bond, double oh seven
Ke\$ha	Kesha

Original text	Text after normalization (human)
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 H20	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
It costs \$3.14	It costs three fourteen

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 (automatic)
"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

de-DE

Human-labeled transcriptions for German audio must be UTF-8 encoded with a byte-order marker.

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: € ø ¥ ¢ § © ª ¬ ® ° ± ² µ × ö Ø ñ ñ.
- 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
¡Eine Frage!	eine frage

Original text	Text after normalization
Wir, haben	wir haben

ja-JP

In Japanese (ja-JP), there's a maximum length of 90 characters for each sentence. Lines with longer sentences will be discarded. To add longer text, insert a period in between.

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

Original text	Text after normalization
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 some examples of automatic transcription normalization:

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

Next Steps

- Test model quantitatively
- Test recognition quality
- Train your model

Structured text phonetic pronunciation data

Article • 09/13/2022 • 2 minutes to read

You can specify the phonetic pronunciation of words using the Universal Phone Set (UPS) in a [structured text data](#) file. The UPS is a machine-readable phone set that is based on the International Phonetic Set Alphabet (IPA). The IPA is a standard used by linguists world-wide.

UPS pronunciations consist of a string of UPS phonemes, each separated by whitespace. UPS phoneme labels are all defined using ASCII character strings.

For steps on implementing UPS, see [Structured text phonetic pronunciation](#). Structured text phonetic pronunciation data is separate from [pronunciation data](#), and they cannot be used together. The first one is "sounds-like" or spoken-form data, and is input as a separate file, and trains the model what the spoken form sounds like

[Structured text phonetic pronunciation data](#) is specified per syllable in a markdown file. Separately, [pronunciation data](#) it input on its own, and trains the model what the spoken form sounds like. You can either use a pronunciation data file on its own, or you can add pronunciation within a structured text data file. The Speech service doesn't support training a model with both of those datasets as input.

See the sections in this article for the Universal Phone Set for each locale.

en-US

Consonants

UPS Phonemes	IPA	Example
B	b	big
CH	tʃ / tɬ	chin
D	d	dig
DH	ð	then
F	f	fork
G	g	gut

UPS Phonemes	IPA	Example
H	h	help
JH	dʒ / dʒ	joy
K	k	cut
L	l	lid
M	m	mat
N	n	no
NG	ŋ	sing
P	p	put
R	ɹ	red
S	s	sit
SH	ʃ	she
T	t	talk
TH	θ	thin
V	v	vat
W	w	with
J	j	yard
Z	z	zap
ZH	ʒ	pleasure

Vowels

UPS Phonemes	IPA	Example
AA	a	father
AE	æ	cat
AH	ʌ	cut
AO	ɔ	dog

UPS Phonemes	IPA	Example
AOX	ɔ.ə	four
AU	a.ʊ	foul
AX	ə	ago
AX R	ə̯	minor
AI	a.i	bite
EH	ɛ	pet
EHX	ɛ.ə	stairs
ER R	ɜ̯	urban
EI	e.i	ate
IH	ɪ	fill
I	i	feel
O	o	go
OI	ɔ.i	toy
OwX	o.ə	boa
Q	ɒ	hot
UH	ʊ	book
U	u	too, blue
UwX	u.ə	lure

Next steps

- [Upload your data](#)
- [Test recognition quality](#)
- [Train your model](#)

How to use the online transcription editor

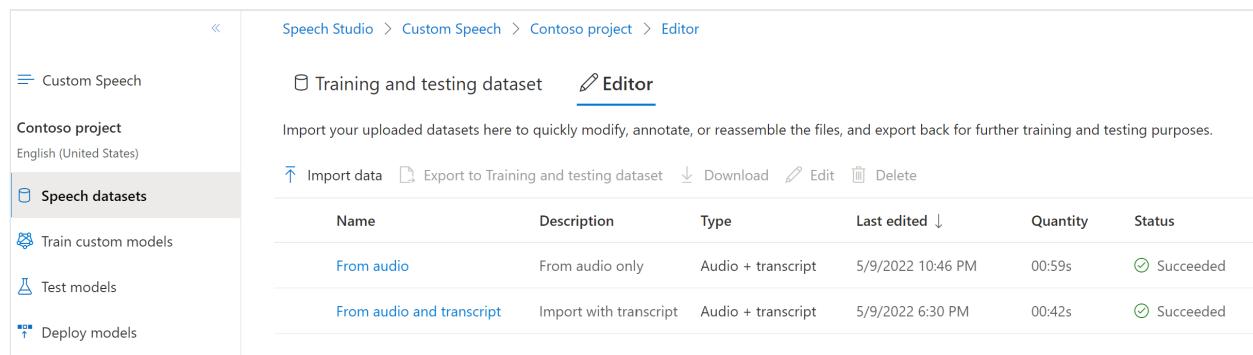
Article • 05/27/2022 • 3 minutes to read

The online transcription editor allows you to create or edit audio + human-labeled transcriptions for Custom Speech. The main use cases of the editor are as follows:

- You only have audio data, but want to build accurate audio + human-labeled datasets from scratch to use in model training.
- You already have audio + human-labeled datasets, but there are errors or defects in the transcription. The editor allows you to quickly modify the transcriptions to get best training accuracy.

The only requirement to use the transcription editor is to have audio data uploaded, with or without corresponding transcriptions.

You can find the **Editor** tab next to the **Training and testing dataset** tab on the main **Speech datasets** page.



The screenshot shows the Microsoft Azure Speech Studio interface. On the left, there's a sidebar with navigation links: 'Custom Speech', 'Contoso project English (United States)', 'Speech datasets' (which is highlighted in blue), 'Train custom models', 'Test models', and 'Deploy models'. The main content area has a breadcrumb trail: 'Speech Studio > Custom Speech > Contoso project > Editor'. Below the breadcrumb, it says 'Import your uploaded datasets here to quickly modify, annotate, or reassemble the files, and export back for further training and testing purposes.' There are buttons for 'Import data', 'Export to Training and testing dataset', 'Download', 'Edit', and 'Delete'. A table below shows two datasets:

Name	Description	Type	Last edited	Quantity	Status
From audio	From audio only	Audio + transcript	5/9/2022 10:46 PM	00:59s	Succeeded
From audio and transcript	Import with transcript	Audio + transcript	5/9/2022 6:30 PM	00:42s	Succeeded

Datasets in the **Training and testing dataset** tab can't be updated. You can import a copy of a training or testing dataset to the **Editor** tab, add or edit human-labeled transcriptions to match the audio, and then export the edited dataset to the **Training and testing dataset** tab. Please also note that you can't use a dataset that's in the Editor to train or test a model.

Import datasets to the Editor

To import a dataset to the Editor, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Speech datasets** > **Editor**.
3. Select **Import data**.

4. Select datasets. You can select audio data only, audio + human-labeled data, or both. For audio-only data, you can use the default models to automatically generate machine transcription after importing to the editor.
5. Enter a name and description for the new dataset, and then select **Next**.
6. Review your settings, and then select **Import and close** to kick off the import process. After data has been successfully imported, you can select datasets and start editing.

 **Note**

You can also select a dataset from the main **Speech datasets** page and export them to the Editor. Select a dataset and then select **Export to Editor**.

Edit transcription to match audio

Once a dataset has been imported to the Editor, you can start editing the dataset. You can add or edit human-labeled transcriptions to match the audio as you hear it. You do not edit any audio data.

To edit a dataset's transcription in the Editor, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Speech datasets** > **Editor**.
3. Select the link to a dataset by name.
4. From the **Audio + text files** table, select the link to an audio file by name.
5. After you've made edits, select **Save**.

If there are multiple files in the dataset, you can select **Previous** and **Next** to move from file to file. Edit and save changes to each file as you go.

The detail page lists all the segments in each audio file, and you can select the desired utterance. For each utterance, you can play and compare the audio with the corresponding transcription. Edit the transcriptions if you find any insertion, deletion, or substitution errors. For more information about word error types, see [Test model quantitatively](#).

Export datasets from the Editor

Datasets in the Editor can be exported to the **Training and testing** dataset tab, where they can be used to train or test a model.

To export datasets from the Editor, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Speech datasets** > **Editor**.
3. Select the link to a dataset by name.
4. Select one or more rows from the **Audio + text files** table.
5. Select **Export** to export all of the selected files as one new dataset.

The files are exported as a new dataset, and will not impact or replace other training or testing datasets.

Next steps

- [Test recognition quality](#)
- [Train your model](#)
- [Deploy your model](#)

Custom Speech model lifecycle

Article • 01/11/2023 • 6 minutes to read

You can use a Custom Speech model for some time after it's deployed to your custom endpoint. But when new base models are made available, the older models are expired. You must periodically recreate and train your custom model from the latest base model to take advantage of the improved accuracy and quality.

Here are some key terms related to the model lifecycle:

- **Training:** Taking a base model and customizing it to your domain/scenario by using text data and/or audio data. In some contexts such as the REST API properties, training is also referred to as **adaptation**.
- **Transcription:** Using a model and performing speech recognition (decoding audio into text).
- **Endpoint:** A specific deployment of either a base model or a custom model that only you can access.

ⓘ Note

Endpoints used by F0 Speech resources are deleted after seven days.

Expiration timeline

Here are timelines for model adaptation and transcription expiration:

- Training is available for one year after the quarter when the base model was created by Microsoft.
- Transcription with a base model is available for two years after the quarter when the base model was created by Microsoft.
- Transcription with a custom model is available for two years after the quarter when you created the custom model.

In this context, quarters end on January 15th, April 15th, July 15th, and October 15th.

What to do when a model expires

When a custom model or base model expires, it is no longer available for transcription. You can change the model that is used by your custom speech endpoint without downtime.

Transcription route	Expired model result	Recommendation
Custom endpoint	Speech recognition requests will fall back to the most recent base model for the same locale . You will get results, but recognition might not accurately transcribe your domain data.	Update the endpoint's model as described in the Deploy a Custom Speech model guide.
Batch transcription	Batch transcription requests for expired models will fail with a 4xx error.	In each Transcriptions_Create REST API request body, set the <code>model</code> property to a base model or custom model that hasn't yet expired. Otherwise don't include the <code>model</code> property to always use the latest base model.

Get base model expiration dates

The last date that you could use the base model for training was shown when you created the custom model. For more information, see [Train a Custom Speech model](#).

Follow these instructions to get the transcription expiration date for a base model:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Deploy models**.
3. The expiration date for the model is shown in the **Expiration** column. This is the last date that you can use the model for transcription.

Name	Description	Model	Logging	Created	Expiration	Status
Endpoint with base model	Base model deployed	20210831 + Audio file adaptation	No	5/22/2022 8:47 AM	1/14/2024 4:00 PM	Succeeded
Endpoint with custom model	Custom model deployed	My Model	No	5/22/2022 8:45 AM	7/14/2024 5:00 PM	Succeeded

Get custom model expiration dates

Follow these instructions to get the transcription expiration date for a custom model:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Train custom models**.

3. The expiration date the custom model is shown in the **Expiration** column. This is the last date that you can use the custom model for transcription. Base models are not shown on the **Train custom models** page.

The screenshot shows the 'Train custom models' section of the Azure Speech Studio. On the left, a sidebar lists 'Custom Speech', 'My Project (English (United States))', 'Speech datasets', 'Train custom models' (which is selected and highlighted in grey), 'Test models', and 'Deploy models'. The main area has a title 'Use your speech data to train a custom speech model' and a toolbar with 'Train a new model', 'Deploy models', 'Test model', 'Edit', 'Delete', and 'Copy to'. A table lists one model: 'My Model' with 'My Model Description', 'Baseline 20210831 + Audio file adaptation', 'Created 5/22/2022 8:37 AM', and 'Expiration 7/14/2024 5:00 PM'.

You can also follow these instructions to get the transcription expiration date for a custom model:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Speech** > Your project name > **Deploy models**.
3. The expiration date for the model is shown in the **Expiration** column. This is the last date that you can use the model for transcription.

The screenshot shows the 'Deploy models' section of the Azure Speech Studio. On the left, a sidebar lists 'Custom Speech', 'My Project (English (United States))', 'Speech datasets', 'Train custom models', 'Test models' (which is selected and highlighted in grey), and 'Deploy models'. The main area has a title 'Define an endpoint to deploy a custom speech model in your solution' and a toolbar with 'Deploy model', 'Change model', 'Edit', and 'Delete'. A table lists two endpoints: 'Endpoint with base model' (Base model deployed, Model 20210831 + Audio file adaptation, Logging No, Created 5/22/2022 8:47 AM, Expiration 1/14/2024 4:00 PM, Status Succeeded) and 'Endpoint with custom model' (Custom model deployed, Model My Model, Logging No, Created 5/22/2022 8:45 AM, Expiration 7/14/2024 5:00 PM, Status Succeeded).

Next steps

- [Train a model](#)
- [CI/CD for Custom Speech](#)

Additional resources

Documentation

[Human-labeled transcriptions guidelines - Speech service - Azure Cognitive Services](#)

You use human-labeled transcriptions with your audio data to improve speech recognition accuracy. This is especially helpful when words are deleted or incorrectly replaced.

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Deploy a Custom Speech model - Speech service - Azure Cognitive Services](#)

Learn how to deploy Custom Speech models.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[Get batch transcription results - Speech service - Azure Cognitive Services](#)

With batch transcription, the Speech service transcribes the audio data and stores the results in a storage container. You can then retrieve the results from the storage container.

[How to get Speech-to-text Session ID and Transcription ID - Azure Cognitive Services](#)

Learn how to get Speech service Speech-to-text Session ID and Transcription ID

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Training and testing datasets - Speech service - Azure Cognitive Services](#)

Learn about types of training and testing data for a Custom Speech project, along with how to use and manage that data.

[Show 5 more](#)

CI/CD for Custom Speech

Article • 10/20/2022 • 4 minutes to read

Implement automated training, testing, and release management to enable continuous improvement of Custom Speech models as you apply updates to training and testing data. Through effective implementation of CI/CD workflows, you can ensure that the endpoint for the best-performing Custom Speech model is always available.

[Continuous integration](#) (CI) is the engineering practice of frequently committing updates in a shared repository, and performing an automated build on it. CI workflows for Custom Speech train a new model from its data sources and perform automated testing on the new model to ensure that it performs better than the previous model.

[Continuous delivery](#) (CD) takes models from the CI process and creates an endpoint for each improved Custom Speech model. CD makes endpoints easily available to be integrated into solutions.

Custom CI/CD solutions are possible, but for a robust, pre-built solution, use the [Speech DevOps template repository](#), which executes CI/CD workflows using GitHub Actions.

CI/CD workflows for Custom Speech

The purpose of these workflows is to ensure that each Custom Speech model has better recognition accuracy than the previous build. If the updates to the testing and/or training data improve the accuracy, these workflows create a new Custom Speech endpoint.

Git servers such as GitHub and Azure DevOps can run automated workflows when specific Git events happen, such as merges or pull requests. For example, a CI workflow can be triggered when updates to testing data are pushed to the *main* branch. Different Git Servers will have different tooling, but will allow scripting command-line interface (CLI) commands so that they can execute on a build server.

Along the way, the workflows should name and store data, tests, test files, models, and endpoints such that they can be traced back to the commit or version they came from. It is also helpful to name these assets so that it is easy to see which were created after updating testing data versus training data.

CI workflow for testing data updates

The principal purpose of the CI/CD workflows is to build a new model using the training data, and to test that model using the testing data to establish whether the [Word Error Rate](#) (WER) has improved compared to the previous best-performing model (the "benchmark model"). If the new model performs better, it becomes the new benchmark model against which future models are compared.

The CI workflow for testing data updates should retest the current benchmark model with the updated test data to calculate the revised WER. This ensures that when the WER of a new model is compared to the WER of the benchmark, both models have been tested against the same test data and you're comparing like with like.

This workflow should trigger on updates to testing data and:

- Test the benchmark model against the updated testing data.
- Store the test output, which contains the WER of the benchmark model, using the updated data.
- The WER from these tests will become the new benchmark WER that future models must beat.
- The CD workflow does not execute for updates to testing data.

CI workflow for training data updates

Updates to training data signify updates to the custom model.

This workflow should trigger on updates to training data and:

- Train a new model with the updated training data.
- Test the new model against the testing data.
- Store the test output, which contains the WER.
- Compare the WER from the new model to the WER from the benchmark model.
- If the WER does not improve, stop the workflow.
- If the WER improves, execute the CD workflow to create a Custom Speech endpoint.

CD workflow

After an update to the training data improves a model's recognition, the CD workflow should automatically execute to create a new endpoint for that model, and make that endpoint available such that it can be used in a solution.

Release management

Most teams require a manual review and approval process for deployment to a production environment. For a production deployment, you might want to make sure it happens when key people on the development team are available for support, or during low-traffic periods.

Tools for Custom Speech workflows

Use the following tools for CI/CD automation workflows for Custom Speech:

- [Azure CLI](#) to create an Azure service principal authentication, query Azure subscriptions, and store test results in Azure Blob.
- [Azure Speech CLI](#) to interact with the Speech Service from the command line or an automated workflow.

DevOps solution for Custom Speech using GitHub Actions

For an already-implemented DevOps solution for Custom Speech, go to the [Speech DevOps template repo ↗](#). Create a copy of the template and begin development of custom models with a robust DevOps system that includes testing, training, and versioning using GitHub Actions. The repository provides sample testing and training data to aid in setup and explain the workflow. After initial setup, replace the sample data with your project data.

The [Speech DevOps template repo ↗](#) provides the infrastructure and detailed guidance to:

- Copy the template repository to your GitHub account, then create Azure resources and a [service principal](#) for the GitHub Actions CI/CD workflows.
- Walk through the "[dev inner loop](#)." Update training and testing data from a feature branch, test the changes with a temporary development model, and raise a pull request to propose and review the changes.
- When training data is updated in a pull request to *main*, train models with the GitHub Actions CI workflow.
- Perform automated accuracy testing to establish a model's [Word Error Rate](#) (WER). Store the test results in Azure Blob.
- Execute the CD workflow to create an endpoint when the WER improves.

Next steps

- Use the [Speech DevOps template repo](#) to implement DevOps for Custom Speech with GitHub Actions.

Improve recognition accuracy with phrase list

Article • 01/05/2023 • 3 minutes to read

A phrase list is a list of words or phrases provided ahead of time to help improve their recognition. Adding a phrase to a phrase list increases its importance, thus making it more likely to be recognized.

For supported phrase list locales, see [Language and voice support for the Speech service](#).

Examples of phrases include:

- Names
- Geographical locations
- Homonyms
- Words or acronyms unique to your industry or organization

Phrase lists are simple and lightweight:

- **Just-in-time:** A phrase list is provided just before starting the speech recognition, eliminating the need to train a custom model.
- **Lightweight:** You don't need a large data set. Simply provide a word or phrase to boost its recognition.

You can use phrase lists with the [Speech Studio](#), [Speech SDK](#), or [Speech Command Line Interface \(CLI\)](#). The [Batch transcription API](#) doesn't support phrase lists.

You can use phrase lists with both standard and [custom speech](#). There are some situations where training a custom model that includes phrases is likely the best option to improve accuracy. For example, in the following cases you would use Custom Speech:

- If you need to use a large list of phrases. A phrase list shouldn't have more than 500 phrases.
- If you need a phrase list for languages that are not currently supported.

Try it in Speech Studio

You can use [Speech Studio](#) to test how phrase list would help improve recognition for your audio. To implement a phrase list with your application in production, you'll use the Speech SDK or Speech CLI.

For example, let's say that you want the Speech service to recognize this sentence: "Hi Rehaan, this is Jessie from Contoso bank."

After testing, you might find that it's incorrectly recognized as: "Hi **everyone**, this is Jesse from **can't do so bank**."

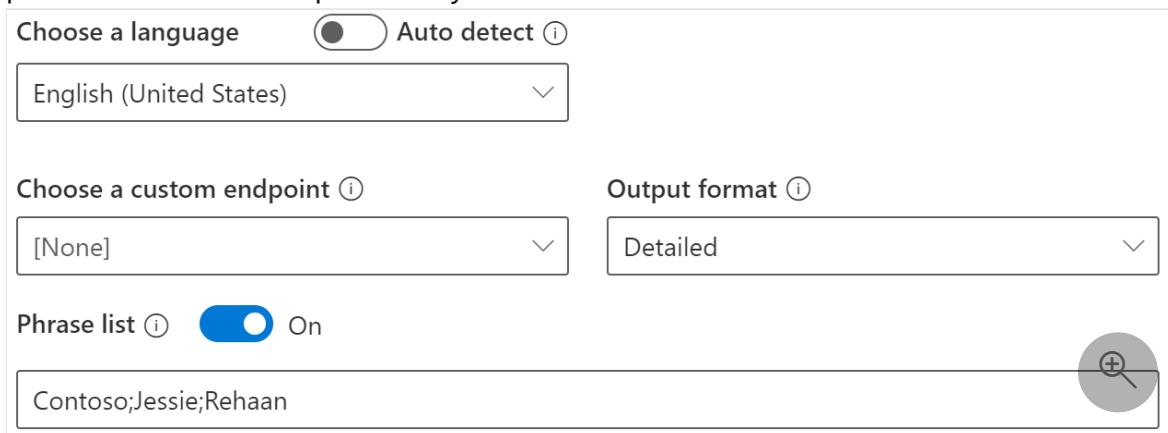
In this case you would want to add "Rehaan", "Jessie", and "Contoso" to your phrase list. Then the names should be recognized correctly.

Now try Speech Studio to see how phrase list can improve recognition accuracy.

① Note

You may be prompted to select your Azure subscription and Speech resource, and then acknowledge billing for your region.

1. Go to **Real-time Speech-to-text** in [Speech Studio](#).
2. You test speech recognition by uploading an audio file or recording audio with a microphone. For example, select **record audio with a microphone** and then say "Hi Rehaan, this is Jessie from Contoso bank." Then select the red button to stop recording.
3. You should see the transcription result in the **Test results** text box. If "Rehaan", "Jessie", or "Contoso" were recognized incorrectly, you can add the terms to a phrase list in the next step.
4. Select **Show advanced options** and turn on **Phrase list**.
5. Enter "Contoso;Jessie;Rehaan" in the phrase list text box. Note that multiple phrases need to be separated by a semicolon.



6. Use the microphone to test recognition again. Otherwise you can select the retry arrow next to your audio file to re-run your audio. The terms "Rehaan", "Jessie", or "Contoso" should be recognized.

Implement phrase list

With the [Speech SDK](#) you can add phrases individually and then run speech recognition.

C#

```
var phraseList = PhraseListGrammar.FromRecognizer(recognizer);
phraseList.AddPhrase("Contoso");
phraseList.AddPhrase("Jessie");
phraseList.AddPhrase("Rehaan");
```

Allowed characters include locale-specific letters and digits, white space characters, and special characters such as +, -, \$, :, (,), {, }, _, ., ?, @, \, ', &, #, %, ^, *, ` , <, >, ;, /. Other special characters are removed internally from the phrase.

Next steps

Check out more options to improve recognition accuracy.

[Custom Speech](#)

Additional resources

Documentation

[azure.cognitiveservices.speech.Recognizer class](#)

Base class for different recognizers

[Asynchronous Conversation Transcription - Speech service - Azure Cognitive Services](#)

Learn how to use asynchronous Conversation Transcription using the Speech service. Available for Java and C# only.

[How to use compressed input audio - Speech service - Azure Cognitive Services](#)

Learn how to use compressed input audio the Speech SDK and CLI.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Real-time Conversation Transcription quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, learn how to transcribe meetings and other conversations. You can add, remove, and identify multiple participants by streaming audio to the Speech service.

[Speech-to-text FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions about the speech-to-text service.

[SpeechRecognizer class](#)

Performs speech recognition from microphone, file, or other audio input streams, and gets transcribed text as result.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Show 5 more](#)

Speech-to-text FAQ

FAQ

This article answers commonly asked questions about the speech-to-text service. If you can't find answers to your questions here, check out [other support options](#).

General

What is the difference between a base model and a custom speech-to-text model?

A baseline speech-to-text model has been trained with Microsoft-owned data and is already deployed in the cloud. You can create and use a custom model to better fit an environment that has specific ambient noise or language. Factory floors, cars, or noisy streets would require an adapted acoustic model. Topics such as biology, physics, radiology, product names, and custom acronyms would require an adapted language model. If you want to train a custom model, you should start with related text to improve the recognition of special terms and phrases.

Where do I start if I want to use a base model?

First, get a Speech resource key and region in the [Azure portal](#). If you want to make REST calls to a predeployed base model, see the [REST APIs](#) documentation. If you want to use WebSockets, [download the Speech SDK](#).

Do I always need to build a custom speech model?

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 base model versus a custom model.

How will I know when the processing for my dataset or model is complete?

Currently, the only way to know is to view the status of the model or dataset in the table. When the processing is complete, the status is *Succeeded*.

Can I create more than one model?

There's no limit to the number of models you can have in your collection.

I realize that I've made a mistake. How do I cancel a data import or model creation that's in progress?

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.

I get several results for each phrase with the detailed output format. Which one should I use?

Always take the first result, even if another result ("N-Best") might have a higher confidence value. Speech service considers the first result to be the best. The result can also be an empty string if no speech was recognized.

The other results are likely worse and might not have full capitalization and punctuation applied. These results are most useful in special scenarios, such as giving users the option to pick corrections from a list or handling incorrectly recognized commands.

Why are there multiple base models?

You can choose from more than one base model in Speech service. Each model name contains the date when it was added. When you start training a custom model, use the most recent model to get the best accuracy. Older base models are still available for some time after a new model is made available. You can continue using the model that you've worked with until it is retired (see [Model and endpoint lifecycle](#)). We still recommend that you switch to the latest base model for better accuracy.

Can I update my existing model (model stacking)?

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 the adaptation is finished, redeploy the new, updated model to obtain a new endpoint.

When a new version of a base model is available, is my deployment automatically updated?

Deployments are *not* automatically updated.

If you've adapted and deployed a model, the existing deployment will remain as is. You can decommission the deployed model, readapt it by using the newer version of the base model, and redeploy it for better accuracy.

Both base models and custom models will be retired after some time (see [Model and endpoint lifecycle](#)).

Can I download my model and run it locally?

You can run a custom model locally in a [Docker container](#).

Can I copy or move my datasets, models, and deployments to another region or subscription?

You can use the [REST API ↗](#) to copy a custom model to another region or subscription. Datasets and deployments can't be copied. You can import a dataset again in another subscription and create endpoints there by using the model copies.

Are my requests logged?

By default, requests aren't logged (neither audio nor transcription). If necessary, you can select the **Log content from this endpoint** option when you [create a custom endpoint](#). You can also enable audio logging in the [Speech SDK](#) on a per-request basis, without having to create a custom endpoint. In both cases, audio and recognition results of

requests will be stored in secure storage. Subscriptions that use Microsoft-owned storage will be available for 30 days.

You can export the logged files on the deployment page in Speech Studio if you use a custom endpoint with **Log content from this endpoint** enabled. If audio logging is enabled via the SDK, call the API to access the files. You can also use API to [delete the logs](#) any time.

Are my requests throttled?

For information, see [Speech service quotas and limits](#).

How am I charged for dual channel audio?

If you submit each channel separately in their own file, you'll be charged for the duration of each file. If you submit a single file with the channels multiplexed together, you'll be charged for the duration of the single file. For more information about pricing, see the [Azure Cognitive Services pricing page](#).

 **Important**

If you have further privacy concerns that prevent you from using the Custom Speech service, contact one of the support channels.

Increasing concurrency

For information, see [Speech service quotas and limits](#).

Importing data

What is the limit to the size of a dataset, and why is it the limit?

The limit is because of the restriction on the size of files for HTTP upload. For the actual limit, see [Speech service quotas and limits](#). You can split your data into multiple datasets and select all of them to train the model.

Can I zip (compress) my text files so that I can upload a larger text file?

No. Currently, only uncompressed text files are allowed.

The data report says there were failed utterances. What is the issue?

A failure to upload 100 percent of the utterances in a file isn't 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 still recommend that you try to understand why the utterances failed and then fix the problem. Most common problems, such as formatting errors, are easy to fix.

Creating an acoustic model

How much acoustic data do I need?

We recommend starting with from 30 minutes to 1 hour of acoustic data.

What data should I collect?

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.

How should I collect acoustic data?

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.

Do I need to transcribe adaptation data myself?

Yes. You can transcribe it yourself or use a professional transcription service. Some users prefer professional transcribers, and others use crowdsourcing or transcribe the data themselves.

How long does it take to train a custom model with audio data?

Training a model with audio data can be a lengthy process. Depending on the amount of data, it can take several days to create a custom model. If it can't be finished within one week, the service might abort the training operation and report the model as failed.

In general, Speech service processes approximately 10 hours of audio data per day in regions that have dedicated hardware. It can process only about 1 hour of audio data per day in other regions. Training with text only is much faster and ordinarily finishes within minutes.

Use one of the regions where dedicated hardware is available for training. Speech service will use up to 20 hours of audio for training in these regions. In other regions, Speech service will use only up to 8 hours.

Accuracy testing

What is word error rate (WER), and how is it computed?

WER is the evaluation metric for speech recognition. WER is calculated as the total number of errors (insertions, deletions, and substitutions), divided by the total number of words in the reference transcription. For more information, see [Test model quantitatively](#).

How do I determine whether the results of an accuracy test are good?

The results show a comparison between the base model and the model you customized. To make customization worthwhile, you should aim to beat the base model.

How do I determine the WER of a base model so I can see whether it improved?

The offline test results show the baseline accuracy of the custom model and the improvement over baseline.

Creating a language model

How much text data do I need to upload?

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, providing many examples is useful because it tells the system to listen for these terms also. It's common to have at least 100 and, ordinarily, 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.

Can I simply upload a list of words?

Uploading a list of words adds them to the vocabulary, but it doesn't teach the system how the words are ordinarily 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're 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.

Next steps

- [Troubleshoot the Speech SDK](#)
- [Speech service release notes](#)

Text-to-speech documentation

Text-to-speech from the Speech service enables your applications, tools, or devices to convert text into human-like synthesized speech.

About text-to-speech

OVERVIEW

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

[Use the Speech CLI for text-to-speech with no code](#)

QUICKSTART

[Get started with text-to-speech](#)

Develop with text-to-speech

HOW-TO GUIDE

[Improve synthesis with SSML](#)

[Batch synthesis for long-form text](#)

CONCEPT

[What is Custom Neural Voice?](#)

[Get started with Custom Voice](#)

[Create and use Custom Voice models](#)

Reference

REFERENCE

[Neural voice support](#)

[SSML phonetic sets](#)

[Text-to-speech pricing ↗](#)

What is text-to-speech?

Article • 01/13/2023 • 6 minutes to read

In this overview, you learn about the benefits and capabilities of the text-to-speech feature of the Speech service, which is part of Azure Cognitive Services.

Text-to-speech enables your applications, tools, or devices to convert text into humanlike synthesized speech. The text-to-speech capability is also known as speech synthesis. Use humanlike prebuilt neural voices out of the box, or create a custom neural voice that's unique to your product or brand. For a full list of supported voices, languages, and locales, see [Language and voice support for the Speech service](#).

Core features

Text-to-speech includes the following features:

Feature	Summary	Demo
Prebuilt neural voice (called Neural on the pricing page)	Highly natural out-of-the-box voices. Create an Azure account and Speech service subscription, and then use the Speech SDK or visit the Speech Studio portal and select prebuilt neural voices to get started. Check the pricing details .	Check the Voice Gallery and determine the right voice for your business needs.
Custom Neural Voice (called Custom Neural on the pricing page)	Easy-to-use self-service for creating a natural brand voice, with limited access for responsible use. Create an Azure account and Speech service subscription (with the S0 tier), and apply to use the custom neural feature. After you've been granted access, visit the Speech Studio portal and select Custom Voice to get started. Check the pricing details .	Check the voice samples .

More about neural text-to-speech features

The text-to-speech feature of the Speech service on Azure has been fully upgraded to the neural text-to-speech engine. This engine uses deep neural networks to make the voices of computers nearly indistinguishable from the recordings of people. With the clear articulation of words, neural text-to-speech significantly reduces listening fatigue when users interact with AI systems.

The patterns of stress and intonation in spoken language are called *prosody*. Traditional text-to-speech systems break down prosody into separate linguistic analysis and acoustic prediction steps that are governed by independent models. That can result in muffled, buzzy voice synthesis.

Here's more information about neural text-to-speech features in the Speech service, and how they overcome the limits of traditional text-to-speech systems:

- **Real-time speech synthesis:** Use the [Speech SDK](#) or [REST API](#) to convert text-to-speech by using [prebuilt neural voices](#) or [custom neural voices](#).
- **Asynchronous synthesis of long audio:** Use the [batch synthesis API](#) (Preview) to asynchronously synthesize text-to-speech files longer than 10 minutes (for example, audio books or lectures). Unlike synthesis performed via 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 synthesized audio is downloaded when the service makes it available.
- **Prebuilt neural voices:** Microsoft neural text-to-speech capability uses deep neural networks to overcome the limits of traditional speech synthesis with regard to stress and intonation in spoken language. Prosody prediction and voice synthesis happen simultaneously, which results in more fluid and natural-sounding outputs. Each prebuilt neural voice model is available at 24kHz and high-fidelity 48kHz. You can use neural voices to:
 - Make interactions with chatbots and voice assistants more natural and engaging.
 - Convert digital texts such as e-books into audiobooks.
 - Enhance in-car navigation systems.

For a full list of platform neural voices, see [Language and voice support for the Speech service](#).

- **Fine-tuning text-to-speech output with SSML:** Speech Synthesis Markup Language (SSML) is an XML-based markup language that's used to customize text-to-speech outputs. With SSML, you can adjust pitch, add pauses, improve pronunciation, change speaking rate, adjust volume, and attribute multiple voices to a single document.

You can use SSML to define your own lexicons or switch to different speaking styles. With the [multilingual voices](#), you can also adjust the speaking languages via SSML. To fine-tune the voice output for your scenario, see [Improve synthesis with Speech Synthesis Markup Language](#) and [Speech synthesis with the Audio Content Creation tool](#).

- **Visemes:** Visemes are the key poses in observed speech, including the position of the lips, jaw, and tongue in producing a particular phoneme. Visemes have a strong correlation with voices and phonemes.

By using viseme events in Speech SDK, you can generate facial animation data. This data can be used to animate faces in lip-reading communication, education, entertainment, and customer service. Viseme is currently supported only for the en-US (US English) [neural voices](#).

Note

We plan to retire the traditional/standard voices and non-neural custom voice in 2024. After that, we'll no longer support them.

If your applications, tools, or products are using any of the standard voices and custom voices, you must migrate to the neural version. For more information, see [Migrate to neural voices](#).

Get started

To get started with text-to-speech, see the [quickstart](#). Text-to-speech is available via the [Speech SDK](#), the [REST API](#), and the [Speech CLI](#).

Tip

To convert text-to-speech with a no-code approach, try the [Audio Content Creation tool](#) in [Speech Studio](#).

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 \(REST\)](#)

Custom Neural Voice

In addition to prebuilt neural voices, you can create and fine-tune custom neural voices that are unique to your product or brand. All it takes to get started is a handful of audio files and the associated transcriptions. For more information, see [Get started with Custom Neural Voice](#).

Pricing note

Billable characters

When you use the text-to-speech feature, you're billed for each character that's converted to speech, including punctuation. Although 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 feature 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 [Speech service pricing](#).

Important

Each Chinese character is counted as two characters for billing, including kanji used in Japanese, hanja used in Korean, or hanzi used in other languages.

Model training and hosting time for Custom Neural Voice

Custom Neural Voice training and hosting are both calculated by hour and billed per second. For the billing unit price, see [Speech service pricing](#).

Custom Neural Voice (CNV) training time is measured by 'compute hour' (a unit to measure machine running time). Typically, when training a voice model, two computing tasks are running in parallel. So, the calculated compute hours will be longer than the actual training time. On average, it takes less than one compute hour to train a CNV Lite voice; while for CNV Pro, it usually takes 20 to 40 compute hours to train a single-style voice, and around 90 compute hours to train a multi-style voice. The CNV training time

is billed with a cap of 96 compute hours. So in the case that a voice model is trained in 98 compute hours, you will only be charged with 96 compute hours.

Custom Neural Voice (CNV) endpoint hosting is measured by the actual time (hour). The hosting time (hours) for each endpoint is calculated at 00:00 UTC every day for the previous 24 hours. For example, if the endpoint has been active for 24 hours on day one, it will be billed for 24 hours at 00:00 UTC the second day. If the endpoint is newly created or has been suspended during the day, it will be billed for its accumulated running time until 00:00 UTC the second day. If the endpoint is not currently hosted, it will not be billed. In addition to the daily calculation at 00:00 UTC each day, the billing is also triggered immediately when an endpoint is deleted or suspended. For example, for an endpoint created at 08:00 UTC on December 1, the hosting hour will be calculated to 16 hours at 00:00 UTC on December 2 and 24 hours at 00:00 UTC on December 3. If the user suspends hosting the endpoint at 16:30 UTC on December 3, the duration (16.5 hours) from 00:00 to 16:30 UTC on December 3 will be calculated for billing.

Reference docs

- [Speech SDK](#)
- [REST API: Text-to-speech](#)

Next steps

- [Text to speech quickstart](#)
- [Get the Speech SDK](#)

Additional resources

Documentation

[Custom Neural Voice overview - Speech service - Azure Cognitive Services](#)

Custom Neural Voice is a text-to-speech feature that allows you to create a one-of-a-kind, customized, synthetic voice for your applications. You provide your own audio data as a sample.

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Speech Studio overview - Speech service - Azure Cognitive Services](#)

Speech Studio is a set of UI-based tools for building and integrating features from Azure Speech service in your applications.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Text-to-speech API reference \(REST\) - Speech service - Azure Cognitive Services](#)

Learn how to use the REST API to convert text into synthesized speech.

[Quickstart: The Speech CLI - Speech service - Azure Cognitive Services](#)

In this Azure Speech CLI quickstart, you interact with speech-to-text, text-to-speech, and speech translation without having to write code.

[Speech-to-text quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you convert speech to text with recognition from a microphone.

[Show 5 more](#)

Quickstart: Convert text to speech

Article • 09/20/2022 • 46 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this quickstart, you run an application that does text to speech synthesis.

💡 Tip

You can try text-to-speech in [Speech Studio](#) without signing up or writing any code.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

[I ran into an issue](#)

Set up the environment

The Speech SDK is available as a [NuGet package](#) and implements .NET Standard 2.0. You install the Speech SDK later in this guide, but first check the [SDK installation guide](#) for any more requirements.

Set environment variables

Your application must be authenticated to access Cognitive Services resources. For production, use a secure way of storing and accessing your credentials. For example, after you [get a key for your Speech resource](#), write it to a new environment variable on the local machine running the application.

💡 Tip

Don't include the key directly in your code, and never post it publicly. See the Cognitive Services [security](#) article for more authentication options like [Azure Key](#)

Vault

To set the environment variable for your Speech resource key, open a console window, and follow the instructions for your operating system and development environment. To set the `SPEECH_KEY` environment variable, replace `your-key` with one of the keys for your resource.

Windows

Console

```
setx SPEECH_KEY your-key
```

➊ Note

If you only need to access the environment variable in the current running console, you can set the environment variable with `set` instead of `setx`.

After you add the environment variable, you may need to restart any running programs that will need to read the environment variable, including the console window. For example, if you are using Visual Studio as your editor, restart Visual Studio before running the example.

To set the environment variable for your Speech resource region, follow the same steps. Set `SPEECH_REGION` to the region of your resource. For example, `westus`.

I ran into an issue

Synthesize to speaker output

Follow these steps to create a new console application and install the Speech SDK.

1. Open a command prompt where you want the new project, and create a console application with the .NET CLI. The `Program.cs` file should be created in the project directory.

.NET CLI

```
dotnet new console
```

2. Install the Speech SDK in your new project with the .NET CLI.

```
.NET CLI
```

```
dotnet add package Microsoft.CognitiveServices.Speech
```

3. Replace the contents of `Program.cs` with the following code.

```
C#
```

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

class Program
{
    // This example requires environment variables named "SPEECH_KEY"
    // and "SPEECH_REGION"
    static string speechKey =
        Environment.GetEnvironmentVariable("SPEECH_KEY");
    static string speechRegion =
        Environment.GetEnvironmentVariable("SPEECH_REGION");

    static void OutputSpeechSynthesisResult(SpeechSynthesisResult
        speechSynthesisResult, string text)
    {
        switch (speechSynthesisResult.Reason)
        {
            case ResultReason.SynthesizingAudioCompleted:
                Console.WriteLine($"Speech synthesized for text:
[{text}]");
                break;
            case ResultReason.Canceled:
                var cancellation =
                    SpeechSynthesisCancellationDetails.FromResult(speechSynthesisResult);
                Console.WriteLine($"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 set the
speech resource key and region values?");
                }
                break;
            default:
                break;
        }
    }
}
```

```

        }

    }

    async static Task Main(string[] args)
    {
        var speechConfig = SpeechConfig.FromSubscription(speechKey,
speechRegion);

        // The language of the voice that speaks.
        speechConfig.SpeechSynthesisVoiceName = "en-US-JennyNeural";

        using (var speechSynthesizer = new
SpeechSynthesizer(speechConfig))
        {
            // Get text from the console and synthesize to the default
            speaker.
            Console.WriteLine("Enter some text that you want to speak
>");
            string text = Console.ReadLine();

            var speechSynthesisResult = await
speechSynthesizer.SpeakTextAsync(text);
            OutputSpeechSynthesisResult(speechSynthesisResult, text);
        }

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

```

4. To change the speech synthesis language, replace `en-US-JennyNeural` with another [supported voice](#). All neural voices are multilingual and fluent in their own language and English. For example, if the input text in English is "I'm excited to try text to speech" and you set `es-ES-ElviraNeural`, the text is spoken in English with a Spanish accent. If the voice does not speak the language of the input text, the Speech service won't output synthesized audio.

[Build and run](#) your new console application to start speech synthesis to the default speaker.

Console

`dotnet run`

ⓘ Important

Make sure that you set the `SPEECH_KEY` and `SPEECH_REGION` environment variables as described [above](#). If you don't set these variables, the sample will fail with an

error message.

Enter some text that you want to speak. For example, type "I'm excited to try text to speech." Press the Enter key to hear the synthesized speech.

Console

```
Enter some text that you want to speak >
I'm excited to try text to speech
```

I ran into an issue

⚠ Warning

There is a known issue on Windows 11 that might affect some types of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) connections. For more information, see the [troubleshooting guide](#).

Remarks

Now that you've completed the quickstart, here are some additional considerations:

This quickstart uses the `SpeakTextAsync` operation to synthesize a short block of text that you enter. You can also get text from files as described in these guides:

- For information about speech synthesis from a file and finer control over voice styles, prosody, and other settings, see [How to synthesize speech](#) and [Improve synthesis with Speech Synthesis Markup Language \(SSML\)](#).
- For information about synthesizing long-form text to speech, see [batch synthesis](#).

Clean up resources

You can use the [Azure portal](#) or [Azure Command Line Interface \(CLI\)](#) to remove the Speech resource you created.

Next steps

[Learn more about speech synthesis](#)

How to synthesize speech from text

Article • 10/27/2022 • 68 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) ↗ | [Additional Samples on GitHub](#) ↗

In this how-to guide, you learn common design patterns for doing text-to-speech synthesis.

See the text-to-speech [overview](#) for more information about:

- Getting responses as in-memory streams.
- Customizing output sample rate and bit rate.
- Submitting synthesis requests by using Speech Synthesis Markup Language (SSML).
- Using neural voices.
- Subscribing to events and acting on results.

Select synthesis language and voice

The text-to-speech feature in the Azure Speech service supports more than 270 voices and more than 110 languages and variants. You can get the [full list](#) or try them in a [text-to-speech demo](#) ↗.

Specify the language or voice of `SpeechConfig` to match your input text and use the wanted voice:

C#

```
static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
    // Set either the `SpeechSynthesisVoiceName` or
    `SpeechSynthesisLanguage`.
    speechConfig.SpeechSynthesisLanguage = "en-US";
    speechConfig.SpeechSynthesisVoiceName = "en-US-JennyNeural";
}
```

All neural voices are multilingual and fluent in their own language and English. For example, if the input text in English is "I'm excited to try text to speech" and you set `es-ES-ElviraNeural`, the text is spoken in English with a Spanish accent. If the voice doesn't speak the language of the input text, the Speech service won't output synthesized audio. See the [full list](#) of supported neural voices.

ⓘ Note

The default voice is the first voice returned per locale via the [Voice List API](#).

The voice that speaks is determined in order of priority as follows:

- If you don't set `SpeechSynthesisVoiceName` or `SpeechSynthesisLanguage`, the default voice for `en-US` will speak.
- If you only set `SpeechSynthesisLanguage`, the default voice for the specified locale will speak.
- If both `SpeechSynthesisVoiceName` and `SpeechSynthesisLanguage` are set, the `SpeechSynthesisLanguage` setting is ignored. The voice that you specified via `SpeechSynthesisVoiceName` will speak.
- If the voice element is set via [Speech Synthesis Markup Language \(SSML\)](#), the `SpeechSynthesisVoiceName` and `SpeechSynthesisLanguage` settings are ignored.

Synthesize speech to a file

Next, you create a `SpeechSynthesizer` object. This object executes text-to-speech conversions and outputs to speakers, files, or other output streams. `SpeechSynthesizer` accepts as parameters:

- The `SpeechConfig` object that you created in the previous step
- An `AudioConfig` object that specifies how output results should be handled

To start, create an `AudioConfig` instance to automatically write the output to a .wav file by using the `FromWavFileOutput()` function. Instantiate it with a `using` statement. A `using` statement in this context automatically disposes of unmanaged resources and causes the object to go out of scope after disposal.

C#

```
static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
    "YourSpeechRegion");
    using var audioConfig =
        AudioConfig.FromWavFileOutput("path/to/write/file.wav");
}
```

Next, instantiate a `SpeechSynthesizer` instance with another `using` statement. Pass your `speechConfig` object and the `audioConfig` object as parameters. Then, the process of

executing speech synthesis and writing to a file is as simple as running `SpeakTextAsync()` with a string of text.

C#

```
static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
    using var audioConfig =
AudioConfig.FromWavFileOutput("path/to/write/file.wav");
    using var synthesizer = new SpeechSynthesizer(speechConfig,
audioConfig);
    await synthesizer.SpeakTextAsync("I'm excited to try text-to-speech");
}
```

Run the program. A synthesized .wav file is written to the location that you specified. This is a good example of the most basic usage. Next, you look at customizing output and handling the output response as an in-memory stream for working with custom scenarios.

Synthesize to speaker output

To output synthesized speech to the current active output device such as a speaker, omit the `AudioConfig` parameter when you're creating the `SpeechSynthesizer` instance. Here's an example:

C#

```
static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
    using var synthesizer = new SpeechSynthesizer(speechConfig);
    await synthesizer.SpeakTextAsync("I'm excited to try text to speech");
}
```

Get a result as an in-memory stream

You can use the resulting audio data as an in-memory stream rather than directly writing to a file. With in-memory stream, you can build custom behavior, including:

- Abstract the resulting byte array as a seekable stream for custom downstream services.

- Integrate the result with other APIs or services.
- Modify the audio data, write custom .wav headers, and do related tasks.

It's simple to make this change from the previous example. First, remove the `AudioConfig` block, because you'll manage the output behavior manually from this point onward for increased control. Then pass `null` for `AudioConfig` in the `SpeechSynthesizer` constructor.

ⓘ Note

Passing `null` for `AudioConfig`, rather than omitting it as you did in the previous speaker output example, will not play the audio by default on the current active output device.

This time, save the result to a `SpeechSynthesisResult` variable. The `AudioData` property contains a `byte []` instance for the output data. You can work with this `byte []` instance manually, or you can use the `AudioDataStream` class to manage the in-memory stream. In this example, you use the `AudioDataStream.FromResult()` static function to get a stream from the result:

C#

```
static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
    "YourSpeechRegion");
    using var synthesizer = new SpeechSynthesizer(speechConfig, null);

    var result = await synthesizer.SpeakTextAsync("I'm excited to try text-
    to-speech");
    using var stream = AudioDataStream.FromResult(result);
}
```

From here, you can implement any custom behavior by using the resulting `stream` object.

Customize audio format

You can customize audio output attributes, including:

- Audio file type
- Sample rate
- Bit depth

To change the audio format, you use the `SetSpeechSynthesisOutputFormat()` function on the `SpeechConfig` object. This function expects an `enum` instance of type `SpeechSynthesisOutputFormat`, which you use to select the output format. See the [list of audio formats](#) that are available.

There are various options for different file types, depending on your requirements. By definition, raw formats like `Raw24Khz16BitMonoPcm` don't include audio headers. Use raw formats only in one of these situations:

- You know that your downstream implementation can decode a raw bitstream.
- You plan to manually build headers based on factors like bit depth, sample rate, and number of channels.

In this example, you specify the high-fidelity RIFF format `Riff24Khz16BitMonoPcm` by setting `SpeechSynthesisOutputFormat` on the `SpeechConfig` object. Similar to the example in the previous section, you use `AudioDataStream` to get an in-memory stream of the result, and then write it to a file.

C#

```
static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");

    speechConfig.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff
24Khz16BitMonoPcm);

    using var synthesizer = new SpeechSynthesizer(speechConfig, null);
    var result = await synthesizer.SpeakTextAsync("I'm excited to try text-
to-speech");

    using var stream = AudioDataStream.FromResult(result);
    await stream.SaveToWaveFileAsync("path/to/write/file.wav");
}
```

Running your program again will write a .wav file to the specified path.

Use SSML to customize speech characteristics

You can use SSML to fine-tune the pitch, pronunciation, speaking rate, volume, and more in the text-to-speech output by submitting your requests from an XML schema. This section shows an example of changing the voice. For a more detailed guide, see the [SSML how-to article](#).

To start using SSML for customization, you make a simple change that switches the voice.

First, create a new XML file for the SSML configuration in your root project directory. In this example, it's `ssml.xml`. The root element is always `<speak>`. Wrapping the text in a `<voice>` element allows you to change the voice by using the `name` parameter. See the [full list](#) of supported *neural* voices.

XML

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    When you're on the freeway, it's a good idea to use a GPS.
  </voice>
</speak>
```

Next, you need to change the speech synthesis request to reference your XML file. The request is mostly the same, but instead of using the `SpeakTextAsync()` function, you use `SpeakSsmlAsync()`. This function expects an XML string, so you first load your SSML configuration as a string by using `File.ReadAllText()`. From here, the result object is exactly the same as previous examples.

ⓘ Note

If you're using Visual Studio, your build configuration likely won't find your XML file by default. To fix this, right-click the XML file and select **Properties**. Change **Build Action** to **Content**, and change **Copy to Output Directory** to **Copy always**.

C#

```
public static async Task SynthesizeAudioAsync()
{
    var speechConfig = SpeechConfig.FromSubscription("YourSpeechKey",
"YourSpeechRegion");
    using var synthesizer = new SpeechSynthesizer(speechConfig, null);

    var ssml = File.ReadAllText("./ssml.xml");
    var result = await synthesizer.SpeakSsmlAsync(ssml);

    using var stream = AudioDataStream.FromResult(result);
    await stream.SaveToWaveFileAsync("path/to/write/file.wav");
}
```

(!) Note

To change the voice without using SSML, you can set the property on `SpeechConfig` by using `SpeechConfig.SpeechSynthesisVoiceName = "en-US-JennyNeural";`.

Subscribe to synthesizer events

You might want more insights about the text-to-speech processing and results. For example, you might want to know when the synthesizer starts and stops, or you might want to know about other events encountered during synthesis.

While using the [SpeechSynthesizer](#) for text-to-speech, you can subscribe to the events in this table:

Event	Description	Use case
<code>BookmarkReached</code>	Signals that a bookmark was reached. To trigger a bookmark reached event, a <code>bookmark</code> element is required in the SSML . This event reports the output audio's elapsed time between the beginning of synthesis and the <code>bookmark</code> element. The event's <code>Text</code> property is the string value that you set in the bookmark's <code>mark</code> attribute. The <code>bookmark</code> elements won't be spoken.	You can use the <code>bookmark</code> element to insert custom markers in SSML to get the offset of each marker in the audio stream. The <code>bookmark</code> element can be used to reference a specific location in the text or tag sequence.
<code>SynthesisCanceled</code>	Signals that the speech synthesis was canceled.	You can confirm when synthesis has been canceled.
<code>SynthesisCompleted</code>	Signals that speech synthesis has completed.	You can confirm when synthesis has completed.
<code>SynthesisStarted</code>	Signals that speech synthesis has started.	You can confirm when synthesis has started.
<code>Synthesizing</code>	Signals that speech synthesis is ongoing. This event fires each time the SDK receives an audio chunk from the Speech service.	You can confirm when synthesis is in progress.

Event	Description	Use case
VisemeReceived	Signals that a viseme event was received.	Visemes are often used to represent the key poses in observed speech. Key poses include the position of the lips, jaw, and tongue in producing a particular phoneme. You can use visemes to animate the face of a character as speech audio plays.
WordBoundary	Signals that a word boundary was received. This event is raised at the beginning of each new spoken word, punctuation, and sentence. The event reports the current word's time offset (in ticks) from the beginning of the output audio. This event also reports the character position in the input text (or SSML) immediately before the word that's about to be spoken.	This event is commonly used to get relative positions of the text and corresponding audio. You might want to know about a new word, and then take action based on the timing. For example, you can get information that can help you decide when and for how long to highlight words as they're spoken.

ⓘ Note

Events are raised as the output audio data becomes available, which will be faster than playback to an output device. The caller must appropriately synchronize streaming and real time.

Here's an example that shows how to subscribe to events for speech synthesis. You can follow the instructions in the [quickstart](#), but replace the contents of that `Program.cs` file with the following C# code.

C#

```
using Microsoft.CognitiveServices.Speech;

class Program
{
    // This example requires environment variables named "SPEECH_KEY" and
    "SPEECH_REGION"
    static string speechKey =
        Environment.GetEnvironmentVariable("SPEECH_KEY");
    static string speechRegion =
        Environment.GetEnvironmentVariable("SPEECH_REGION");

    async static Task Main(string[] args)
    {
```

```

        var speechConfig = SpeechConfig.FromSubscription(speechKey,
speechRegion);

        var speechSynthesisVoiceName = "en-US-JennyNeural";
        var ssml = @$"<speak version='1.0' xml:lang='en-US'
xmlns='http://www.w3.org/2001/10/synthesis'
xmlns:mstts='http://www.w3.org/2001/mstts'>
    <voice name='{speechSynthesisVoiceName}'>
        <mstts:viseme type='redlips_front'/>
        The rainbow has seven colors: <bookmark
mark='colors_list_begin'/>Red, orange, yellow, green, blue, indigo, and
violet.<bookmark mark='colors_list_end'/>.
    </voice>
</speak>";

        // Required for sentence-level WordBoundary events

speechConfig SetProperty(PropertyId.SpeechServiceResponse_RequestSentenceBoundary, "true");

        using (var speechSynthesizer = new SpeechSynthesizer(speechConfig))
{
    // Subscribe to events

    speechSynthesizer.BookmarkReached += (s, e) =>
    {
        Console.WriteLine($"BookmarkReached event:" +
                     $"\\r\\n\\tAudioOffset: {(e.AudioOffset + 5000) / 10000}ms"
+
                     $"\\r\\n\\tText: \"{e.Text}\"");
    };

    speechSynthesizer.SynthesisCanceled += (s, e) =>
    {
        Console.WriteLine("SynthesisCanceled event");
    };

    speechSynthesizer.SynthesisCompleted += (s, e) =>
    {
        Console.WriteLine($"SynthesisCompleted event:" +
                     $"\\r\\n\\tAudioData: {e.Result.AudioData.Length} bytes" +
                     $"\\r\\n\\tAudioDuration: {e.Result.AudioDuration}");
    };

    speechSynthesizer.SynthesisStarted += (s, e) =>
    {
        Console.WriteLine("SynthesisStarted event");
    };

    speechSynthesizer.Synthesizing += (s, e) =>
    {
        Console.WriteLine($"Synthesizing event:" +
                     $"\\r\\n\\tAudioData: {e.Result.AudioData.Length} bytes");
    };
}

```

```

speechSynthesizer.VisemeReceived += (s, e) =>
{
    Console.WriteLine($"VisemeReceived event:" +
        $"\\r\\n\\tAudioOffset: {(e.AudioOffset + 5000) / 10000}ms"
+
        $"\\r\\n\\tVisemeId: {e.VisemeId}");
};

speechSynthesizer.WordBoundary += (s, e) =>
{
    Console.WriteLine($"WordBoundary event:" +
        // Word, Punctuation, or Sentence
        $"\\r\\n\\tBoundaryType: {e.BoundaryType}" +
        $"\\r\\n\\tAudioOffset: {(e.AudioOffset + 5000) / 10000}ms"
+
        $"\\r\\n\\tDuration: {e.Duration}" +
        $"\\r\\n\\tText: \"{e.Text}\"" +
        $"\\r\\n\\tTextOffset: {e.TextOffset}" +
        $"\\r\\n\\tWordLength: {e.WordLength}");
};

// Synthesize the SSML
Console.WriteLine($"SSML to synthesize: \\r\\n{ssml}");
var speechSynthesisResult = await
speechSynthesizer.SpeakSsmlAsync(ssml);

// Output the results
switch (speechSynthesisResult.Reason)
{
    case ResultReason.SynthesizingAudioCompleted:
        Console.WriteLine("SynthesizingAudioCompleted result");
        break;
    case ResultReason.Canceled:
        var cancellation =
SpeechSynthesisCancellationDetails.FromResult(speechSynthesisResult);
        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 set the speech
resource key and region values?");
        }
        break;
    default:
        break;
}
}

Console.WriteLine("Press any key to exit...");
Console.ReadKey();

```

```
    }  
}
```

You can find more text-to-speech samples at [GitHub](#).

Next steps

- Try the text-to-speech quickstart
- Get started with Custom Neural Voice
- Improve synthesis with SSML

Batch synthesis API (Preview) for text to speech

Article • 01/31/2023 • 15 minutes to read

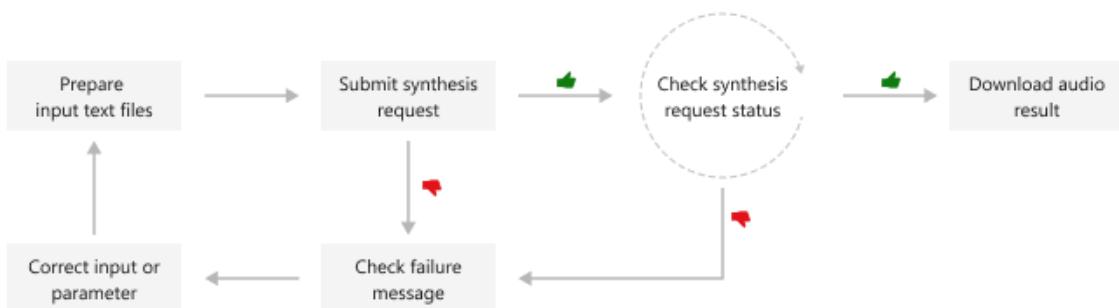
The Batch synthesis API (Preview) can synthesize a large volume of text input (long and short) asynchronously. Publishers and audio content platforms can create long audio content in a batch. For example: audio books, news articles, and documents. The batch synthesis API can create synthesized audio longer than 10 minutes.

ⓘ Important

The Batch synthesis API is currently in public preview. Once it's generally available, the Long Audio API will be deprecated. For more information, see [Migrate to batch synthesis API](#).

The batch synthesis API is asynchronous and doesn't return synthesized audio in real time. You submit text files to be synthesized, poll for the status, and download the audio output when the status indicates success. The text inputs must be plain text or [Speech Synthesis Markup Language \(SSML\)](#) text.

This diagram provides a high-level overview of the workflow.



ⓘ Tip

You can also use the [Speech SDK](#) to create synthesized audio longer than 10 minutes by iterating over the text and synthesizing it in chunks. For a C# example, see [GitHub](#).

You can use the following REST API operations for batch synthesis:

Operation	Method	REST API call
Create batch synthesis	POST	texttospeech/3.1-preview1/batchsynthesis
Get batch synthesis	GET	texttospeech/3.1-preview1/batchsynthesis/{id}
List batch synthesis	GET	texttospeech/3.1-preview1/batchsynthesis
Delete batch synthesis	DELETE	texttospeech/3.1-preview1/batchsynthesis/{id}

For code samples, see [GitHub](#).

Create batch synthesis

To submit a batch synthesis request, construct the HTTP POST request body according to the following instructions:

- Set the required `textType` property.
- If the `textType` property is set to "PlainText", then you must also set the `voice` property in the `synthesisConfig`. In the example below, the `textType` is set to "SSML", so the `speechSynthesis` isn't set.
- Set the required `displayName` property. Choose a name that you can refer to later. The display name doesn't have to be unique.
- Optionally you can set the `description`, `timeToLive`, and other properties. For more information, see [batch synthesis properties](#).

ⓘ Note

The maximum JSON payload size that will be accepted is 500 kilobytes. Each Speech resource can have up to 200 batch synthesis jobs that are running concurrently.

Make an HTTP POST request using the URI as shown in the following example. Replace `YourSpeechKey` with your Speech resource key, replace `YourSpeechRegion` with your Speech resource region, and set the request body properties as previously described.

Azure CLI

```
curl -v -X POST -H "Ocp-Apim-Subscription-Key: YourSpeechKey" -H "Content-Type: application/json" -d '{
  "displayName": "batch synthesis sample",
  "description": "my ssml test",
  "textType": "SSML",
  "inputs": [
```

```

    },
    "text": "<speak version='1.0'> <xml:lang='en-US'>
        <voice xml:lang='en-US' xml:gender='Female'>
            name='en-US-JennyNeural'>
                The rainbow has seven colors.
            </voice>
        </speak>",
    },
],
"properties": {
    "outputFormat": "riff-24khz-16bit-mono-pcm",
    "wordBoundaryEnabled": false,
    "sentenceBoundaryEnabled": false,
    "concatenateResult": false,
    "decompressOutputFiles": false
},
}
}

"https://YourSpeechRegion.customvoice.api.speech.microsoft.com/api/texttospeech/3.1-preview1/batchsynthesis"

```

You should receive a response body in the following format:

JSON

```
{
    "textType": "SSML",
    "synthesisConfig": {},
    "customVoices": {},
    "properties": {
        "timeToLive": "P31D",
        "outputFormat": "riff-24khz-16bit-mono-pcm",
        "concatenateResult": false,
        "decompressOutputFiles": false,
        "wordBoundaryEnabled": false,
        "sentenceBoundaryEnabled": false
    },
    "lastActionDateTime": "2022-11-16T15:07:04.121Z",
    "status": "NotStarted",
    "id": "1e2e0fe8-e403-417c-a382-b55eb2ea943d",
    "createdDateTime": "2022-11-16T15:07:04.121Z",
    "displayName": "batch synthesis sample",
    "description": "my ssml test"
}
```

The `status` property should progress from `NotStarted` status, to `Running`, and finally to `Succeeded` or `Failed`. You can call the [GET batch synthesis API](#) periodically until the returned status is `Succeeded` or `Failed`.

Get batch synthesis

To get the status of the batch synthesis job, make an HTTP GET request using the URI as shown in the following example. Replace `YourSynthesisId` with your batch synthesis ID, replace `YourSpeechKey` with your Speech resource key, and replace `YourSpeechRegion` with your Speech resource region.

Azure CLI

```
curl -v -X GET  
"https://YourSpeechRegion.customvoice.api.speech.microsoft.com/api/texttospe  
ech/3.1-preview1/batchsynthesis/YourSynthesisId" -H "Ocp-Apim-Subscription-  
Key: YourSpeechKey"
```

You should receive a response body in the following format:

JSON

```
{  
    "textType": "SSML",  
    "synthesisConfig": {},  
    "customVoices": {},  
    "properties": {  
        "audioSize": 100000,  
        "durationInTicks": 31250000,  
        "succeededAudioCount": 1,  
        "failedAudioCount": 0,  
        "duration": "PT3.125S",  
        "billingDetails": {  
            "customNeural": 0,  
            "neural": 33  
        },  
        "timeToLive": "P31D",  
        "outputFormat": "riff-24khz-16bit-mono-pcm",  
        "concatenateResult": false,  
        "decompressOutputFiles": false,  
        "wordBoundaryEnabled": false,  
        "sentenceBoundaryEnabled": false  
    },  
    "outputs": {  
        "result": "https://cvoiceprodeus.blob.core.windows.net/batch-  
synthesis-output/41b83de2-380d-45dc-91af-722b68cf8e/results.zip?SAS_Token"  
    },  
    "lastActionDateTime": "2022-11-05T14:00:32.523Z",  
    "status": "Succeeded",  
    "id": "41b83de2-380d-45dc-91af-722b68cf8e",  
    "createdDateTime": "2022-11-05T14:00:31.523Z",  
    "displayName": "batch synthesis sample",  
    "description": "my test"  
}
```

From `outputs.result`, you can download a ZIP file that contains the audio (such as `0001.wav`), summary, and debug details. For more information, see [batch synthesis results](#).

List batch synthesis

To list all batch synthesis jobs for the Speech resource, make an HTTP GET request using the URI as shown in the following example. Replace `YourSpeechKey` with your Speech resource key and replace `YourSpeechRegion` with your Speech resource region.

Optionally, you can set the `skip` and `top` (page size) query parameters in URL. The default value for `skip` is 0 and the default value for `top` is 100.

Azure CLI

```
curl -v -X GET  
"https://YourSpeechRegion.customvoice.api.speech.microsoft.com/api/texttospe  
ech/3.1-preview1/batchsynthesis?skip=0&top=2" -H "Ocp-Apim-Subscription-Key:  
YourSpeechKey"
```

You should receive a response body in the following format:

JSON

```
{  
  "values": [  
    {  
      "textType": "SSML",  
      "synthesisConfig": {},  
      "customVoices": {},  
      "properties": {  
        "audioSize": 100000,  
        "durationInTicks": 31250000,  
        "succeededAudioCount": 1,  
        "failedAudioCount": 0,  
        "duration": "PT3.125S",  
        "billingDetails": {  
          "customNeural": 0,  
          "neural": 33  
        },  
        "timeToLive": "P31D",  
        "outputFormat": "riff-24khz-16bit-mono-pcm",  
        "concatenateResult": false,  
        "decompressOutputFiles": false,  
        "wordBoundaryEnabled": false,  
        "sentenceBoundaryEnabled": false  
      },  
      "outputs": {  
        "result": "https://cvoiceprodeus.blob.core.windows.net/batch-"
```

```
synthesis-output/41b83de2-380d-45dc-91af-722b68cf8e/results.zip?SAS_Token"
},
"lastActionDateTime": "2022-11-05T14:00:32.523Z",
"status": "Succeeded",
"id": "41b83de2-380d-45dc-91af-722b68cf8e",
"createdDateTime": "2022-11-05T14:00:31.523Z",
"displayName": "batch synthesis sample",
"description": "my test"
},
{
"textType": "PlainText",
"synthesisConfig": {
"voice": "en-US-JennyNeural",
"style": "chat",
"rate": "+30.00%",
"pitch": "x-high",
"volume": "80"
},
"customVoices": {},
"properties": {
"audioSize": 79384,
"durationInTicks": 24800000,
"succeededAudioCount": 1,
"failedAudioCount": 0,
"duration": "PT2.48S",
"billingDetails": {
"customNeural": 0,
"neural": 33
},
"timeToLive": "P31D",
"outputFormat": "riff-24khz-16bit-mono-pcm",
"concatenateResult": false,
"decompressOutputFiles": false,
"wordBoundaryEnabled": false,
"sentenceBoundaryEnabled": false
},
"outputs": {
"result": "https://cvoiceprodeus.blob.core.windows.net/batch-
synthesis-output/38e249bf-2607-4236-930b-82f6724048d8/results.zip?SAS_Token"
},
"lastActionDateTime": "2022-11-05T18:52:23.210Z",
"status": "Succeeded",
"id": "38e249bf-2607-4236-930b-82f6724048d8",
"createdDateTime": "2022-11-05T18:52:22.807Z",
"displayName": "batch synthesis sample",
"description": "my test"
},
],
// The next page link of the list of batch synthesis.
"@nextLink":
"https://[region].customvoice.api.speech.microsoft.com/api/texttospeech/3.1-
preview1/batchsynthesis?skip=0&top=2"
}
```

From `outputs.result`, you can download a ZIP file that contains the audio (such as `0001.wav`), summary, and debug details. For more information, see [batch synthesis results](#).

The `values` property in the json response lists your synthesis requests. The list is paginated, with a maximum page size of 100. The `"@nextLink"` property is provided as needed to get the next page of the paginated list.

Delete batch synthesis

Delete the batch synthesis job history after you retrieved the audio output results. The Speech service will keep each synthesis history for up to 31 days, or the duration of the request `timeToLive` property, whichever comes sooner. The date and time of automatic deletion (for synthesis jobs with a status of "Succeeded" or "Failed") is equal to the `lastActionDateTime` + `timeToLive` properties.

To delete a batch synthesis job, make an HTTP DELETE request using the URI as shown in the following example. Replace `YourSynthesisId` with your batch synthesis ID, replace `YourSpeechKey` with your Speech resource key, and replace `YourSpeechRegion` with your Speech resource region.

Azure CLI

```
curl -v -X DELETE  
"https://YourSpeechRegion.customvoice.api.speech.microsoft.com/api/texttospe  
ech/3.1-preview1/batchsynthesis/YourSynthesisId" -H "Ocp-Apim-Subscription-  
Key: YourSpeechKey"
```

The response headers will include `HTTP/1.1 204 No Content` if the delete request was successful.

Batch synthesis results

After you [get a batch synthesis job](#) with `status` of "Succeeded", you can download the audio output results. Use the URL from the `outputs.result` property of the [get batch synthesis](#) response.

To get the batch synthesis results file, make an HTTP GET request using the URI as shown in the following example. Replace `YourOutputsResultUrl` with the URL from the `outputs.result` property of the [get batch synthesis](#) response. Replace `YourSpeechKey` with your Speech resource key.

Azure CLI

```
curl -v -X GET "YourOutputsResultUrl" -H "Ocp-Apim-Subscription-Key: YourSpeechKey" > results.zip
```

The results are in a ZIP file that contains the audio (such as `0001.wav`), summary, and debug details. The numbered prefix of each filename (shown below as `[nnnn]`) is in the same order as the text inputs used when you created the batch synthesis.

ⓘ Note

The `[nnnn].debug.json` file contains the synthesis result ID and other information that might help with troubleshooting. The properties that it contains might change, so you shouldn't take any dependencies on the JSON format.

The summary file contains the synthesis results for each text input. Here's an example `summary.json` file:

JSON

```
{
  "jobID": "41b83de2-380d-45dc-91af-722b68cf8e",
  "status": "Succeeded",
  "results": [
    {
      "texts": [
        "<speak version='1.0' xml:lang='en-US'>\n\t\t\t\t\t<voice
xml:lang='en-US' xml:gender='Female' name='en-US-
JennyNeural'>\n\t\t\t\t\t\tThe rainbow has seven
colors.\n\t\t\t\t\t</voice>\n\t\t\t\t</speak>"
      ],
      "status": "Succeeded",
      "billingDetails": {
        "CustomNeural": "0",
        "Neural": "33"
      },
      "audioFileName": "0001.wav",
      "properties": {
        "audioSize": "100000",
        "duration": "PT3.1S",
        "durationInTicks": "31250000"
      }
    }
  ]
}
```

If sentence boundary data was requested ("sentenceBoundaryEnabled": true), then a corresponding [nnnn].sentence.json file will be included in the results. Likewise, if word boundary data was requested ("wordBoundaryEnabled": true), then a corresponding [nnnn].word.json file will be included in the results.

Here's an example word data file with both audio offset and duration in milliseconds:

```
JSON
[{"Text": "the", "AudioOffset": 38, "Duration": 153}, {"Text": "rainbow", "AudioOffset": 201, "Duration": 326}, {"Text": "has", "AudioOffset": 567, "Duration": 96}, {"Text": "seven", "AudioOffset": 673, "Duration": 96}, {"Text": "colors", "AudioOffset": 778, "Duration": 451}]
```

Batch synthesis properties

Batch synthesis properties are described in the following table.

Property	Description
createdDateTime	The date and time when the batch synthesis job was created. This property is read-only.

Property	Description
<code>customProperties</code>	<p>A custom set of optional batch synthesis configuration settings.</p> <p>This property is stored for your convenience to associate the synthesis jobs that you created with the synthesis jobs that you get or list. This property is stored, but isn't used by the Speech service.</p> <p>You can specify up to 10 custom properties as key and value pairs. The maximum allowed key length is 64 characters, and the maximum allowed value length is 256 characters.</p>
<code>customVoices</code>	<p>The map of a custom voice name and its deployment ID.</p> <p>For example: <code>"customVoices": {"your-custom-voice-name": "502ac834-6537-4bc3-9fd6-140114daa66d"}</code></p> <p>You can use the voice name in your <code>synthesisConfig.voice</code> (when the <code>textType</code> is set to <code>"PlainText"</code>) or within the SSML text of <code>inputs</code> (when the <code>textType</code> is set to <code>"SSML"</code>).</p> <p>This property is required to use a custom voice. If you try to use a custom voice that isn't defined here, the service returns an error.</p>
<code>description</code>	<p>The description of the batch synthesis.</p> <p>This property is optional.</p>
<code>displayName</code>	<p>The name of the batch synthesis. Choose a name that you can refer to later. The display name doesn't have to be unique.</p> <p>This property is required.</p>
<code>id</code>	<p>The batch synthesis job ID.</p> <p>This property is read-only.</p>

Property	Description
<code>inputs</code>	<p>The plain text or SSML to be synthesized.</p> <p>When the <code>textType</code> is set to "PlainText", provide plain text as shown here: <code>"inputs": [{"text": "The rainbow has seven colors."}]</code>. When the <code>textType</code> is set to "SSML", provide text in the Speech Synthesis Markup Language (SSML) as shown here: <code>"inputs": [{"text": "<speak version='1.0'><voice xml:lang='en-US'>The rainbow has seven colors.</voice></speak>"}]</code>.</p> <p>Include up to 1,000 text objects if you want multiple audio output files. Here's example input text that should be synthesized to two audio output files: <code>"inputs": [{"text": "synthesize this to a file"}, {"text": "synthesize this to another file"}]</code>. However, if the <code>properties.concatenateResult</code> property is set to <code>true</code>, then each synthesized result will be written to the same audio output file.</p>
	<p>You don't need separate text inputs for new paragraphs. Within any of the (up to 1,000) text inputs, you can specify new paragraphs using the "\r\n" (newline) string. Here's example input text with two paragraphs that should be synthesized to the same audio output file: <code>"inputs": [{"text": "synthesize this to a file\r\nsynthesize this to another paragraph in the same file"}]</code></p>
	<p>There are no paragraph limits, but keep in mind that the maximum JSON payload size (including all text inputs and other properties) that will be accepted is 500 kilobytes.</p>
	<p>This property is required when you create a new batch synthesis job. This property isn't included in the response when you get the synthesis job.</p>
<code>lastActionDateTime</code>	<p>The most recent date and time when the <code>status</code> property value changed.</p> <p>This property is read-only.</p>
<code>outputs.result</code>	<p>The location of the batch synthesis result files with audio output and logs.</p> <p>This property is read-only.</p>

Property	Description
<code>properties</code>	A defined set of optional batch synthesis configuration settings.
<code>properties.audioSize</code>	The audio output size in bytes. This property is read-only.
<code>properties.billingDetails</code>	The number of words that were processed and billed by <code>customNeural</code> versus <code>neural</code> (prebuilt) voices. This property is read-only.
<code>properties.concatenateResult</code>	Determines whether to concatenate the result. This optional <code>bool</code> value ("true" or "false") is "false" by default.
<code>properties.decompressOutputFiles</code>	Determines whether to unzip the synthesis result files in the destination container. This property can only be set when the <code>destinationContainerUrl</code> property is set or BYOS (Bring Your Own Storage) is configured for the Speech resource. This optional <code>bool</code> value ("true" or "false") is "false" by default.
<code>properties.destinationContainerUrl</code>	The batch synthesis results can be stored in a writable Azure container. If you don't specify a container URI with shared access signatures (SAS) token, the Speech service stores the results in a container managed by Microsoft. SAS with stored access policies isn't supported. When the synthesis job is deleted, the result data is also deleted. This optional property isn't included in the response when you get the synthesis job.
<code>properties.duration</code>	The audio output duration. The value is an ISO 8601 encoded duration. This property is read-only.
<code>properties.durationInTicks</code>	The audio output duration in ticks. This property is read-only.
<code>properties.failedAudioCount</code>	The count of batch synthesis inputs to audio output failed. This property is read-only.

Property	Description
<code>properties.outputFormat</code>	<p>The audio output format.</p> <p>For information about the accepted values, see audio output formats. The default output format is <code>riff-24khz-16bit-mono-pcm</code>.</p>
<code>properties.sentenceBoundaryEnabled</code>	<p>Determines whether to generate sentence boundary data. This optional <code>bool</code> value ("true" or "false") is "false" by default.</p> <p>If sentence boundary data is requested, then a corresponding <code>[nnnn].sentence.json</code> file will be included in the results data ZIP file.</p>
<code>properties.succeededAudioCount</code>	<p>The count of batch synthesis inputs to audio output succeeded.</p> <p>This property is read-only.</p>
<code>properties.timeToLive</code>	<p>A duration after the synthesis job is created, when the synthesis results will be automatically deleted. The value is an ISO 8601 encoded duration. For example, specify <code>PT12H</code> for 12 hours. This optional setting is <code>P31D</code> (31 days) by default. The maximum time to live is 31 days. The date and time of automatic deletion (for synthesis jobs with a status of "Succeeded" or "Failed") is equal to the <code>lastActionDateTime</code> + <code>timeToLive</code> properties.</p> <p>Otherwise, you can call the Delete synthesis method to remove the job sooner.</p>
<code>properties.wordBoundaryEnabled</code>	<p>Determines whether to generate word boundary data. This optional <code>bool</code> value ("true" or "false") is "false" by default.</p> <p>If word boundary data is requested, then a corresponding <code>[nnnn].word.json</code> file will be included in the results data ZIP file.</p>
<code>status</code>	<p>The batch synthesis processing status.</p> <p>The status should progress from "NotStarted" to "Running", and finally to either "Succeeded" or "Failed".</p> <p>This property is read-only.</p>

Property	Description
<code>synthesisConfig</code>	<p>The configuration settings to use for batch synthesis of plain text.</p> <p>This property is only applicable when <code>textType</code> is set to <code>"PlainText"</code>.</p>
<code>synthesisConfig.pitch</code>	<p>The pitch of the audio output.</p> <p>For information about the accepted values, see the adjust prosody table in the Speech Synthesis Markup Language (SSML) documentation. Invalid values are ignored.</p> <p>This optional property is only applicable when <code>textType</code> is set to <code>"PlainText"</code>.</p>
<code>synthesisConfig.rate</code>	<p>The rate of the audio output.</p> <p>For information about the accepted values, see the adjust prosody table in the Speech Synthesis Markup Language (SSML) documentation. Invalid values are ignored.</p> <p>This optional property is only applicable when <code>textType</code> is set to <code>"PlainText"</code>.</p>
<code>synthesisConfig.style</code>	<p>For some voices, you can adjust the speaking style to express different emotions like cheerfulness, empathy, and calm. You can optimize the voice for different scenarios like customer service, newscast, and voice assistant.</p> <p>For information about the available styles per voice, see voice styles and roles.</p> <p>This optional property is only applicable when <code>textType</code> is set to <code>"PlainText"</code>.</p>
<code>synthesisConfig.voice</code>	<p>The voice that speaks the audio output.</p> <p>For information about the available prebuilt neural voices, see language and voice support. To use a custom voice, you must specify a valid custom voice and deployment ID mapping in the <code>customVoices</code> property.</p> <p>This property is required when <code>textType</code> is set to <code>"PlainText"</code>.</p>

Property	Description
<code>synthesisConfig.volume</code>	<p>The volume of the audio output.</p> <p>For information about the accepted values, see the adjust prosody table in the Speech Synthesis Markup Language (SSML) documentation. Invalid values are ignored.</p> <p>This optional property is only applicable when <code>textType</code> is set to <code>"PlainText"</code>.</p>
<code>textType</code>	<p>Indicates whether the <code>inputs</code> text property should be plain text or SSML. The possible case-insensitive values are "PlainText" and "SSML". When the <code>textType</code> is set to "PlainText", you must also set the <code>synthesisConfig</code> voice property.</p> <p>This property is required.</p>

HTTP status codes

The section details the HTTP response codes and messages from the batch synthesis API.

HTTP 200 OK

HTTP 200 OK indicates that the request was successful.

HTTP 201 Created

HTTP 201 Created indicates that the create batch synthesis request (via HTTP POST) was successful.

HTTP 204 error

An HTTP 204 error indicates that the request was successful, but the resource doesn't exist. For example:

- You tried to get or delete a synthesis job that doesn't exist.
- You successfully deleted a synthesis job.

HTTP 400 error

Here are examples that can result in the 400 error:

- The `outputFormat` is unsupported or invalid. Provide a valid format value, or leave `outputFormat` empty to use the default setting.
- The number of requested text inputs exceeded the limit of 1,000.
- The `top` query parameter exceeded the limit of 100.
- You tried to use an invalid deployment ID or a custom voice that isn't successfully deployed. Make sure the Speech resource has access to the custom voice, and the custom voice is successfully deployed. You must also ensure that the mapping of `{"your-custom-voice-name": "your-deployment-ID"}` is correct in your batch synthesis request.
- You tried to delete a batch synthesis job that hasn't started or hasn't completed running. You can only delete batch synthesis jobs that have a status of "Succeeded" or "Failed".
- You tried to use a *F0* Speech resource, but the region only supports the *Standard* Speech resource pricing tier.
- You tried to create a new batch synthesis job that would exceed the limit of 200 active jobs. Each Speech resource can have up to 200 batch synthesis jobs that don't have a status of "Succeeded" or "Failed".

HTTP 404 error

The specified entity can't be found. Make sure the synthesis ID is correct.

HTTP 429 error

There are too many recent requests. Each client application can submit up to 50 requests per 5 seconds for each Speech resource. Reduce the number of requests per second.

You can check the rate limit and quota remaining via the HTTP headers as shown in the following example:

HTTP

```
X-RateLimit-Limit: 50
X-RateLimit-Remaining: 49
X-RateLimit-Reset: 2022-11-11T01:49:43Z
```

HTTP 500 error

HTTP 500 Internal Server Error indicates that the request failed. The response body contains the error message.

HTTP error example

Here's an example request that results in an HTTP 400 error, because the `top` query parameter is set to a value greater than 100.

Console

```
curl -v -X GET  
"https://YourSpeechRegion.customvoice.api.speech.microsoft.com/api/texttospe  
ech/3.1-preview1/batchsynthesis?skip=0&top=200" -H "Ocp-Apim-Subscription-  
Key: YourSpeechKey"
```

In this case, the response headers will include `HTTP/1.1 400 Bad Request`.

The response body will resemble the following JSON example:

JSON

```
{  
  "code": "InvalidRequest",  
  "message": "The top parameter should not be greater than 100.",  
  "innerError": {  
    "code": "InvalidParameter",  
    "message": "The top parameter should not be greater than 100."  
  }  
}
```

Next steps

- Speech Synthesis Markup Language (SSML)
- Text-to-speech quickstart
- Migrate to batch synthesis

Additional resources

 Documentation

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[Speech-to-text REST API - Speech service - Azure Cognitive Services](#)

Get reference documentation for Speech-to-text REST API.

[How to use pronunciation assessment in Speech Studio - Azure Cognitive Services](#)

The pronunciation assessment tool in Speech Studio gives you feedback on the accuracy and fluency of your speech, no coding required.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[azure.cognitiveservices.speech package](#)

Microsoft Speech SDK for Python

[azure.cognitiveservices.speech.SpeechConfig class](#)

Class that defines configurations for speech / intent recognition and speech synthesis. The configuration can be initialized in different ways: from subscription: pass a subscription key and a region from endpoint: pass an endpoint. Subscription key or authorization token are optional. from...

[About the Speech SDK - Speech service - Azure Cognitive Services](#)

The Speech software development kit (SDK) exposes many of the Speech service capabilities, making it easier to develop speech-enabled applications.

[Show 5 more](#)

Speech Synthesis Markup Language (SSML) overview

Article • 01/13/2023 • 2 minutes to read

Speech Synthesis Markup Language (SSML) is an XML-based markup language that can be used to fine-tune the text-to-speech output attributes such as pitch, pronunciation, speaking rate, volume, and more. You have more control and flexibility compared to plain text input.

💡 Tip

You can hear voices in different styles and pitches reading example text via the [Voice Gallery](#).

Scenarios

You can use SSML to:

- [Define the input text structure](#) that determines the structure, content, and other characteristics of the text-to-speech output. For example, you can use SSML to define a paragraph, a sentence, a break or a pause, or silence. You can wrap text with event tags such as bookmark or viseme that can be processed later by your application.
- [Choose the voice](#), language, name, style, and role. You can use multiple voices in a single SSML document. Adjust the emphasis, speaking rate, pitch, and volume. You can also use SSML to insert pre-recorded audio, such as a sound effect or a musical note.
- [Control pronunciation](#) of the output audio. For example, you can use SSML with phonemes and a custom lexicon to improve pronunciation. You can also use SSML to define how a word or mathematical expression is pronounced.

Use SSML

ⓘ Important

You're billed for each character that's converted to speech, including punctuation. Although 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. For more information, see [text-to-speech pricing notes](#).

You can use SSML in the following ways:

- [Audio Content Creation](#) tool: Author plain text and SSML in Speech Studio: You can listen to the output audio and adjust the SSML to improve speech synthesis. For more information, see [Speech synthesis with the Audio Content Creation tool](#).
- [Batch synthesis API](#): Provide SSML via the `inputs` property.
- [Speech CLI](#): Provide SSML via the `spx synthesize --ssml SSML` command line argument.
- [Speech SDK](#): Provide SSML via the "speak" SSML method.

Next steps

- [SSML document structure and events](#)
- [Voice and sound with SSML](#)
- [Pronunciation with SSML](#)
- [Language support: Voices, locales, languages](#)

Additional resources

Documentation

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[Speech phonetic alphabets - Speech service - Azure Cognitive Services](#)

This article presents Speech service phonetic alphabet and International Phonetic Alphabet (IPA) examples.

[Text-to-speech API reference \(REST\) - Speech service - Azure Cognitive Services](#)

Learn how to use the REST API to convert text into synthesized speech.

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Custom Neural Voice overview - Speech service - Azure Cognitive Services](#)

Custom Neural Voice is a text-to-speech feature that allows you to create a one-of-a-kind, customized, synthetic voice for your applications. You provide your own audio data as a sample.

[Speech Synthesis Markup Language \(SSML\) document structure and events - Speech service - Azure Cognitive Services](#)

Learn about the Speech Synthesis Markup Language (SSML) document structure.

[How to lower speech synthesis latency using Speech SDK - Azure Cognitive Services](#)

How to lower speech synthesis latency using Speech SDK, including streaming, pre-connection, and so on.

[Show 5 more](#)

SSML document structure and events

Article • 02/02/2023 • 9 minutes to read

The Speech Synthesis Markup Language (SSML) with input text determines the structure, content, and other characteristics of the text-to-speech output. For example, you can use SSML to define a paragraph, a sentence, a break or a pause, or silence. You can wrap text with event tags such as bookmark or viseme that can be processed later by your application.

Refer to the sections below for details about how to structure elements in the SSML document.

Document structure

The Speech service implementation of SSML is based on the World Wide Web Consortium's [Speech Synthesis Markup Language Version 1.0](#). The elements supported by the Speech can differ from the W3C standard.

Each SSML document is created with SSML elements or tags. These elements are used to adjust the voice, style, pitch, prosody, volume, and more.

Here's a subset of the basic structure and syntax of an SSML document:

```
XML

<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="string">
    <mstts:backgroundaudio src="string" volume="string" fadein="string"
    fadeout="string"/>
    <voice name="string">
        <audio src="string"/></audio>
        <bookmark mark="string"/>
        <break strength="string" time="string" />
        <emphasis level="value"></emphasis>
        <lang xml:lang="string"></lang>
        <lexicon uri="string"/>
        <math xmlns="http://www.w3.org/1998/Math/MathML"></math>
        <mstts:audioduration value="string"/>
        <mstts:express-as style="string" styledegree="value" role="string">
    </mstts:express-as>
        <mstts:silence type="string" value="string"/>
        <mstts:viseme type="string"/>
        <p></p>
        <phoneme alphabet="string" ph="string"></phoneme>
        <prosody pitch="value" contour="value" range="value" rate="value"
        volume="value"></prosody>
```

```
<s></s>
<say-as interpret-as="string" format="string" detail="string"></say-
as>
<sub alias="string"></sub>
</voice>
</speak>
```

Some examples of contents that are allowed in each element are described in the following list:

- `audio`: The body of the `audio` element can contain plain text or SSML markup that's spoken if the audio file is unavailable or unplayable. The `audio` element can also contain text and the following elements: `audio`, `break`, `p`, `s`, `phoneme`, `prosody`, `say-as`, and `sub`.
- `bookmark`: This element can't contain text or any other elements.
- `break`: This element can't contain text or any other elements.
- `emphasis`: This element can contain text and the following elements: `audio`, `break`, `emphasis`, `lang`, `phoneme`, `prosody`, `say-as`, and `sub`.
- `lang`: This element can contain all other elements except `mstts:backgroundaudio`, `voice`, and `speak`.
- `lexicon`: This element can't contain text or any other elements.
- `math`: This element can only contain text and MathML elements.
- `mstts:audioduration`: This element can't contain text or any other elements.
- `mstts:backgroundaudio`: This element can't contain text or any other elements.
- `mstts:express-as`: This element can contain text and the following elements: `audio`, `break`, `emphasis`, `lang`, `phoneme`, `prosody`, `say-as`, and `sub`.
- `mstts:silence`: This element can't contain text or any other elements.
- `mstts:viseme`: This element can't contain text or any other elements.
- `p`: This element can contain text and the following elements: `audio`, `break`, `phoneme`, `prosody`, `say-as`, `sub`, `mstts:express-as`, and `s`.
- `phoneme`: This element can only contain text and no other elements.
- `prosody`: This element can contain text and the following elements: `audio`, `break`, `p`, `phoneme`, `prosody`, `say-as`, `sub`, and `s`.
- `s`: This element can contain text and the following elements: `audio`, `break`, `phoneme`, `prosody`, `say-as`, `mstts:express-as`, and `sub`.
- `say-as`: This element can only contain text and no other elements.
- `sub`: This element can only contain text and no other elements.
- `speak`: The root element of an SSML document. This element can contain the following elements: `mstts:backgroundaudio` and `voice`.

- `voice`: This element can contain all other elements except `mstts:backgroundaudio` and `speak`.

The Speech service automatically handles punctuation as appropriate, such as pausing after a period, or using the correct intonation when a sentence ends with a question mark.

Special characters such as quotation marks, apostrophes, and brackets, must be escaped. For more information, see [Extensible Markup Language \(XML\) 1.0: Appendix D ↴](#).

Attribute values must be enclosed by double quotation marks. For example, `<prosody volume="90">` is a well-formed, valid element, but `<prosody volume=90>` won't be recognized.

Speak root element

The `speak` element is the root element that's required for all SSML documents. The `speak` element contains information such as version, language, and the markup vocabulary definition.

Here's the syntax for the `speak` element:

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="string"></speak>
```

Attribute	Description	Required or optional
<code>version</code>	Indicates the version of the SSML specification used to interpret the document markup. The current version is "1.0".	Required
<code>xml:lang</code>	The language of the root document. The value can contain a language code such as <code>en</code> (English), or a locale such as <code>en-us</code> (English - United States).	Required
<code>xmlns</code>	The URI to the document that defines the markup vocabulary (the element types and attribute names) of the SSML document. The current URI is "http://www.w3.org/2001/10/synthesis".	Required

The `speak` element must contain at least one [voice element](#).

speak examples

The supported values for attributes of the `speak` element were described previously.

Single voice example

This example uses the `en-US-JennyNeural` voice. For more examples, see [voice examples](#).

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    This is the text that is spoken.
  </voice>
</speak>
```

Add or prevent a break

Use the `break` element to override the default behavior of breaks or pauses between words. You can use it to add or prevent pauses that are otherwise automatically inserted by the Speech service.

Usage of the `break` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>strength</code>	<p>The relative duration of a pause by using one of the following values:</p> <ul style="list-style-type: none">• none• x-weak• weak• medium (default)• strong• x-strong	Optional
<code>time</code>	<p>Set <code>strength</code> to <code>none</code> to prevent automatic insertion of a prosodic break.</p> <p>The absolute duration of a pause in seconds (such as <code>2s</code>) or milliseconds (such as <code>500ms</code>). Valid values range from 0 to 5000 milliseconds. If you set a value greater than the supported maximum, the service will use <code>5000ms</code>. If the <code>time</code> attribute is set, the <code>strength</code> attribute is ignored.</p>	Optional

Here are more details about the `strength` attribute.

Strength	Relative duration
None, or if no value provided	0 ms
X-weak	250 ms
Weak	500 ms
Medium	750 ms
Strong	1,000 ms
X-strong	1,250 ms

Break examples

The supported values for attributes of the `break` element were [described previously](#).

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    Welcome <break /> to text-to-speech.
    Welcome <break strength="medium" /> to text-to-speech.
    Welcome <break time="250ms" /> to text-to-speech.
  </voice>
</speak>
```

Add silence

Use the `mstts:silence` element to insert pauses before or after text, or between two adjacent sentences.

One of the differences between `mstts:silence` and `break` is that a `break` element can be inserted anywhere in the text. Silence only works at the beginning or end of input text or at the boundary of two adjacent sentences.

The silence setting is applied to all input text within its enclosing `voice` element. To reset or change the silence setting again, you must use a new `voice` element with either the same voice or a different voice.

Usage of the `mstts:silence` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>type</code>	<p>Specifies where and how to add silence. The following silence types are supported:</p> <ul style="list-style-type: none"> • <code>Leading</code> – Additional silence at the beginning of the text. The value that you set is added to the natural silence before the start of text. • <code>Leading-exact</code> – Silence at the beginning of the text. The value is an absolute silence length. • <code>Tailing</code> – Additional silence at the end of text. The value that you set is added to the natural silence after the last word. • <code>Tailing-exact</code> – Silence at the end of the text. The value is an absolute silence length. • <code>Sentenceboundary</code> – Additional silence between adjacent sentences. The actual silence length for this type includes the natural silence after the last word in the previous sentence, the value you set for this type, and the natural silence before the starting word in the next sentence. • <code>Sentenceboundary-exact</code> – Silence between adjacent sentences. The value is an absolute silence length. • <code>Comma-exact</code> – Silence at the comma in half-width or full-width format. The value is an absolute silence length. • <code>Semicolon-exact</code> – Silence at the semicolon in half-width or full-width format. The value is an absolute silence length. • <code>Enumerationcomma-exact</code> – Silence at the enumeration comma in full-width format. The value is an absolute silence length. <p>An absolute silence type (with the <code>-exact</code> suffix) replaces any otherwise natural leading or trailing silence. Absolute silence types take precedence over the corresponding non-absolute type. For example, if you set both <code>Leading</code> and <code>Leading-exact</code> types, the <code>Leading-exact</code> type will take effect.</p>	Required
<code>Value</code>	The duration of a pause in seconds (such as <code>2s</code>) or milliseconds (such as <code>500ms</code>). Valid values range from 0 to 5000 milliseconds. If you set a value greater than the supported maximum, the service will use <code>5000ms</code> .	Required

mstts silence examples

The supported values for attributes of the `mstts:silence` element were [described previously](#).

In this example, `mstts:silence` is used to add 200 ms of silence between two sentences.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <mstts:silence type="Sentenceboundary" value="200ms"/>
    If we're home schooling, the best we can do is roll with what each day
    brings and try to have fun along the way.
    A good place to start is by trying out the slew of educational apps that are
    helping children stay happy and smash their schooling at the same time.
  </voice>
</speak>
```

In this example, `mstts:silence` is used to add 50 ms of silence at the comma, 100 ms of silence at the semicolon, and 150 ms of silence at the enumeration comma.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="zh-CN">
  <voice name="zh-CN-YunxiNeural">
    <mstts:silence type="comma-exact" value="50ms"/><mstts:silence
    type="semicolon-exact" value="100ms"/><mstts:silence type="enumerationcomma-
    exact" value="150ms"/>你好呀，云希、晓晓；你好呀。
  </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 Speech service automatically determines the structure of the SSML document.

Paragraph and sentence examples

The following example defines two paragraphs that each contain sentences. In the second paragraph, the Speech service automatically determines the sentence structure, since they aren't defined in the SSML document.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <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>

```

Bookmark element

You can use the `bookmark` element in SSML to reference a specific location in the text or tag sequence. Then you'll use the Speech SDK and subscribe to the `BookmarkReached` event to get the offset of each marker in the audio stream. The `bookmark` element won't be spoken. For more information, see [Subscribe to synthesizer events](#).

Usage of the `bookmark` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>mark</code>	The reference text of the <code>bookmark</code> element.	Required

Bookmark examples

The supported values for attributes of the `bookmark` element were [described previously](#).

As an example, you might want to know the time offset of each flower word in the following snippet:

XML

```

<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-AriaNeural">
        We are selling <bookmark mark='flower_1' />roses and <bookmark
        mark='flower_2' />daisies.
    </voice>
</speak>

```

Viseme element

A viseme is the visual description of a phoneme in spoken language. It defines the position of the face and mouth while a person is speaking. You can use the

`mstts:viseme` element in SSML to request viseme output. For more information, see [Get facial position with viseme](#).

The viseme setting is applied to all input text within its enclosing `voice` element. To reset or change the viseme setting again, you must use a new `voice` element with either the same voice or a different voice.

Usage of the `viseme` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>type</code>	<p>The type of viseme output.</p> <ul style="list-style-type: none">• <code>redlips_front</code> – lip-sync with viseme ID and audio offset output• <code>FacialExpression</code> – blend shapes output	Required

ⓘ Note

Currently, `redlips_front` only supports neural voices in `en-US` locale, and `FacialExpression` supports neural voices in `en-US` and `zh-CN` locales.

Viseme examples

The supported values for attributes of the `viseme` element were [described previously](#).

This SSML snippet illustrates how to request blend shapes with your synthesized speech.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <mstts:viseme type="FacialExpression"/>
    Rainbow has seven colors: Red, orange, yellow, green, blue, indigo, and
    violet.
  </voice>
</speak>
```

Next steps

- [SSML overview](#)

- Voice and sound with SSML
 - Language support: Voices, locales, languages
-

Additional resources

Documentation

[Voice and sound with Speech Synthesis Markup Language \(SSML\) - Speech service - Azure Cognitive Services](#)

Learn about Speech Synthesis Markup Language (SSML) elements to determine what your output audio will sound like.

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Speech phonetic alphabets - Speech service - Azure Cognitive Services](#)

This article presents Speech service phonetic alphabet and International Phonetic Alphabet (IPA) examples.

[Install the Speech SDK - Azure Cognitive Services](#)

In this quickstart, you'll learn how to install the Speech SDK for your preferred programming language.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Troubleshoot the Speech SDK - Speech service - Azure Cognitive Services](#)

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

[microsoft-cognitiveservices-speech-sdk package](#)

[Show 5 more](#)

Voice and sound with SSML

Article • 02/02/2023 • 19 minutes to read

Use Speech Synthesis Markup Language (SSML) to specify the text-to-speech voice, language, name, style, and role. You can use multiple voices in a single SSML document. Adjust the emphasis, speaking rate, pitch, and volume. You can also use SSML to insert pre-recorded audio, such as a sound effect or a musical note.

Refer to the sections below for details about how to use SSML elements to specify voice and sound. For more information about SSML syntax, see [SSML document structure and events](#).

Voice element

At least one `voice` element must be specified within each SSML `speak` element. This element determines the voice that's used for text-to-speech.

You can include multiple `voice` elements in a single SSML document. Each `voice` element can specify a different voice. You can also use the same voice multiple times with different settings, such as when you [change the silence duration](#) between sentences.

Usage of the `voice` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>name</code>	The voice used for text-to-speech output. For a complete list of supported prebuilt voices, see Language support .	Required

Voice examples

The supported values for attributes of the `voice` element were [described previously](#).

Single voice example

This example uses the `en-US-JennyNeural` voice.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xml:lang="en-US">
```

```
<voice name="en-US-JennyNeural">
    This is the text that is spoken.
</voice>
</speak>
```

Multiple voices example

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.

This example alternates between the `en-US-JennyNeural` and `en-US-ChristopherNeural` voices.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        Good morning!
    </voice>
    <voice name="en-US-ChristopherNeural">
        Good morning to you too Jenny!
    </voice>
</speak>
```

Custom neural voice example

To use your [custom neural voice](#), specify the model name as the voice name in SSML.

This example uses a custom voice named "my-custom-voice".

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="my-custom-voice">
        This is the text that is spoken.
    </voice>
</speak>
```

Speaking styles and roles

By default, neural voices have a neutral speaking style. You can adjust the speaking style, style degree, and role at the sentence level.

ⓘ Note

Styles, style degree, and roles are supported for a subset of neural voices as described in the [voice styles and roles](#) documentation. To determine what styles and roles are supported for each voice, you can also use the [list voices API](#) and the [Audio Content Creation](#) web application.

Usage of the `mstts:express-as` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>style</code>	The voice-specific speaking style. You can express emotions like cheerfulness, empathy, and calm. You can also optimize the voice for different scenarios like customer service, newscast, and voice assistant. If the style value is missing or invalid, the entire <code>mstts:express-as</code> element is ignored and the service uses the default neutral speech. For custom neural voice styles, see the custom neural voice style example .	Required
<code>styledegree</code>	The intensity of the speaking style. You can specify a stronger or softer style to make the speech more expressive or subdued. The range of accepted values are: 0.01 to 2 inclusive. The default value is 1, which means the predefined style intensity. The minimum unit is 0.01, which results in a slight tendency for the target style. A value of 2 results in a doubling of the default style intensity. If the style degree is missing or isn't supported for your voice, this attribute is ignored.	Optional
<code>role</code>	The speaking role-play. The voice can imitate a different age and gender, but the voice name isn't changed. For example, a male voice can raise the pitch and change the intonation to imitate a female voice, but the voice name won't be changed. If the role is missing or isn't supported for your voice, this attribute is ignored.	Optional

The following table has descriptions of each supported `style` attribute.

Style	Description
<code>style="advertisement_upbeat"</code>	Expresses an excited and high-energy tone for promoting a product or service.

Style	Description
<code>style="affectionate"</code>	Expresses a warm and affectionate tone, with higher pitch and vocal energy. The speaker is in a state of attracting the attention of the listener. The personality of the speaker is often endearing in nature.
<code>style="angry"</code>	Expresses an angry and annoyed tone.
<code>style="assistant"</code>	Expresses a warm and relaxed tone for digital assistants.
<code>style="calm"</code>	Expresses a cool, collected, and composed attitude when speaking. Tone, pitch, and prosody are more uniform compared to other types of speech.
<code>style="chat"</code>	Expresses a casual and relaxed tone.
<code>style="cheerful"</code>	Expresses a positive and happy tone.
<code>style="customerservice"</code>	Expresses a friendly and helpful tone for customer support.
<code>style="depressed"</code>	Expresses a melancholic and despondent tone with lower pitch and energy.
<code>style="disgruntled"</code>	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
<code>style="documentary-narration"</code>	Narrates documentaries in a relaxed, interested, and informative style suitable for dubbing documentaries, expert commentary, and similar content.
<code>style="embarrassed"</code>	Expresses an uncertain and hesitant tone when the speaker is feeling uncomfortable.
<code>style="empathetic"</code>	Expresses a sense of caring and understanding.
<code>style="envious"</code>	Expresses a tone of admiration when you desire something that someone else has.
<code>style="excited"</code>	Expresses an upbeat and hopeful tone. It sounds like something great is happening and the speaker is really happy about that.
<code>style="fearful"</code>	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tension and unease.
<code>style="friendly"</code>	Expresses a pleasant, inviting, and warm tone. It sounds sincere and caring.
<code>style="gentle"</code>	Expresses a mild, polite, and pleasant tone, with lower pitch and vocal energy.

Style	Description
<code>style="hopeful"</code>	Expresses a warm and yearning tone. It sounds like something good will happen to the speaker.
<code>style="lyrical"</code>	Expresses emotions in a melodic and sentimental way.
<code>style="narration-professional"</code>	Expresses a professional, objective tone for content reading.
<code>style="narration-relaxed"</code>	Express a soothing and melodious tone for content reading.
<code>style="newscast"</code>	Expresses a formal and professional tone for narrating news.
<code>style="newscast-casual"</code>	Expresses a versatile and casual tone for general news delivery.
<code>style="newscast-formal"</code>	Expresses a formal, confident, and authoritative tone for news delivery.
<code>style="poetry-reading"</code>	Expresses an emotional and rhythmic tone while reading a poem.
<code>style="sad"</code>	Expresses a sorrowful tone.
<code>style="serious"</code>	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.
<code>style="shouting"</code>	Speaks like from a far distant or outside and to make self be clearly heard
<code>style="sports_commentary"</code>	Expresses a relaxed and interesting tone for broadcasting a sports event.
<code>style="sports_commentary_excited"</code>	Expresses an intensive and energetic tone for broadcasting exciting moments in a sports event.
<code>style="whispering"</code>	Speaks very softly and make a quiet and gentle sound
<code>style="terrified"</code>	Expresses a very scared tone, with faster pace and a shakier voice. It sounds like the speaker is in an unsteady and frantic status.
<code>style="unfriendly"</code>	Expresses a cold and indifferent tone.

The following table has descriptions of each supported `role` attribute.

Role	Description
-------------	--------------------

Role	Description
role="Girl"	The voice imitates a girl.
role="Boy"	The voice imitates a boy.
role="YoungAdultFemale"	The voice imitates a young adult female.
role="YoungAdultMale"	The voice imitates a young adult male.
role="OlderAdultFemale"	The voice imitates an older adult female.
role="OlderAdultMale"	The voice imitates an older adult male.
role="SeniorFemale"	The voice imitates a senior female.
role="SeniorMale"	The voice imitates a senior male.

mstts express-as examples

The supported values for attributes of the `mstts:express-as` element were described previously.

Style and degree example

You use the `mstts:express-as` element to express emotions like cheerfulness, empathy, and calm. You can also optimize the voice for different scenarios like customer service, newscast, and voice assistant.

The following SSML example uses the `<mstts:express-as>` element with a sad style degree of "2".

XML

```

<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="zh-CN">
  <voice name="zh-CN-XiaomoNeural">
    <mstts:express-as style="sad" styledegree="2">
      快走吧，路上一定要注意安全，早去早回。
    </mstts:express-as>
  </voice>
</speak>

```

Role example

Apart from adjusting the speaking styles and style degree, you can also adjust the `role` parameter so that the voice imitates a different age and gender. For example, a male voice can raise the pitch and change the intonation to imitate a female voice, but the voice name won't be changed.

This SSML snippet illustrates how the `role` attribute is used to change the role-play for `zh-CN-XiaomoNeural`.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="zh-CN">
  <voice name="zh-CN-XiaomoNeural">
    女儿看见父亲走了进来，问道：
    <mstts:express-as role="YoungAdultFemale" style="calm">
      “您来的挺快的，怎么过来的？”
    </mstts:express-as>
    父亲放下手提包，说：
    <mstts:express-as role="OlderAdultMale" style="calm">
      “刚打车过来的，路上还挺顺畅。”
    </mstts:express-as>
  </voice>
</speak>
```

Custom neural voice style example

Your custom neural voice can be trained to speak with some preset styles such as cheerful, sad, and whispering. You can also [train a custom neural voice](#) to speak in a custom style as determined by your training data. To use your custom neural voice style in SSML, specify the style name that you previously entered in Speech Studio.

This example uses a custom voice named "my-custom-voice". The custom voice speaks with the "cheerful" preset style, and then with a custom style named "my-custom-style".

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="my-custom-voice">
    <mstts:express-as style="cheerful">
      That'd be just amazing!
    </mstts:express-as>
    <mstts:express-as style="my-custom-style">
      What's next?
    </mstts:express-as>
  </voice>
</speak>
```

Adjust speaking languages

By default, all neural voices are fluent in their own language and English without using the `<lang xml:lang>` element. For example, if the input text in English is "I'm excited to try text to speech" and you use the `es-ES-ElviraNeural` voice, the text is spoken in English with a Spanish accent. With most neural voices, setting a specific speaking language with `<lang xml:lang>` element at the sentence or word level is currently not supported.

You can adjust the speaking language for the `en-US-JennyMultilingualNeural` neural voice at the sentence level and word level by using the `<lang xml:lang>` element. The `en-US-JennyMultilingualNeural` neural voice is multilingual in 14 languages (For example: English, Spanish, and Chinese). The supported languages are provided in a table following the `<lang>` syntax and attribute definitions.

Usage of the `lang` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>xml:lang</code>	The language that you want the neural voice to speak.	Required to adjust the speaking language for the neural voice. If you're using <code>lang xml:lang</code> , the locale must be provided.

ⓘ Note

The `<lang xml:lang>` element is incompatible with the `prosody` and `break` elements. You can't adjust pause and prosody like pitch, contour, rate, or volume in this element.

Use this table to determine which speaking languages are supported for each neural voice. If the voice doesn't speak the language of the input text, the Speech service won't output synthesized audio.

Voice	Primary and default locale	Secondary locales
<code>en-US-JennyMultilingualNeural</code>	<code>en-US</code>	<code>de-DE, en-AU, en-CA, en-GB, es-ES, es-MX, fr-CA, fr-FR, it-IT, ja-JP, ko-KR, pt-BR, zh-CN</code>

Lang examples

The supported values for attributes of the `lang` element were described previously.

The primary language for `en-US-JennyMultilingualNeural` is `en-US`. You must specify `en-US` as the default language within the `speak` element, whether or not the language is adjusted elsewhere.

This SSML snippet shows how to use the `lang` element (and `xml:lang` attribute) to speak `de-DE` with the `en-US-JennyMultilingualNeural` neural voice.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-JennyMultilingualNeural">
    <lang xml:lang="de-DE">
      Wir freuen uns auf die Zusammenarbeit mit Ihnen!
    </lang>
  </voice>
</speak>
```

Within the `speak` element, you can specify multiple languages including `en-US` for text-to-speech output. For each adjusted language, the text must match the language and be wrapped in a `voice` element. This SSML snippet shows how to use `<lang xml:lang>` to change the speaking languages to `es-MX`, `en-US`, and `fr-FR`.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-JennyMultilingualNeural">
    <lang xml:lang="es-MX">
      ¡Esperamos trabajar con usted!
    </lang>
    <lang xml:lang="en-US">
      We look forward to working with you!
    </lang>
    <lang xml:lang="fr-FR">
      Nous avons hâte de travailler avec vous!
    </lang>
  </voice>
</speak>
```

Adjust prosody

The `prosody` element is used to specify changes to pitch, contour, range, rate, and volume for the text-to-speech output. The `prosody` element can 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. Text-to-speech limits or substitutes values that aren't supported. Examples of unsupported values are a pitch of 1 MHz or a volume of 120.

Usage of the `prosody` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>contour</code>	<p>Contour represents changes in pitch. These changes are represented 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><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 by using a relative value or an enumeration value for pitch (see <code>pitch</code>).</p>	Optional
<code>pitch</code>	<p>Indicates the baseline pitch for the text. Pitch changes can be applied at the sentence level. The pitch changes should be within 0.5 to 1.5 times the original audio. You can express the pitch as:</p> <ul style="list-style-type: none">• An absolute value: Expressed as a number followed by "Hz" (Hertz). For example, <code><prosody pitch="600Hz">some text</prosody></code>.• A relative value:<ul style="list-style-type: none">◦ As a relative number: Expressed as a number preceded by "+" or "-" and followed by "Hz" or "st" that specifies an amount to change the pitch. For example: <code><prosody pitch="+80Hz">some text</prosody></code> or <code><prosody pitch="-2st">some text</prosody></code>. The "st" indicates the change unit is semitone, which is half of a tone (a half step) on the standard diatonic scale.◦ As a percentage: Expressed as a number preceded by "+" (optionally) or "-" and followed by "%", indicating the relative change. For example: <code><prosody pitch="50%">some text</prosody></code> or <code><prosody pitch="-50%">some text</prosody></code>.• A constant value:<ul style="list-style-type: none">◦ x-low◦ low◦ medium◦ high◦ x-high◦ default	Optional

Attribute	Description	Required or optional
<code>range</code>	<p>A value that represents the range of pitch for the text. You can express <code>range</code> by using the same absolute values, relative values, or enumeration values used to describe <code>pitch</code>.</p>	Optional
<code>rate</code>	<p>Indicates the speaking rate of the text. Speaking rate can be applied at the word or sentence level. The rate changes should be within 0.5 to 2 times the original audio. You can express <code>rate</code> as:</p> <ul style="list-style-type: none"> • A relative value: <ul style="list-style-type: none"> ◦ As a relative number: 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 original rate. A value of <code>0.5</code> results in a halving of the original rate. A value of <code>2</code> results in twice the original rate. ◦ As a percentage: Expressed as a number preceded by "+" (optionally) or "-" and followed by "%", indicating the relative change. For example: <code><prosody rate="50%">some text</prosody></code> or <code><prosody rate="-50%">some text</prosody></code>. • 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

Attribute	Description	Required or optional
volume	<p>Indicates the volume level of the speaking voice. Volume changes can be applied at the sentence level. You can express the volume as:</p> <ul style="list-style-type: none"> • An absolute value: Expressed as a number in the range of 0.0 to 100.0, from <i>quietest</i> to <i>loudest</i>. An example is 75. The default is 100.0. • A relative value: <ul style="list-style-type: none"> ◦ As a relative number: Expressed as a number preceded by "+" or "-" that specifies an amount to change the volume. Examples are +10 or -5.5. ◦ As a percentage: Expressed as a number preceded by "+" (optionally) or "-" and followed by "%", indicating the relative change. For example: <code><prosody volume="50%">some text</prosody></code> Or <code><prosody volume="+3%">some text</prosody></code>. • A constant value: <ul style="list-style-type: none"> ◦ silent ◦ x-soft ◦ soft ◦ medium ◦ loud ◦ x-loud ◦ default 	Optional

Prosody examples

The supported values for attributes of the `prosody` element were [described previously](#).

Change speaking rate example

This SSML snippet illustrates how the `rate` attribute is used to change the speaking rate to 30% greater than the default rate.

XML

```

<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <prosody rate="+30.00%">
            Enjoy using text-to-speech.
        </prosody>
    </voice>
</speak>

```

Change volume example

This SSML snippet illustrates how the `volume` attribute is used to change the volume to 20% greater than the default volume.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <prosody volume="+20.00%">
            Enjoy using text-to-speech.
        </prosody>
    </voice>
</speak>
```

Change pitch example

This SSML snippet illustrates how the `pitch` attribute is used so that the voice speaks in a high pitch.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        Welcome to <prosody pitch="high">Enjoy using text-to-speech.
    </prosody>
    </voice>
</speak>
```

Change pitch contour example

This SSML snippet illustrates how the `contour` attribute is used to change the contour.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <prosody contour="(60%,-60%) (100%,+80%)">
            Were you the only person in the room?
        </prosody>
    </voice>
</speak>
```

Adjust emphasis

The optional `emphasis` element is used to add or remove word-level stress for the text. This element can only contain text and the following elements: `audio`, `break`, `emphasis`, `lang`, `phoneme`, `prosody`, `say-as`, `sub`, and `voice`.

ⓘ Note

The word-level emphasis tuning is only available for these neural voices: `en-US-GuyNeural`, `en-US-DavisNeural`, and `en-US-JaneNeural`.

Usage of the `emphasis` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>level</code>	Indicates the strength of emphasis to be applied: <ul style="list-style-type: none">• <code>reduced</code>• <code>none</code>• <code>moderate</code>• <code>strong</code>	Optional

When the `level` attribute isn't specified, the default level is `moderate`.
For details on each attribute, see [emphasis element ↗](#)

Emphasis examples

The supported values for attributes of the `emphasis` element were [described previously](#).

This SSML snippet demonstrates how the `emphasis` element is used to add moderate level emphasis for the word "meetings".

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-GuyNeural">
    I can help you join your <emphasis level="moderate">meetings</emphasis>
    fast.
  </voice>
</speak>
```

Add recorded audio

The `audio` element is optional. You can use it to insert prerecorded audio into an SSML document. The body of the `audio` element can contain plain text or SSML markup that's spoken if the audio file is unavailable or unplayable. The `audio` element can also 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 audio file must be valid `*.mp3`, `*.wav`, `*.opus`, `*.ogg`, `*.flac`, or `*.wma` files.
- The combined total time for all text and audio files in a single response can't exceed 600 seconds.
- The audio must not contain any customer-specific or other sensitive information.

ⓘ Note

The `audio` element is not supported by the **Long Audio API**. For long-form text-to-speech, use the **batch synthesis API (Preview)** instead.

Usage of the `audio` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>src</code>	The URI location of the audio file. The audio must be hosted on an internet-accessible HTTPS endpoint. HTTPS is required, and the domain hosting the file must present a valid, trusted TLS/SSL certificate. We recommend that you put the audio file into Blob Storage in the same Azure region as the text-to-speech endpoint to minimize the latency.	Required

Audio examples

The supported values for attributes of the `audio` element were [described previously](#).

This SSML snippet illustrates how the `src` attribute is used to insert audio from two .wav files.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
```

```

<voice name="en-US-JennyNeural">
  <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>
</voice>
</speak>

```

Audio duration

Use the `mstts:audioduration` element to set the duration of the output audio. Use this element to help synchronize the timing of audio output completion. The audio duration can be decreased or increased between 0.5 to 2 times the rate of the original audio. The original audio here is the audio without any other rate settings. The speaking rate will be slowed down or sped up accordingly based on the set value.

The audio duration setting is applied to all input text within its enclosing `voice` element. To reset or change the audio duration setting again, you must use a new `voice` element with either the same voice or a different voice.

Usage of the `mstts:audioduration` element's attributes are described in the following table.

Attribute	Description	Required or optional

Attribute	Description	Required or optional
value	<p>The requested duration of the output audio in either seconds (such as <code>2s</code>) or milliseconds (such as <code>2000ms</code>).</p> <p>This value should be within 0.5 to 2 times the original audio without any other rate settings. For example, if the requested duration of your audio is <code>30s</code>, then the original audio must have otherwise been between 15 and 60 seconds. If you set a value outside of these boundaries, the duration is set according to the respective minimum or maximum multiple.</p> <p>Given your requested output audio duration, the Speech service adjusts the speaking rate accordingly. Use the voice list API and check the <code>WordsPerMinute</code> attribute to find out the speaking rate of the neural voice that you're using. You can divide the number of words in your input text by the value of the <code>WordsPerMinute</code> attribute to get the approximate original output audio duration. The output audio will sound most natural when you set the audio duration closest to the estimated duration.</p>	Required

mstts audio duration examples

The supported values for attributes of the `mstts:audioduration` element were [described previously](#).

In this example, the original audio is around 15 seconds. The `mstts:audioduration` element is used to set the audio duration to 20 seconds (`20s`).

XML

```

<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <mstts:audioduration value="20s"/>
    If we're home schooling, the best we can do is roll with what each day
    brings and try to have fun along the way.
    A good place to start is by trying out the slew of educational apps that are
    helping children stay happy and smash their schooling at the same time.
  </voice>
</speak>
```

Background audio

You can use the `mstts:backgroundaudio` element 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 loops. If it's longer than the text-to-speech, it stops when the fade out has finished.

Only one background audio file is allowed per SSML document. You can intersperse `audio` tags within the `voice` element to add more audio to your SSML document.

ⓘ Note

The `mstts:backgroundaudio` element should be put in front of all `voice` elements. If specified, it must be the first child of the `speak` element.

The `mstts:backgroundaudio` element is not supported by the [Long Audio API](#). For long-form text-to-speech, use the [batch synthesis API \(Preview\)](#) instead.

Usage of the `mstts:backgroundaudio` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>src</code>	The URI location of the background audio file.	Required
<code>volume</code>	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
<code>fadein</code>	The duration of the background audio fade-in as 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
<code>fadeout</code>	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

mstts backgroundaudio examples

The supported values for attributes of the `mstts:backgroundaudio` element were [described previously](#).

XML

```
<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="en-US-JennyNeural">
      The text provided in this document will be spoken over the
      background audio.
    </voice>
</speak>
```

Next steps

- SSML overview
- SSML document structure and events
- Language support: Voices, locales, languages

Additional resources

Documentation

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Speech Synthesis Markup Language \(SSML\) document structure and events - Speech service - Azure Cognitive Services](#)

Learn about the Speech Synthesis Markup Language (SSML) document structure.

[Speech phonetic alphabets - Speech service - Azure Cognitive Services](#)

This article presents Speech service phonetic alphabet and International Phonetic Alphabet (IPA) examples.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Install the Speech SDK - Azure Cognitive Services](#)

In this quickstart, you'll learn how to install the Speech SDK for your preferred programming language.

[Troubleshoot the Speech SDK - Speech service - Azure Cognitive Services](#)

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

The Azure Speech CLI - Azure Cognitive Services

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Text-to-speech quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Show 5 more](#)

Pronunciation with SSML

Article • 02/03/2023 • 12 minutes to read

You can use Speech Synthesis Markup Language (SSML) with text-to-speech to specify how the speech is pronounced. For example, you can use SSML with phonemes and a custom lexicon to improve pronunciation. You can also use SSML to define how a word or mathematical expression is pronounced.

Refer to the sections below for details about how to use SSML elements to improve pronunciation. For more information about SSML syntax, see [SSML document structure and events](#).

phoneme element

The `phoneme` element is used for phonetic pronunciation in SSML documents. 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 might represent multiple spoken sounds. Consider the different `en-US` 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."

ⓘ Note

For a list of locales that support phonemes, see footnotes in the language support table.

Usage of the `phoneme` element's attributes are described in the following table.

Attribute	Description	Required or optional

Attribute	Description	Required or optional
alphabet	<p>The phonetic alphabet to use when you synthesize the pronunciation of the string in the <code>ph</code> attribute. The string that specifies the alphabet must be specified in lowercase letters. The following options are the possible alphabets that you can specify:</p> <ul style="list-style-type: none"> • <code>ipa</code> – See SSML phonetic alphabets • <code>sapi</code> – See SSML phonetic alphabets • <code>ups</code> – See Universal Phone Set • <code>x-sampa</code> – See SSML phonetic alphabets <p>The alphabet applies only to the <code>phoneme</code> in the element.</p>	Optional
ph	<p>A string containing phones that specify the pronunciation of the word in the <code>phoneme</code> element. If the specified string contains unrecognized phones, text-to-speech rejects the entire SSML document and produces none of the speech output specified in the document.</p> <p>For <code>ipa</code>, to stress one syllable by placing stress symbol before this syllable, you need to mark all syllables for the word. Or else, the syllable before this stress symbol will be stressed. For <code>sapi</code>, if you want to stress one syllable, you need to place the stress symbol after this syllable, whether or not all syllables of the word are marked.</p>	Required

phoneme examples

The supported values for attributes of the `phoneme` element were [described previously](#). In the first two examples, the values of `ph="tə. 'meɪ.tou"` or `ph="tə'meɪ'tou"` are specified to stress the syllable `meɪ`.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <phoneme alphabet="ipa" ph="tə. 'meɪ.tou"> tomato </phoneme>
    </voice>
</speak>
```

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
```

```
<voice name="en-US-JennyNeural">
    <phoneme alphabet="ipa" ph="təmər'tou"> tomato </phoneme>
</voice>
</speak>
```

XML

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <phoneme alphabet="sapi" ph="iy eh n y uw eh s"> en-US </phoneme>
    </voice>
</speak>
```

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <s>His name is Mike <phoneme alphabet="ups" ph="JH AU"> Zhou
    </phoneme></s>
    </voice>
</speak>
```

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
    <voice name="en-US-JennyNeural">
        <phoneme alphabet='x-sampa' ph='he."lou'>hello</phoneme>
    </voice>
</speak>
```

Custom lexicon

You can define how single entities (such as company, a medical term, or an emoji) are read in SSML by using the `phoneme` and `sub` elements. To define how multiple entities are read, create an XML structured custom lexicon file. Then you upload the custom lexicon XML file and reference it with the SSML `lexicon` element.

ⓘ Note

For a list of locales that support custom lexicon, see footnotes in the [language support](#) table.

The `lexicon` element is not supported by the **Long Audio API**. For long-form text-to-speech, use the **batch synthesis API (Preview)** instead.

Usage of the `lexicon` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>uri</code>	The URI of the publicly accessible custom lexicon XML file with either the <code>.xml</code> or <code>.pls</code> file extension. Using Azure Blob Storage is recommended but not required. For more information about the custom lexicon file, see Pronunciation Lexicon Specification (PLS) Version 1.0 .	Required

Custom lexicon examples

The supported values for attributes of the `lexicon` element were [described previously](#).

After you've published your custom lexicon, you can reference it from your SSML. The following SSML example references a custom lexicon that was uploaded to <https://www.example.com/customlexicon.xml>.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="http://www.w3.org/2001/mstts"
       xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <lexicon uri="https://www.example.com/customlexicon.xml"/>
    BTW, we will be there probably at 8:00 tomorrow morning.
    Could you help leave a message to Robert Benigni for me?
  </voice>
</speak>
```

Custom lexicon file

To define how multiple entities are read, you can define them in a custom lexicon XML file with either the `.xml` or `.pls` file extension.

ⓘ Note

The custom lexicon file is a valid XML document, but it cannot be used as an SSML document.

Here are some limitations of the custom lexicon file:

- **File size:** The custom lexicon file size is limited to a maximum of 100 KB. If the file size exceeds the 100-KB limit, the synthesis request fails.
- **Lexicon cache refresh:** The custom lexicon is cached with the URI as the key on text-to-speech when it's first loaded. The lexicon with the same URI won't be reloaded within 15 minutes, so the custom lexicon change needs to wait 15 minutes at the most to take effect.

The supported elements and attributes of a custom lexicon XML file are described in the [Pronunciation Lexicon Specification \(PLS\) Version 1.0](#). Here are some examples of the supported elements and attributes:

- The `lexicon` element contains at least one `lexeme` element. Lexicon contains the necessary `xml:lang` attribute to indicate which locale it should be applied for. One custom lexicon is limited to one locale by design, so if you apply it for a different locale, it won't work. The `lexicon` element also has an `alphabet` attribute to indicate the alphabet used in the lexicon. The possible values are `ipa` and `x-microsoft-sapi`.
- Each `lexeme` element contains at least one `grapheme` element and one or more `grapheme`, `alias`, and `phoneme` elements. The `lexeme` element is case sensitive in the custom lexicon. For example, if you only provide a phoneme for the `lexeme` "Hello", it won't work for the `lexeme` "hello".
- The `grapheme` element contains text that describes the [orthography](#).
- The `alias` elements are used to indicate the pronunciation of an acronym or an abbreviated term.
- The `phoneme` element provides text that describes how the `lexeme` is pronounced. The syllable boundary is '.' in the IPA alphabet. The `phoneme` element can't contain white space when you use the IPA alphabet.
- When the `alias` and `phoneme` elements are provided with the same `grapheme` element, `alias` has higher priority.

Microsoft provides a [validation tool for the custom lexicon](#) that helps you find errors (with detailed error messages) in the custom lexicon file. Using the tool is recommended before you use the custom lexicon XML file in production with the Speech service.

Custom lexicon file examples

The following XML example (not SSML) would be contained in a custom lexicon `.xml` file. When you use this custom lexicon, "BTW" is read as "By the way." "Benigni" is read with the provided IPA "bɛ'ni:nji."

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
    xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
        http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
    alphabet="ipa" xml:lang="en-US">
    <lexeme>
        <grapheme>BTW</grapheme>
        <alias>By the way</alias>
    </lexeme>
    <lexeme>
        <grapheme> Benigni </grapheme>
        <phoneme> bɛ'ni:nji</phoneme>
    </lexeme>
    <lexeme>
        <grapheme> 😊</grapheme>
        <alias>test emoji</alias>
    </lexeme>
</lexicon>
```

You can't directly set the pronunciation of a phrase by using the custom lexicon. If you need to set the pronunciation for an acronym or an abbreviated term, first provide an `alias`, and then associate the `phoneme` with that `alias`. For example:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
    xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
        http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
    alphabet="ipa" xml:lang="en-US">
    <lexeme>
        <grapheme>Scotland MV</grapheme>
        <alias>ScotlandMV</alias>
    </lexeme>
    <lexeme>
        <grapheme>ScotlandMV</grapheme>
        <phoneme> 'skɒtlənd.'mɪ:dɪəm.weɪv</phoneme>
    </lexeme>
</lexicon>
```

You could also directly provide your expected `alias` for the acronym or abbreviated term. For example:

XML

```
<lexeme>
  <grapheme>Scotland MV</grapheme>
  <alias>Scotland Media Wave</alias>
</lexeme>
```

The preceding custom lexicon XML file examples use the IPA alphabet, which is also known as the IPA phone set. We suggest that you use the IPA because it's the international standard. For some IPA characters, they're the "precomposed" and "decomposed" version when they're being represented with Unicode. The custom lexicon only supports the decomposed Unicode.

The Speech service defines a phonetic set for these locales: `en-US`, `fr-FR`, `de-DE`, `es-ES`, `ja-JP`, `zh-CN`, `zh-HK`, and `zh-TW`. For more information on the detailed Speech service phonetic alphabet, see the [Speech service phonetic sets](#).

You can use the `x-microsoft-sapi` as the value for the `alphabet` attribute with custom lexicons as demonstrated here:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
  alphabet="x-microsoft-sapi" xml:lang="en-US">
  <lexeme>
    <grapheme>BTW</grapheme>
    <alias> By the way </alias>
  </lexeme>
  <lexeme>
    <grapheme> Benigni </grapheme>
    <phoneme> b eh 1 - n iy - n y iy </phoneme>
  </lexeme>
</lexicon>
```

say-as element

The `say-as` element indicates the content type, such as number or date, of the element's text. This element provides guidance to the speech synthesis engine about how to pronounce the text.

Usage of the `say-as` element's attributes are described in the following table.

Attribute	Description	Required or optional
interpret-as	Indicates the content type of an element's text. For a list of types, see the following table.	Required
format	Provides additional information about the precise formatting of the element's text for content types that might have ambiguous formats. SSML defines formats for content types that use them. See the following table.	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 detail.	Optional

The following content types are supported for the `interpret-as` and `format` attributes. Include the `format` attribute only if `format` column isn't empty in the table below.

interpret-as	format	Interpretation
characters, spell-out		<p>The text is spoken as individual letters (spelled out). The speech synthesis engine pronounces:</p> <pre><say-as interpret-as="characters">test</say-as></pre> <p>As "T E S T."</p>
cardinal, number	None	<p>The text is spoken as a cardinal number. The speech synthesis engine pronounces:</p> <pre>There are <say-as interpret-as="cardinal">10</say-as> options</pre> <p>As "There are ten options."</p>
ordinal	None	<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>
number_digit	None	<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>

interpret-as	format	Interpretation
fraction	None	The text is spoken as a fractional number. The speech synthesis engine pronounces: <code><say-as interpret-as="fraction">3/8</say-as> of an inch</code> As "three eighths of an inch."
date	dmy, mdy, ymd, ydm, ym, my, md, dm, d, m, y	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: <code>Today is <say-as interpret-as="date" format="mdy">10-19-2016</say-as></code> As "Today is October nineteenth two thousand sixteen."
time	hms12, hms24	The text is spoken as a time. The <code>format</code> attribute specifies whether the time is specified by using a 12-hour clock (hms12) or a 24-hour clock (hms24). Use a colon to separate numbers representing hours, minutes, and seconds. Here are some valid time examples: 12:35, 1:14:32, 08:15, and 02:50:45. The speech synthesis engine pronounces: <code>The train departs at <say-as interpret-as="time" format="hms12">4:00am</say-as></code> As "The train departs at four A M."
duration	hms, hm, ms	The text is spoken as a duration. The <code>format</code> attribute specifies the duration's format (<i>h=hour, m=minute, and s=second</i>). The speech synthesis engine pronounces: <code><say-as interpret-as="duration">01:18:30</say-as></code> As "one hour eighteen minutes and thirty seconds". Pronounces: <code><say-as interpret-as="duration" format="ms">01:18</say-as></code> As "one minute and eighteen seconds". This tag is only supported on English and Spanish.

interpret-as	format	Interpretation
telephone	None	<p>The text is spoken as a telephone number. The <code>format</code> attribute can contain digits that represent a country code. Examples are "1" for the United States or "39" for Italy. The speech synthesis engine can use this information to guide its pronunciation of a phone number. The phone number might also include the country code, and if so, takes precedence over the country code in the <code>format</code> attribute. The speech synthesis engine pronounces:</p> <pre>The number is <say-as interpret-as="telephone" format="1">(888) 555-1212</say-as></pre> <p>As "My number is area code eight eight eight five five five one two one two."</p>
currency	None	<p>The text is spoken as a currency. The speech synthesis engine pronounces:</p> <pre><say-as interpret-as="currency">99.9 USD</say-as></pre> <p>As "ninety-nine US dollars and ninety cents."</p>
address	None	<p>The text is spoken as an address. The speech synthesis engine pronounces:</p> <pre>I'm at <say-as interpret-as="address">150th CT NE, Redmond, WA</say-as></pre> <p>As "I'm at 150th Court Northeast Redmond Washington."</p>
name	None	<p>The text is spoken as a person's name. The speech synthesis engine pronounces:</p> <pre><say-as interpret-as="name">ED</say-as></pre> <p>As [æd].</p> <p>In Chinese names, some characters pronounce differently when they appear in a family name. For example, the speech synthesis engine says 仇 in</p> <pre><say-as interpret-as="name">仇先生</say-as></pre> <p>As [qiú] instead of [chóu].</p>

say-as examples

The supported values for attributes of the `say-as` element were [described previously](#).

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 PM."

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <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>
  </voice>
</speak>
```

sub element

Use the `sub` element to indicate that the alias attribute's text value should be pronounced instead of the element's enclosed text. In this way, the SSML contains both a spoken and written form.

Usage of the `sub` element's attributes are described in the following table.

Attribute	Description	Required or optional
<code>alias</code>	The text value that should be pronounced instead of the element's enclosed text.	Required

sub examples

The supported values for attributes of the `sub` element were [described previously](#).

The speech synthesis engine speaks the following example as "World Wide Web Consortium."

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
xml:lang="en-US">
  <voice name="en-US-JennyNeural">
    <sub alias="World Wide Web Consortium">W3C</sub>
```

```
</voice>  
</speak>
```

Pronunciation with MathML

The Mathematical Markup Language (MathML) is an XML-compliant markup language that describes mathematical content and structure. The Speech service can use the MathML as input text to properly pronounce mathematical notations in the output audio.

ⓘ Note

The MathML elements (tags) are currently supported by all neural voices in the `en-US` and `en-AU` locales.

All elements from the [MathML 2.0](#) and [MathML 3.0](#) specifications are supported, except the MathML 3.0 [Elementary Math](#) elements.

Take note of these MathML elements and attributes:

- The `xmlns` attribute in `<math xmlns="http://www.w3.org/1998/Math/MathML">` is optional.
- The `semantics`, `annotation`, and `annotation-xml` elements don't output speech, so they're ignored.
- If an element isn't recognized, it will be ignored, and the child elements within it will still be processed.

The MathML entities aren't supported by XML syntax, so you must use the corresponding [unicode characters](#) to represent the entities, for example, the entity `©` should be represented by its unicode characters `©`, otherwise an error will occur.

MathML examples

The text-to-speech output for this example is "a squared plus b squared equals c squared".

XML

```
<speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis'  
      xmlns:mstts='http://www.w3.org/2001/mstts' xml:lang='en-US'><voice name='en-US-JennyNeural'><math xmlns='http://www.w3.org/1998/Math/MathML'><msup>
```

```
<mi>a</mi><mn>2</mn></msup><mo>+</mo><msup><mi>b</mi><mn>2</mn></msup><mo>=</mo><msup><mi>c</mi><mn>2</mn></msup></math></voice></speak>
```

Next steps

- SSML overview
 - SSML document structure and events
 - Language support: Voices, locales, languages
-

Additional resources

Documentation

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[Keyword recognition overview - Speech service - Azure Cognitive Services](#)

An overview of the features, capabilities, and restrictions for keyword recognition by using the Speech Software Development Kit (SDK).

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Configure the Speech CLI datastore - Speech service - Azure Cognitive Services](#)

Learn how to configure the Speech CLI datastore.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Show 5 more](#)

SSML phonetic alphabets

Article • 02/03/2023 • 80 minutes to read

Phonetic alphabets are used with the [Speech Synthesis Markup Language \(SSML\)](#) to improve the pronunciation of text-to-speech voices. To learn when and how to use each alphabet, see [Use phonemes to improve pronunciation](#).

Speech service supports the [International Phonetic Alphabet \(IPA\)](#) suprasegmentals that are listed here. You set `ipa` as the `alphabet` in [SSML](#).

ipa	Symbol	Note
'	Primary stress	Don't use single quote (' or ') though it looks similar.
,	Secondary stress	Don't use comma (,) though it looks similar.
.	Syllable boundary	
:	Long	Don't use colon (: or :) though it looks similar.
()	Linking	

Tip

You can use [the international phonetic alphabet keyboard](#) to create the correct `ipa` suprasegmentals.

For some locales, Speech service defines its own phonetic alphabets, which ordinarily map to the [International Phonetic Alphabet \(IPA\)](#). The eight locales that support the Microsoft Speech API (SAPI, or `sapi`) are en-US, fr-FR, de-DE, es-ES, ja-JP, zh-CN, zh-HK, and zh-TW. For those eight locales, you set `sapi` or `ipa` as the `alphabet` in [SSML](#).

See the sections in this article for the phonemes that are specific to each locale.

Note

The following tables list viseme IDs corresponding to phonemes for different locales. When viseme ID is 0, it indicates silence.

ar-EG/ar-SA

Vowels

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	بازار	أَبْ	أَعْطَاب
d	19	دابة	أَبَادَ	أَفْسَدَ
g	20	جَملُ	رَجَبُ	مَرْجَ
k	20	كآخر	أَكْلُ	أَوْرَاك
t	19	تآخي	مُرْتَبُ	أُخْتٌ
d ^č	19	ضاعن	مُضْرُّ	مَرْضٌ
q	20	قَمْرٌ	أَتَعَاقدَ	أَرْقَ
t ^č	19	طالبان	أَحاطوهُ	رَبْطٌ
? [?]	19	فُقِيٌّ	فَأُزْ	السَّمَاءُ
f	18	فَنٌّ	يَفْرُ	شَرْفٌ
h	12	هِيَاجٌ	أَحَبَّتْهُ	الْأَبْلَهُ
ħ	12	حُرْ	رَحْمٌ	الْبَلَحٌ
s	15	سُرْ	قِسْمٌ	الْجَالِسٌ
θ	19	ثَرَى	أَخَدَاثٌ	الْحَارَثٌ
z	15	زُرٌّ	قَزْمٌ	الْحُبْزٌ

ipa	viseme	Example 1	Example 2	Example 3
ð̚	17	ظَنَّ	أَظْهَرَ	قَيْظَ
ð	17	ذَلِكَ	أَذْدَى	أَنْفُذَ
غ	20	غَنِيٌّ	أَدْمَعَةً	دِمَاغَ
x	12	خَبَرَ	رَحْوُ	أَخَّ
ج	16	شَافَتْ	خَشْيَّةً	وَحْشَ
s̚	15	صَلَّةً	وَصَلَّ	عَوْصَ
j	6	يَوْمُ	أَدْوِيَةً	بِمُدْبِرِيَّ
w	7	وَلَدُ	لَوْنُ	ما كا وَ
ل	14	لَيْسَ	عَلْمُ	عَمَلَ
m	21	مَجْدُ	ثَمَنُ	قَلَمَ
n	19	نَائِبُ	أَذْيَانًا	أَمْنَ
r	13	رَكْبُ	قَرْنُ	بِمُكَرَّرَ
ـ	12	عَامَّا	لَعْبُ	بَيْعَ

bg-BG

Vowels

ipa	viseme	Example 1	Example 2	Example 3
i	6	искат	сценаристката	лечими
ε	4	елен	цена	оспорените
ɔ	3	оправя	което	колело
a	2	ангел	докосват	цена
u	7	уклончивата	хубавичка	табу
Ја	6,2	ябълка	оял	залај
ꙗ	1	ъгъл	ъгъл	имамбаялдъ
Ѩ	6,7	ютия	отвоювал	стою

Consonant

ipa	viseme	Example 1	Example 2	Example 3
n	19	неразбираемия	преразпределените	искан
ʒ	16	жълт	тъжен	таралеж
k	20	камък	вкарвал	петък
tʂ	19,15	цар	меценат	царевец
t	19	тайна	натъжени	прост
p	21	приказката	натъпкан	прилеп
r	13	разлика	прозорец	хитър
s	15	сърна	месец	процес
d	19	дом	меден	джаред
x	12	хора	доход	цех
z ^j	15	зян	Замразявайки	
l ^j	14	любов	влюбване	
l	14	лебед	блед	хмел
n ^j	19	някога	понякога	
v	18	време	навреме	лъвов
m	21	море	време	корем
b	21	баба	риба	ястреб
g	20	гарван	наруган	драг
dʒ	19,16	Джорджева	тютюнджийския	пледж
f	18	филм	реформа	релеф
m ^j	21	мях	смях	
t ^j	19	тяхна	стяга	
r ^j	13	рядка	впрягайки	
p ^j	21	пяхме	спя	

ipa	viseme	Example 1	Example 2	Example 3
d ^j	19	дядовите	задявайки	
j	6	йод	пейо	ручей
v ^j	18	вяло	посивяло	
s ^j	15	сякаш	всяка	
b ^j	21	бяла	Избягваме	
k ^j	20	кюпа	прокюрдският	
g ^j	20	гюргево	панагюрското	
f ^j	18	фючърс	Парфюмерия	
z	15	зима	изразходват	рязказ
ʃ	16	шума	машина	изпечеш
tʃ	19,16	чак	случай	меч
ðz	19,15	дзифт	скрънда	Лодз

ca-ES

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	amen	amaro	està
ɔ	3	odre	ofertori	microtò
ə	1	estan	seré	aigua
e	4	érem	feta	seré
ɛ	4	ecosistema	incorrecta	haver
i	6	itinerants	itinerants	zombi
o	8	ombra	retondre	omissió
u	7	universitaris	candidatures	crono

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	baba	dobra	
β	21	vià	baba	
tʃ	19,16	txadià	matxucs	faig
d	19	dedicada	conduïa	navidad
ð	17	The_Sun	dedicada	trinidad
f	18	facilitades	afectarà	àgraf
g	20	gracia	congratula	
ɣ	20		aigua	
j	6	hiena	esplaia	cofoi
ðʒ	19,16	djakarta	compostatge	george
k	20	curós	dodecà	doblec
l	14	laberint	miolar	preval
ʎ	14	lligada	millorarà	perbull
m	21	macadàmies	femar	sublim
n	19	necessaris	sanitaris	alterament
ŋ	20		algonquí	albenc
ɲ	19	nyasa	remenjar	alemany
p	21	pegues	estepa	cap
r	19		caro	càrter
r̥	13	rabada	carro	lofòfor
s	15	ceri	cursar	cus
ʃ	16	xacar	microxip	midraix
t	19	tabacaires	estratifica	debatut
θ	19	ceará	vecinos	Álvarez

ipa	viseme	Example 1	Example 2	Example 3
w	7	westfalià	inaugurar	inscriu
x	12	juanita	mujeres	heinrich
z	15	zelar	brasils	alianze
ʒ	16	gebrada	ascogènic	orange

CS-CZ

Vowels

ipa	viseme	Example 1	Example 2	Example 3
i	6	ideální	handicapované	giganty
ɛ	4	efektní	grotesce	hygieně
a	2	agens	infarktu	extra
o	8	oáz	hřbitovy	čistο
u	7	ubrání	komponuje	grilu
i:	6	íránské	jedinými	generační
ɛ:	4	éry	jednoduchého	gumové
a:	2	árijské	brán	hořká
o:	8	ódu	famózního	jó
u:	7	úbočí	petúnie	grilů
œ	8,4	ouha	fanouška	hanbou
œ̄	2,4	auto	hydrauliky	
ɛ̄	4,4	euforie	terapeutická	
ə	1			

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	padá	hyperaktivní	handicap
b	21	balon	hraběte	
t	19	tabák	étos	agent
d	19	disco	čidel	
c	16	ťuhýk	kandidáti	kaprad'
ʒ	16	dábel	hrdiny	
k	20	kaligrafie	důkazní	grafik
g	20	gangstera	hygienici	
ts	19,15	civilistům	evoluce	klávesnic
ðz	19,15		leckde	
tʃ	19,16	čepice	hlídači	klíč
ðʒ	19,16	džusy	kilojoulů	
f	18	fádní	grafice	graf
v	18	vabank	exkluzivita	
s	15	sekundě	grimasami	impuls
z	15	záhadným	gruzínské	
r	13	řad	hořícím	
ʃ	16	šálků	grošů	kéž
ʒ	16	žab	loupeže	
j	6	já	číhající	línej
x	12	chlapcem	lichotí	domorodých
h	12	hektar	historického	
r̥	13	rabat	guru	faktor
l̥	14	ladu	eliminovat	netajil
m̥	21	macešsky	amatérský	cením

ipa	viseme	Example 1	Example 2	Example 3
n	19	následkům	cenné	gurmán
ŋ	20		kolonky	
n̪	19	ňadra	komorně	kolegyň
m̪	21		komfort	
ř	13	třída	extratřídy	komentář

da-DK

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	abitur	kortfattet	marina
a	2	abnorme	picaresk	varer
a:	2	ara	pirat	vandkar
ɛ	4	edderkop	andægtig	tomatpuré
ɛ:	4	eventyr	dyrlæge	
ɔ	3	onde	dysfunktion	Suså
ø	2	ånd	abehånd	akkumulator
ø:	2	årbog	fyrreårig	rickshaw
ɔ:	3	åbne	modstående	husarblå
e	4		buntmagere	bortgifter
æ:	1	abegilde	flagdage	hovedfag
e	4	editere	klarheden	kanapé
ø	1	ødem	mælkebøtte	miljø
ø:	1	øde	bortførere	lindø
ə	1	Eyolf	meneders	æde
e:	4	edelweiss	placebo	Culmsee

ipa	viseme	Example 1	Example 2	Example 3
i	6	iaften	lipizzaner	bankkonti
i:	6	ilinger	industrien	bagi
o	8	øase	udelød	bravo
ø	4	øm	varmerør	bøh
œ	4	Earl	arbejdsprøves	Edinburgh
o:	8	øberst	bevatrons	congo
u	7	udøver	spektakulær	endnu
u:	7	uge	spidsbue	eisuu
y	4	ybsalon	underkrydder	menu
y:	4	yde	vidsyn	dødssyg

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	blandt	gabende	dåb
d	19	då	hævder	grædt
ð	17	thousand	baaden	grad
f	18	fabrik	spøgefugle	boligstof
g	20	gaader	fåborgenser	fik
h	12	håb	trægheden	
j	6	hjorte	miljø	bagtøj
k ^h	20	kabale	bortkomme	Cuc
l	14	lå	romanblad	fatal
m	21	maya	taxametret	calcium
n	19	nå	togene	bredden
ŋ	20		funktion	klarering

ipa	viseme	Example 1	Example 2	Example 3
p ^h	21	pa	culpa	
?	19		affyre	bortgå
r	13	rå	areal	
e	4		jonglørkunst	bjørnebær
s	15	så	person	belønnes
c	16	Sjælland	benzinstation	affiche
t	19	toge	sidetal	administrationsapparat
v	18	vaagner	evaluere	dåkalv
w	7	walkman	prøve	meterlov

de-DE/de-CH/de-AT

German suprasegmentals

Example 1 (Onset for consonant, word-initial for vowel)	Example 2 (Intervocalic for consonant, word- medial nucleus for vowel)	Example 3 (Coda for consonant, word- final for vowel)	Comments
anders /a 1 n - d ax r s/	Multiplikationszeichen /m uh l - t iy - p l iy - k a - ts y ow 1 n s - ts ay - c n/	Biologie /b iy - ow - l ow - g iy 1/	Speech service phone set put stress after the vowel of the stressed syllable
Allgemeinwissen /a 2 l - g ax - m ay 1 n - v ih - s n/	Abfallentsorgungsfirma /a 1 p - f a l - ^ eh n t - z oh 2 ax r - g uh ng s - f ih ax r - m a/	Computertomographie /k oh m - p y uw 1 - t ax r - t ow - m ow - g r a - f iy 2/	The Speech service phone set puts stress after the vowel of the sub-stressed syllable

German vowels

sapi	ipa	VisemelD	Example 1	Example 2	Example 3
a:	a:	2	Aber	Maßstab	Schema

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
a	a	2	Abfall	Bach	Agatha
oh	ɔ	3	Osten	Pfosten	
eh:	ɛ:	4	Ähnlichkeit	Bär	Fasciae ¹
eh	ɛ	4	ändern	Prozent	Amygdalae
ax	ə	1	'verstauen ²	Aachen	Frage
iy	i:	6	Iran	abbiegt	Relativitätstheorie
ih	ɪ	6	Innung	singen	Woody
eu	ø:	1	Ösen	ablösten	Malmö
ow	o, o:	8	ohne	Balkon	Treptow
oe	æ	4	Öffnung	befördern	
ey	e, e:	4	Eberhard	abfegt	b
uw	u:	7	Udo	Hut	Akku
uh	ʊ	4	Unterschiedes	bunt	
ue	y:	4	Übermut	pflügt	Menü
uy	y	7	üppig	System	

1 Only in words of foreign origin, such as *Fasciae*.

2 Word-initial only in words of foreign origin, such as *Appointment*. Syllable-initial in 'verstauen.

German diphthong

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
ay	ai	2,6	einsam	Unabhängigkeit	Abtei
aw	au	2,7	außen	abbaust	Stau
oy	ɔy, ɔʏ	3,4	Euphorie	träumt	scheu

German semivowels

sapi	ipa	VisemelID	Example 1	Example 2	Example 3
ax r	e	4		abändern	locker

German consonants

sapi	ipa	VisemelID	Example 1	Example 2	Example 3
b	b	21	Bank		Pub ¹
d	d	19	danken	Lendl ²	Claude ³
jh	χ	16	Jeff	gemanagt	Change ⁴
f	f	18	Fahrt dauer	angriffslustig	abbruchreif
g	g	20	gut	Greg ⁵	
h	h	12	Hausanbau		
y	j	6	Jod	Reaktion	hui
k	k	20	Koma	Aspekt	Fleck
l	l	14	lau	ähneln	zuviel
m	m	21	Mut	Amt	Lehm
n	n	19	nun	und	Huhn
ng	ŋ	20	Nguyen ⁶	Schwank	Ring
p	p	21	Partner	abrupt	Tip
pf	pf	21,18	Pferd	dampft	Topf
r	r, r̥, ʁ	13	Reise	knurrt	Haar
s	s	15	Staccato ⁷	bist	mies
sh	ʃ	16	Schule	mischt	lappisch
t	t	19	Traum	Straße	Mut
ts	ts	19,15	Zug	Arzt	Witz
ch	tʃ	19,16	Tschechien	aufgeputscht	bundesdeutsch
v	v	18	winken	Qualle	Groove ⁸

sapi	ipa	VisemelID	Example 1	Example 2	Example 3
x	x ⁹ , ç ¹⁰	12	Bacherach ¹¹	Macht möglichst	Schmach 'ich
z	z	15	super		
zh	ʒ	16	Genre	Breezinski	Edvige

1 Only in words of foreign origin, such as *Pub*.

2 Only in words of foreign origin, such as *Lendl*.

3 Only in words of foreign origin, such as *Claude*.

4 Only in words of foreign origin, such as *Change*.

5 Word-terminally only in words of foreign origin, such as *Greg*.

6 Only in words of foreign origin, such as *Nguyen*.

7 Only in words of foreign origin, such as *Staccato*.

8 Only in words of foreign origin, such as *Groove*.

9 The IPA x is a hard "ch" after all non-front vowels (a, aa, oh, ow, uh, uw, and the diphthong aw).

10 The IPA ç is a soft "ch" after front vowels (ih, iy, eh, ae, uy, ue, oe, eu, and diphthongs ay, oy) and consonants.

11 Word-initial only in words of foreign origin, such as *Juan*. Syllable-initial also in words such as *Bacherach*.

German oral consonants

sapi	ipa	VisemelID	Example
^	?	19	beachtlich /b ax - ^ a 1 x t - l ih c/

ⓘ Note

We need to add a [gs] phone between two distinct vowels, except when the two vowels are a genuine diphthong. This oral consonant is a glottal stop. For more information, see [glottal stop ↴](#). Besides, de-CH, de-AT locales don't support SAPI phones now.

el-GR

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	άκουσμα	μαγαζάκι	ουδέτερα
e	4	αίτιο	κατέχω	ουδέποτε
i	6	εισπνοή	ξυρισμένο	εαρινή
o	8	όαση	δώρο	δυ**ο
u	7	ουγγρικό	παπούτσι	μαϊμού

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	μπορώ	φάμπρικα	παμπ
c	16	και	σακί	
ç	12	χέρι	μοναχή	
d	19	ντύνει	πέντε	οφ-σάιντ
ð	17	δρόμος	κραδασμοί	
dz	19,15	τζάμι	φλάντζα	
f	18	φεύγω	καφετέρια	ουφ
g	20	γκρέμισε	όγκος	πινγκ-πονγκ
ɣ	20	γαστρονομική	μεγαλόπνοα	
ʒ	16	γκισέ	φαράγγι	
j	6			
ɟ	12	γέρος	κραγιόν	
k	20	κάρτα	ιππικό	κρακ
l	14	λόγος	μιλώ	προφίλ
m	21	μιλώ	δραματική	πριμ
n	19	νόμιμο	Δανέζα	πριν
p	21	πίνω	απέτυχαν	σελοτέιπ

ipa	viseme	Example 1	Example 2	Example 3
ɾ	19	ρούχα	εαρινή	σέντερ
s	15	σελίδα	μέσο	πλάτος
t	19	τότε	τότε	κιτ
θ	19	Θέλω	φορολογηθείς	μαμούθ
tʂ	19,15	τσάγια	χαράτσι	ματς
v	18	βράδυ	διαβάζω	μοβ
x	12	χρόνος	μονάχα	αχ
z	15	ζέστη	χιονίζει	πλαζ

en-GB/en-IE/en-AU

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a:	2		fast	bra
æ	1		fat	
ʌ	1		bug	
ɛə	4,1			hair
aʊ	2,4	out	mouth	how
ə	1	a		driver
aɪ	2,6		five	
ɛ	4	egg	dress	
ɜ:	5	ernest	shirt	fur
eɪ	4,6	ailment	lake	pay
ɪ	6		adding	
ɪə	6,1		beard	hear
i:	6	eat	seed	see

ipa	viseme	Example 1	Example 2	Example 3
b	2		pod	
ɔ:	3		dawn	
əʊ	1,4		code	pillow
ɔɪ	3,6		point	boy
ʊ	4		look	
ʊə	4,1			tour
u:	7		food	two

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	bike	ribbon	rib
tʃ	19,16	challenge	nature	rich
d	19	date	caddy	slid
ð	17	this	father	breathe
f	18	face	laughing	enough
g	20	gold	bragging	beg
h	12	hurry	ahead	
j	6	yes		
dʒ	19,16	gin	badger	bridge
k	20	cat	lucky	truck
l	14	left	gallon	fill
m	21	mile	limit	ham
n	19	nose	phonetic	tin
ŋ	20		singer	long
p	21	price	super	tip

ipa	viseme	Example 1	Example 2	Example 3
ɹ	13	rate	very	
s	15	say	sissy	pass
ʃ	16	shop	cashier	leash
t	19	top	kitten	bet
θ	19	theatre	mathematics	breath
v	18	very	liver	have
w	7	will		
z	15	zero	blizzard	rose
ʒ	16		vision	beige

en-US/en-CA

English suprasegmentals

Example 1 (onset for consonant, word-initial for vowel)	Example 2 (intervocalic for consonant, word- medial nucleus for vowel)	Example 3 (coda for consonant, word-final for vowel)	Comments
burger /b er 1 r - g ax r/	falafel /f ax - l aa 1 - f ax l/	guitar /g ih - t aa 1 r/	The Speech service phone set puts stress after the vowel of the stressed syllable.
inopportune /ih 2 - n aa - p ax r - t uw 1 n/	dissimilarity /d ih - s ih 2- m ax - l eh 1 - r ax - t iy/	workforce /w er 1 r k - f ao 2 r s/	The Speech service phone set puts stress after the vowel of the sub-stressed syllable.

English vowels

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
iy	i	6	eat	feel	valley

sapi	ipa	VisemelID	Example 1	Example 2	Example 3
ih	I	6	if	fill	
ey	eɪ	4,6	ate	gate	day
eh	ɛ	4	every	pet	meh (rare word-final)
ae	æ	1	active	cat	nah (rare word-final)
aa	a	2	obstinate	poppy	rah (rare word-final)
ao	ɔ	3	orange	cause	Utah
uh	ʊ	4	book		
ow	oʊ	8,4	old	clone	go
uw	u	7	Uber	boost	too
ah	ʌ	1	uncle	cut	
ay	aɪ	11	ice	bite	fly
aw	aʊ	9	out	south	cow
oy	ɔɪ	10	oil	join	toy
y uw	ju	6,7	Yuma	human	few
ax	ə	1	ago	woman	area

English R-colored vowels

sapi	ipa	VisemelID	Example 1	Example 2	Example 3
ih r	ɪɹ	6,13	ears	tiramisu	near
eh r	ɛɹ	4,13	airplane	apparently	scare
uh r	ʊɹ	4,13			cure
ay r	aɪɹ	11,13	Ireland	fireplace	choir
aw r	aʊɹ	9,13	hours	powerful	sour
ao r	ɔɹ	3,13	orange	moral	soar
aa r	əɹ	2,13	artist	start	car

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
er r	ɛ	5	earth	bird	fur
ax r	ə	1		allergy	supper

English semivowels

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
w	w	7	with, suede	always	
y	j	6	yard, few	onion	

English aspirated oral stops

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
p	p	21	put	happen	flap
b	b	21	big	number	crab
t	t	19	talk	capital	sought
d	d	19	dig	random	rod
k	k	20	cut	slacker	Iraq
g	g	20	go	ago	drag

English nasal stops

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
m	m	21	mat, smash	camera	room
n	n	19	no, snow	tent	chicken
ng	ŋ	20		link	sing

English fricatives

sapi	ipa	VisemeID	Example 1	Example 2	Example 3

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
f	f	18	fork	left	half
v	v	18	value	event	love
th	θ	19	thin	empathy	month
dh	ð	17	then	mother	smooth
s	s	15	sit	risk	facts
z	z	15	zap	busy	kids
sh	ʃ	16	sh e	abbreviation	rush
zh	ʒ	16	Jacques	pleasure	garage
h	h	12	help	enhance	a-ha!

English affricates

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
ch	tʃ	19,16	chin	future	attach
jh	dʒ	19,16	joy	original	orange

English approximants

sapi	ipa	VisemeID	Example 1	Example 2	Example 3
l	l	14	lid, glad	palace	chill
r	ɹ	13	red, bring	borrow	tar

ⓘ Note

en-CA locale doesn't support SAPI phones now.

es-ES

Vowels

sapi	ipa	viseme	Example 1	Example 2	Example 3
a	a	2	alto	cantar	casa
i	i	6	ibérica	avispa	taxi
e	e	4	elefante	atento	elefante
o	o	8	ocaso	encontrar	ocaso
u	u	7	usted	punta	Juanlu

Consonant

sapi	ipa	viseme	Example 1	Example 2	Example 3
b	b	21	baobab	cambio	amb
	β	21		baobab	baobab
ch	tʃ	19,16	cheque	coche	Marraquech
d	d	19	dedo	candado	portland
	ð	17		dedo	verdad
f	f	18	fácil	elefante	puf
g	g	20	ganga	ganga	dóping
	ɣ	20		agua	tuareg
j	j	6	iodo	caliente	rey
jj	ʃj	6,6		villa	
k	k	20	coche	boca	titánic
l	l	14	lápiz	ala	cordel
ll	ʎ	14	llave	conllevar	
m	m	21	morder	amar	álbum
n	n	19	nada	cena	ratón
nj	ɲ	19	ñaña	arañazo	
p	p	21	poca	topo	stop

sapi	ipa	viseme	Example 1	Example 2	Example 3
r	r	19		cara	abrir
rr	r	13	radio	corre	purr
s	s	15	saco	vaso	pelos
t	t	19	toldo	atar	disquet
th	θ	19	zebra	azul	lápiz
w	w	7	hueso	agua	guau
x	x	12	jota	ajo	reloj

💡 Tip

The es-ES Speech service phone set doesn't support the following Spanish IPA: β, δ, and γ. If they're needed, consider using the IPA directly.

es-MX

Vowels

ipa	VisemeID	Example 1	Example 2	Example 3
a	2	azúcar	tomate	ropa
e	4	eso	remero	amé
i	6	hilo	líquido	olí
o	8	hogar	olote	caso
u	7	uno	ninguno	tabú

Consonants

ipa	VisemeID	Example 1	Example 2	Example 3
b	21	bote		
β	21	órbita	envolvente	

ipa	VisemeID	Example 1	Example 2	Example 3
tʃ	19,16	chico	hacha	
d	19	dátil		
ð	17	orden	oda	
f	18	foco	oficina	
g	20	gajo		
ɣ	20	agua	hoguera	
j	6	iodo	caliente	rey
χχ	6,6		olla	
k	20	casa	ácaro	
l	14	loco	ala	
ʎ	14	llave	enyugo	
m	21	mata	amar	
n	19	nada	ano	
ɲ	19	ññoño	año	
p	21	papa	papa	
r	19		aro	
r̥	13	rojo	perro	
s	15	silla	asa	
t	19	tomate		soft
w	7	huevo		
x	12	jarra	hoja	

fi-Fl

Vowels

ipa	viseme	Example 1	Example 2	Example 3
-----	--------	-----------	-----------	-----------

ipa	viseme	Example 1	Example 2	Example 3
a	2	avautuu	vaihtuvan	pouta
äi	2,6	aika	vaihtuu	lauantai
äu	2,7	aura	uloskirjaudu	Passau
a:	2	ay-väen	neutraali	poutaa
æ	1	äveriäs	öljyjätin	pöytä
æi	1,6	äiti	äkkäiden	täi
æy	1,4	äyrin	täytyy	käy
æ:	1	ääriiryhmiä	häädetäään	päivää
e	4	enköhän	terve	me
ei	4,6	ei	vaihteita	hei
ø	1	öljyalan	ulkonäön	tiedänkö
øi	1,6	öisin	töitä	viittilöi
øy	1,4	öylätti	pöytä	
ø:	1	Öölanti	ulkoministeriöön	Bodø
eu	4,7	eurot	kyläseura	leu
ey	4,4	Eysturoy	keskeytyä	
e:	4	eesti	kyljelleen	aiheuttanee
i	6	iäkästä	viha	Berliini
ie	6,4	ientaskun	kieli	lie
iu	6,7		viulu	
iy	6,4		vihkiytynt	
i:	6	lida	siika	solmii
o	8	oksa	asuintaloja	spekulaatio
oi	8,6	oivia	koittaa	spekuloi
ou	8,7	outo	autokoulu	window

ipa	viseme	Example 1	Example 2	Example 3
o:	8	ok	koostaa	yo
u	7	ufoista	Bärlund	jätemaksu
u̩	7,6	ui	muita	epäonnistui
u̩	7,8	Uolevi	Suomi	Hilavuo
u:	7	url	innokkuus	kiikkuu
y	4	ydin	ökyrikas	kesy
yɸ	4,1	yö	työtä	järjestötyö
y̩	4,6	Yichangin	syitä	järjestäytyi
y:	4	yo	ryppy	iskeytyy

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	baareissa	Urban	Jakob
d	19	dementia	ladot	jugend
f	18	face	afgaani	Alf
g	20	gaalassa	fagotti	Aslös
h	12	ha	astumahan	Dietrich
j	6	jää	öljyä	Kaj
k	20	Kajaanin	epätarkat	idyllic
l	14	Lyytikäinen	euroseteiden	mail
m	21	mä	roimat	späm
n	19	nää	baanalle	iäkkään
ŋ	20		nähdäänkin	planning
p	21	pa	epäsuoria	backup
r	13	risteilyn	baari	Player

ipa	viseme	Example 1	Example 2	Example 3
s	15	sä	öljyisiä	Bärnäs
ʃ	16	Schauman	Bangladeshin	cash
t	19	tä	eurosta	epäsuorat
u	18	vaadi	innostava	Kiev

fr-FR/fr-CA/fr-CH

French suprasegmentals

The Speech service phone set puts stress after the vowel of the stressed syllable. However, the `fr-FR` Speech service phone set doesn't support the IPA substress '''. If the IPA substress is needed, you should use the IPA directly.

Vowels

sapi	ipa	viseme	Example 1	Example 2	Example 3
ae	a	2	arbre	patte	ira
af	a	2		pâte	pas
an	ã	2	enfant	enfant	temps
ax	ə	1		petite	le
eh	ɛ	4	elle	perdu	était
eu	ø	1	œufs	creuser	queue
ey	e	4	ému	crétin	ôté
in	ĩ	4	important	peinture	matin
iy	i	6	idée	petite	ami
oe	œ	4	œuf	peur	
oh	ɔ	3	obstacle	corps	
on	õ	3	onze	rondeur	bon
ow	o	8	auditeur	beaucoup	pô

sapi	ipa	viseme	Example 1	Example 2	Example 3
un	œ	4	un	lundi	brun
uw	u	7	outrage	introuvable	ou
uy	y	4	une	punir	élu

Consonant

sapi	ipa	viseme	Example 1	Example 2	Example 3
b	b	21	bête	habille	robe
d	d	19	dire	rondeur	chaude
f	f	18	femme	suffixe	bof
g	g	20	gauche	égale	baguette
gn	ɲ	19			peigne
hw	ɥ	7	huile	nuire	
k	k	20	carte	écaille	bec
l	l	14	long	élire	bal
m	m	21	madame	aimer	pomme
n	n	19	nous	tenir	bonne
ng	ŋ	20			parking
p	p	21	patte	repas	cap
r	ʁ	13	rat	chariot	sentir
s	s	15	sourir	assez	passee
sh	ʃ	16	chanter	machine	poche
t	t	19	tête	ôter	net
v	v	18	vent	inventer	réve
w	w	7	oui	fouine	
y	j	6	yod	piétiner	Marseille

sapi	ipa	viseme	Example 1	Example 2	Example 3
z	z	15	zéro	raisonner	rose
	n‿	19			un arbre
	t‿	19			quand
	z‿	15			corps

1 Only for some foreign words.

💡 Tip

The `fr-FR` Speech service phone set doesn't support the following French liaisons, `n‿`, `t‿`, and `z‿`. If they are needed, you should consider using the IPA directly.

ⓘ Note

`fr-CA`, `fr-CH` locales don't support SAPI phones now.

he-IL

Vowels

ipa	viseme	Example 1	Example 2	Example 3
i	6	איש	סִיר	כָּלִי
e	4	אֶש	כֵּל	פָּה
a	2	אָג	קָדִי	מָה
o	8	אוֹת	יְמִין	לֹא
u	7	עוֹגָה	כַּרְוב	הָגִיעָה

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	פּוֹ	קְלִיפָּה	טִינָג

ipa	viseme	Example 1	Example 2	Example 3
b	21	בּד	סְבָא	פָּאַב
t	19	טִיפה	מְתֻנָה	חֹוט
d	19	דְּבָר	אֲדוֹם	תְּמִיד
k	20	כְּתָב	בּוֹקֶר	חָלֵק
g	20	גָּדוֹל	אֲגָפָה	דָּג
?	19	אֲכִיב	שְׁיעָור	
f	18	פִּילֶטֶר	סּוּפָר	סּוּף
v	18	וִילָן	כְּבָד	לָבָד
s	15	שְׂמֵלָה	כְּסָפִי	פְּנָס
z	15	זָאָב	מְזָל	אֲגָז
ʃ	16	שְׁולָחָן	פְּשָׁוֹט	כְּבִישׁ
x	12	חַתּוֹל	אָוָל	פְּרָחָת
h	12	חוֹלָךְ	זָהָב	בָּהָר
tʂ	19,15	צָד	עַצְם	מוֹמְלָץ
m	21	מָאוֹד	סִימָן	חָלוּם
n	19	נְפָשָׁה	תִּינְוקָה	אֲבָן
l	14	לְשָׁוֹן	מִילָה	דָּגָל
ʁ	13	רָאָשָׁון	מָוָה	חִיבָר
j	6	יְלָךְ	מְצִיאָה	כְּדָאי
ʒ	16	צְ'אָנָר	מְצֹזָר	'בָּצָ
tʂ	19,16	צְ'יִיפָה	קְפֹצְ'הָן	'סְנְדוֹזִיָּץ
dʒ	19,16	גְּוָנָגָל	פְּיָגָ'הָה	'קוֹטָג

hr-HR

Vowels

ipa	viseme	Example 1	Example 2	Example 3
e	4	Egipat	Games	drveće
e:	4	eri	brijegom	de1taljne
i	6	ispada	želimo	javnosti
i:	6	iako	list	kompletни
u	7	ubacio	konkurentne	jednu
u:	7	Una	funta	Yu
a	2	američke	kovačić	kredita
a:	2	anđela	gradila	Divulja
o	8	oaza	nanosi	do
o:	8	Olgu	kiselog	to

Consonant

ipa	viseme	Example 1	Example 2	Example 3
d	19	dakle	evidenciji	kod
v	18	volja	građevinskog	imperativ
s	15	sabor	informisanja	interes
t	19	tad	informirati	ispit
n	19	na	žene	jeftin
l	14	logor	konstatirali	kapital
ʎ	14	ljudski	košulju	kralj
ts	19,15	carina	krivice	pravac
tʃ	19,16	četvorica	kritičan	osnivač
j	6	jednostavan	kuju	ovaj
x	12	hrvatskog	mahala	ovakvih
z	15	znanstvenom	mehanizacije	prijelaz

ipa	viseme	Example 1	Example 2	Example 3
ʒ	16	žalbu	mladeži	crtež
r	13	red	moraju	dar
k	20	kažu	nakani	dnevnik
m	21	Mađara	napadima	dobrim
p	21	Poljska	napadnut	kamp
g	20	gore	negativna	kaznenog
tš	16	ćelija	neisplaćene	mladić
f	18	fabula	nostrifikaciji	šef
b	21	Belgija	oba	sukob
ðʒ	19,16	džempera	llidže	George
nj	19	nje	emitiranja	stupanj
đ	16	đakovačkim	gađati	vođ
ʃ	16	šef	stišati	Glavaš

hu-HU

Vowels

ipa	viseme	Example 1	Example 2	Example 3
ø	1	ördög	ördög	huszonkettő
ø:	1	ők	önöműködően	öntöző
a	2		Schumacher	
a:	2	árut	bizonyára	burzsuá
ɛ	4	előzni	kisbéresnek	hogyne
e:	4	Édes	komédia	fölfelé
i	6	idegen	omnibuszok	kollégiumi
i:	6	ígyen	áhítat	

ipa	viseme	Example 1	Example 2	Example 3
o	8	oldali	komor	Figo
ø	2	atyját	olvasni	Olga
o:	8	ólmot	históriát	fénymásoló
u	7	ugyanis	ezután	falu
u:	7	úrrá	fékút	számú
y	4	üdítőt	átsütve	alsóbbrendű
y:	4	űrállomás	gépjárművek	idejű

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	busz	hiába	
b:	21		hasábbburgonya	rövidebb
d	19	dévédé	fapados	szaporítanád
ʒ	16	gyártó	franciaágynas	huszonegy
d:	19		Goddess	hadd
ʒ:	16		ebédjén	hagyj
ðʒ	19,16	Dzsó	menedzselési	college
ðʒ:	19,16		Baggio	
dz	19,15	dzémsz	edző	McDonalds
dz:	19,15		eddzeni	
f	18	figura	kofa	golf
f:	18		koffer	seriff
g	20	gondolom	számítógép	rúg
g:	20		faggatta	függ
h	12	hit	ruhában	ah

ipa	viseme	Example 1	Example 2	Example 3
h:	12		ehhez	
j	6	János	olyan	karéj
ɲ	19	nyakán	pecsenyét	példány
j:	6	Ljeszkovai	majmolják	állj
ɲ:	19		pihenjen	
k	20	kofa	ruhákat	ruhák
k:	20		megkondult	makk
l	14	lent	rúla	evvel
l:	14		kolléga	áll
m	21	magyar	számít	órám
m:	21		anyámmal	kilogramm
n	19	népies	szótlanul	hiszen
ŋ	20		angel	
n:	19		onnan	fenn
p	21	Pál	tapogatódzó	számítógép
p:	21		beröppent	befejezéseképp
r	13	rág	óraára	órakor
r:	13		amerre	forr
s	15	számára	fölveszi	fölmész
ʃ	16	saját	förtelmesen	főorvos
s:	15		halasszuk	hazajössz
ʃ:	16		házassága	keress
t	19	Tata	rútak	hit
c	16		egyházmegye	dirty
t:	19		hitte	vágodott

ipa	viseme	Example 1	Example 2	Example 3
c:	16		bátyja	Pretty
tʂ	19,15	címe	biciklis	huszonnyolc
tʃ	19,16	csigán	húgocskám	Gregorics
tʂ:	19,15		játszad	játsz
tʃ:	19,16		barátságos	futballmeccs
v	18	varr	Olívia	Dönöv
v:	18		evvel	
x	12	hrabovszki	ihletével	
ψ	20		alapján	Kapj
z	15	zúgást	csizmám	csimpánz
ʒ	16	zsûrivel	félmázsás	Balázs
ʐ:	15		húzza	fékezz
ʒ:	16		garázzsal	

id-ID

Vowels

ipa	viseme	Example 1	Example 2	Example 3
ə	1		benar	komite
a	2		babat	engga
ai	2,6	air	raikage	pantai
au	2,4	aurat	ataupun	pisau
e	4	energi	feri	tempe
ɛ	4	enrique	nenek	
I	6	indah	yakin	key
i	6	irama	biar	deflasi

ipa	viseme	Example 1	Example 2	Example 3
ɔ	3	off	esok	law
o	8	obat	bobot	domino
ɔɪ	3,6		reboisasi	sepoi-sepoi
u	7	umur	buah	linu
ʊ	4		duduk	

Consonant

ipa	viseme	Example 1	Example 2	Example 3
?	19	amores	maaf	pencak
b	21	babat	nebak	dishub
d	19	deder	pedang	cloud
ðʒ	19,16	jarum	penajah	buruj
f	18	flat	aversi	genitif
g	20	gabus	dagel	hamburg
h	12	hama	tuhan	entah
n	19	nyaman	menyuci	
j	6	yakin	jaya	baduy
k	20	ketan	aku	polsek
l	14	labu	talenta	vital
m	21	masuk	namanya	sinom
n	19	nada	sekunar	proton
ŋ	20	ngengat	kenanga	abang
p	21	pacar	hampa	cakap
r	13	rabu	dikurang	nasar
s	15	sabuk	tetesan	jenius

ipa	viseme	Example 1	Example 2	Example 3
ʃ	16	syarat	isyarat	british
t	19	tabir	adaptasi	durat
tʃ	19,16	cakap	dicari	
w	7	wajah	yuwana	
x	12	khusuk	akhirnya	barzakh
z	15	zakat	pezina	mahfuz

it-IT

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	amo	sano	scorta
ai	2,6	aics	abbaino	mai
au	2,7	audio	rauco	bau
e	4	eroico	venti	sapore
ɛ	4	elle	avvento	lacchè
ɛj	4,6	eira	email	lei
eu	4,7	euro	neuro	
ei	4,6		aseità	scultorei
eu	4,7	europeo	feudale	
i	6	italiano	vino	soli
u	7	unico	luna	zebù
o	8	obesità	straordinari	amico
ɔ	3	otto	botte	però
ɔj	3,6		oppiodi	
oi	8,6	oibò	intellettualoide	Gameboy

ipa	viseme	Example 1	Example 2	Example 3
ou	8,7		show	talkshow

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	bene	ebanista	Euroclub
b:	21		gobba	
tʃ	16	cenare	acido	french
tʃ:	19,16		braccio	
k:	20		pacco	Innsbruck
d	19	dente	adorare	interland
d:	19		cadde	
ɖ	15	zero	orzo	
ɖ:	15		mezzo	
f	18	fame	afa	alef
f:	18		beffa	bluff
ɸ	16	gente	agire	beige
ɸ:	16		oggi	
g	20	gara	alghe	smog
g:	20		fugga	Zuegg
ʎ	14	gli	ammiragli	
ʎ:	14		foglia	
j:	19		bagno	
ɲ	19	gnocco	padrigno	Montaigne
j	6	ieri	piede	freewifi
k	20	caro	anche	tic

ipa	viseme	Example 1	Example 2	Example 3
l	14	lana	alato	col
l:	14		colla	full
m	21	mano	amare	Adam
m:	21		grammo	
n	19	naso	lana	non
n:	19		panna	
p	21	pane	epico	stop
p:	21		coppa	
r	19	rana	motore	per
r:	13		carro	Starr
s	15	sano	cascata	lapis
s:	15		cassa	cordless
ʃ	16	scemo	Gramsci	slash
ʃ:	16		ascia	fiches
t	19	tana	eterno	alt
t:	19		zitto	
ts	15	tsunami	turbolenza	subtests
ts:	15		bozza	
v	18	vento	avarо	Asimov
v:	18		bewvi	
w	7	uovo	duomo	Marlowe
z	15	smodato	casa	elections

ja-JP

The Speech service phone set for ja-JP is based on the native phone Kana ↗ set.

Please see the following tables for kana and corresponding viseme in parentheses.

Katakana

Katakana	ア	イ	ウ	エ	オ
ア	ア (19,2)	イ (6,6)	ウ (7,6)	エ (19,4)	オ (19,8)
カ	カ (20,2)	キ (20,6)	ク (20,6)	ケ (20,4)	コ (20,8)
サ	サ (15,2)	シ (16,6)	ス (15,6)	セ (15,4)	ソ (15,8)
タ	タ (19,2)	チ (16,6)	ツ (19,15,6)	テ (19,4)	ト (19,8)
ナ	ナ (19,2)	ニ (19,6)	ヌ (19,6)	ネ (19,4)	ノ (19,8)
ハ	ハ (12,2)	ヒ (12,6)	フ (12,6)	ヘ (12,4)	ホ (12,8)
マ	マ (21,2)	ミ (21,6)	ム (21,6)	メ (21,4)	モ (21,8)
ヤ	ヤ (6,2)	n/a	ユ (6,6)	n/a	ヨ (6,8)
ラ	ラ (19,2)	リ (19,6)	ル (19,6)	レ (19,4)	ロ (19,8)
ワ	ワ (7,2)	n/a	n/a	n/a	ヲ (19,8)
	ン (19)	n/a	n/a	n/a	n/a

Katakana diacritics

Katakana diacritics	ア	イ	ウ	エ	オ
ガ	ガ (20,2)	ギ (20,6)	グ (20,6)	ゲ (20,4)	ゴ (20,8)
ザ	ザ (15,2)	ジ (16,6)	ズ (15,6)	ゼ (15,4)	ゾ (15,8)
ダ	ダ (19,2)	ヂ (16,6)	ヅ (15,6)	デ (19,4)	ド (19,8)
バ	バ (21,2)	ビ (21,6)	ブ (21,6)	ベ (21,4)	ボ (21,8)
パ	パ (21,2)	ピ (21,6)	プ (21,6)	ペ (21,4)	ポ (21,8)

Katakana Yōon

Katakana Yōon	ヤ	ュ	ョ
キ	キヤ(20,6,2)	キュ(20,6,6)	キョ(20,6,8)

Katakana Yōon	ヤ	ユ	ヨ
シ	シャ(16,6,2)	シュ(16,6,6)	ショ(16,6,8)
チ	チャ(16,6,2)	チュ(16,6,6)	チヨ(16,6,8)
ニ	ニヤ(19,6,2)	ニュ(19,6,6)	ニヨ(19,6,8)
ヒ	ヒヤ(12,6,2)	ヒュ(12,6,6)	ヒヨ(12,6,8)
ミ	ミヤ(21,6,2)	ミュ(21,6,6)	ミヨ(21,6,8)
リ	リヤ(19,6,2)	リュ(19,6,6)	リヨ(19,6,8)
ギ	ギヤ(20,6,2)	ギュ(20,6,6)	ギヨ(20,6,8)
ジ	ジヤ(16,6,2)	ジュ(16,6,6)	ジヨ(16,6,8)
ヂ	ヂヤ(16,6,2)	ヂュ(16,6,6)	ヂヨ(16,6,8)
ビ	ビヤ(21,6,2)	ビュ(21,6,6)	ビヨ(21,6,8)
ピ	ピヤ(21,6,2)	ピュ(21,6,6)	ピヨ(21,6,8)

Examples

Character	sapi	ipa
合成	ゴ'ウセ	go'wuseji
所有者	ショユ'ウ?ヤ	ɕyojus'wɯçja
最適化	サイテキカ+	sajitecika,

ko-KR

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	아가씨	강	하다
ɛ	4	애국가	백	세번째
e	4	에너지	가겠구나	가게
w	6	으레	으쓱해	가스

ipa	viseme	Example 1	Example 2	Example 3
i	6	이거	아직까지	어미
ʌ	1	어미	차선변경	가시겠어
o	8	오래	의혹을	차고
u	7	우간다	은둔자의	가시나무
ɯɪ̯	20,6	의자	배추흰나비가	가요계의
ɸ	1	외가댁	배치된다	하되
wa	7,2	와글와글	가치관과	가지와
wɛ	7,4	왜관	호쾌가	안돼
wɛ	7,4	웨딩드레스		
wi	7,6	위계적	구조위원회가	한가위
wʌ	7,1	워낙	가까워서	가까워
jɑ	6,2	야구	기술집약도가	끌려가야
jɛ	6,4	얘가	가수얘기예요	
jɛ	6,4	예감	적대관계가	의례
jʌ	6,1	여가	감정평가사	이리하여
jɔ	6,8	요구가	사용중지	가거든요
jʊ	6,7	유가적	경제교류가	소유

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	바가		밥
p	21	빠른가	막부정부가	
b	21		사범학교가	
tʃʰ	19,16	참가비는	아침과	
d	19	동네가		바깥

ipa	viseme	Example 1	Example 2	Example 3
t	19	따라가지	깍두기가	
d	19		산도가	
g	20	가조		각
k	20	까마귀가	젖가락으로	
g	20		단추가	
h	12		손가락질하거나	
h	12	하기가	손가락질하며	
ðz	19,16		손잡이가	
ðz	19,16	자유가		
tç	19,16	짜가면서	손가락질할	
k ^h	20	키가	아킬레스건	
l	14			국가체제를
m	21	마다가스카르	통나무가	기침
n	19	나가서는	아느냐	따라가다보면
ŋ	20		강아지	한강
p ^h	21	파티가	아파트	
r	19	라디오가	아름답게	
s ^h	15	사고가	아스팔트	
s	15	쌍둥이가	멕시코가	
t ^h	19	택시가	여타의	

ms-MY

Vowels

ipa	viseme	Example 1	Example 2	Example 3
i	6	ibu	iklim	ahli

ipa	viseme	Example 1	Example 2	Example 3
u	7	uang	buah	bahu
ə	1		kerja	nasionalisme
e	4	edar	aktres	kue
o	8	orang	anggota	pidato
a	2	anjing	anak	ada
ai	2,6			cerai
au	2,7	auto	akaun	bakau
oi	8,6			amboi

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	pekat	sekeping	cakap
b	21	banjir	lebih	jawab
t	19	tidak	peta	sikit
d	19	dekat	adakah	akad
k	20	ketat	akak	book
g	20	gabung	bogel	dialog
?	19	abang	kepercayaan	letak
tʃ	19,16	cepat	baca	
dʒ	19,16	jabatan	aja	kolej
m	21	memang	laman	malam
n	19	negeri	tanam	taman
ŋ	19	nyanyi	tanya	
ɳ	20		mangga	sayang
f	18	filem	artifak	aktif

ipa	viseme	Example 1	Example 2	Example 3
v	18	vaksin	aktiviti	
s	15	sahabat	akses	tumis
z	15	zaman	lazat	
ʃ	16	syarikat	bersyarat	
x	12	khabar	akhir	tarikh
r	13	racun	merah	lebar
h	12	hingga	aduhai	boleh
j	6	yang	ayah	
w	7	walau	bawah	
l	14	lidah	alam	katil

nb-NO

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	annonse	betrakte	hoppa
æ	1	ergre	Palermo	
æ:	1	ærlig	belære	bæ
a:	2	are	betale	bedra
ɛ	4	energi	kadetten	hoppe
ø:	1	øre	behøve	adjø
e:	4	ener	berede	distre
i	6	ikke	setningen	taxi
i:	6	Eagle	bevise	konditori
ɔ	3	åtte	kontrollen	altså
ø	4	ønske	belønning	Mossø

ipa	viseme	Example 1	Example 2	Example 3
o:	8	år	område	begå
u	7	okse	økokrim	ego
u:	7	ord	telefonen	bidro
y	7	ytterst	benytte	Ally
ø	6	under	forundret	jaggu
ø:	6	ule	umulig	intervju
y:	4	yte	belyse	paraply
æɪ	1,6	eiendom**	uttleide	snarvei
æø	1,6	aura	Litauen	fortau
ai	2,6	aibel**	Aserbajdsjan	Dubai
œy	4,7	øyer	ablegøyser	syltetøy
ɔy	3,7	Oilers	boikotten	konvoi
øɪ	6,6		Bruins	Mitsui

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	pil	ape	lapp
t	19	tall	matte	matt
k	20	kall	jakke	takk
b	21	bil	klubbe	lobb
d	19	dal	lide	gadd
g	20	gås	sagen	lag
f	18	fil	klaffe	klaff
h	12	hall	beholde	
s	15	sil	vise	viss

ipa	viseme	Example 1	Example 2	Example 3
ʂ	15	sju	maskin	dusj
ç	12	tjukk	bekjenne	Korch
v	18	vår	leve	lov
m	21	mil	komme	lam
n	19	nål	minnes	søvn
ŋ	20		penger	lang
l	14	løs	måle	tal
r	13	ris	karre	tørr
j	6	jag	utjevne	detalj
ɖ	19		burde	ferd
ɫ	14		farlig	jarl
ɳ	19		barnet	jern
t	19		skjorte	gjort

nl-NL/nl-BE

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	af	bak	bavarois
a:	2	aan	maal	ja
ã	2	enfin	manchet	croissant
œ̄	2,7	oud	bout	hou
ɛ	4	en	weg	hè
ɛ:	4	één	heet	nee
ɛ̄:	4	airbag	blèr	
ɛ̄i	4,6	eis	wijn	zij

ipa	viseme	Example 1	Example 2	Example 3
ɛ̄	4		lingerie	elektricien
ɸ̄:	1	euro	deur	milieu
ɪ̄	6	ik	ding	
ī	6	iets	sliep	drie
ɔ̄	3	op	slot	joh
ū	7	oefen	hoed	doe
ɔ̄:	3		roze	
ɔ̄̄	3			Macron
ō:	8	ook	boom	zo
ȳ	7	urn	dus	
ə̄	1	een	trommel	de
œ̄ȳ	4,4	uil	juist	bui
œ̄	4	œuvre	service	
ȳ	4	uur	tuur	nu

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b̄	21	boos	fobie	
d̄	19	dat	kudde	
f̄	18	fiets	gefeest	blaf
x̄	12	ga	magie	hoog
?̄	19		beam	
h̄	12	hoek	behaard	
ḡ	20	gujarati	again	drug
j̄	6	jij	boeien	haai

ipa	viseme	Example 1	Example 2	Example 3
k	20	kat	haken	zwak
l	14	land	familie	kool
m	21	man	demon	raam
n	19	niks	kannon	pan
ŋ	20		brengen	zing
p	21	poer	rapen	heb
r	13	romp	waarom	kier
s	15	soms	precies	heus
ʃ	16	sjaal	vaasje	lunch
t	19	tot	laten	groot
w	7		flauwe	follow
v	18	voor	haven	
u	18	wat	fusiewet	
z	15	zal	lezen	
ʒ	16	jus	beige	

pl-PL

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	autor	kolację	karasia
ɛ	4	elbląska	reflektor	osiągnięcie
ɛ̄	4		nieczęsto	skorupę
i	6	informatycznym	powiśle	gadali
ı	6		cudzymi	rodziły
ɔ	3	osobniki	uboju	rogatkɔ

ipa	viseme	Example 1	Example 2	Example 3
ĩ	3		mąż	intelektualistą
u	7	unosimy	arkuszy	przeznaczeniu

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	bliska	wyśrubowane	
b ^j	21	biuro	zapobiegać	
tç	19,16	ciągników	uczuciowych	suchość
tʂ	19,15	cząstkowe	odpoczynek	niszcz
c	16	kije	lisikiewicz	
d	19	drogowy	porodówkę	bastard
ɖ ^j	19	dializy	studiuję	
ðz	19,15	dzwonnica	wrożony	
ðʐ	19,16	dziurawe	pochodziłam	
f	18	wprost	długofalowy	konserwantów
f ^j	18	filmoteki	grafiką	
g	20	geometrią	nawigacja	
ʐ	16	gitarzysta	religijnym	
ðʐ	19,15	dżungle	menedżerskie	
k	20	królestwa	naukowo	matematyk
l	14	latający	populacje	handel
l ^j	14	lisek	okolica	
m	21	majątkowy	wytłumaczenia	błagam
m ^j	21	mieszkającej	dynamicznie	
n	19	napędu	poczynaniach	balkon

ipa	viseme	Example 1	Example 2	Example 3
ŋ	20		ciągłość	
n̪	19	niewidoczna	zmieniała	poznań
p̪	21	potoczne	terapeutyczne	odstęp
p̪j	21	pijawek	skupieniu	
r̪	13	regionu	operową	administrator
r̪j	13	ripostuje	imperialnej	
s̪	15	solone	przetasowania	biogaz
ç̪	16	sierpień	doniesieniem	mogłaś
ʃ̪	16	szanowanych	wpatrzeniu	skręcasz
t̪	19	talentom	kwaterze	dowód
t̪j	19	tirami	marketingiem	
ts̪	19,15	cyfrą	agencyjne	pałac
v̪	18	wysłaniu	przeprowadzają	
v̪j	18	winiarstwa	bukowianka	
w̪	7	łączenie	prałata	drukował
x̪	12	hamulcem	zdychają	kostiumach
x̪j	12	hiszpańscy	psychice	
j̪	6	jeździła	popijałam	najważniejszej
z̪	15	zaskakują	partyzanckich	wiz
z̪	16	ziemniaki	zgryzienia	
ʒ̪	16	żyrandol	nowożytnej	

pt-BR

Vowels

ipa	viseme	Example 1	Example 2	Example 3
-----	--------	-----------	-----------	-----------

ipa	viseme	Example 1	Example 2	Example 3
i	6	ilha	ficar	comi
ĩ	6	intacto	pintar	aberdeen
a	2	água	dada	má
ɔ	3	ora	porta	cipó
u	7	ufanista	mula	peru
ũ	7	uns	pungente	kuhn
o	8	ortopedista	fofo	avô
e	4	elefante	elefante	você
ẽ	4	anta	canta	amanhã
ə	1	aqui	amaciar	dada
ɛ	4	ela	serra	até
ẽ	4	endorfina	pender	
õ	8	ontologia	conto	

Consonant

ipa	viseme	Example 1	Example 2	Example 3
ĩ	7			atualizaçõo
w	7	washington	água	usou
p	21	pato	capital	
b	21	bola	cabeça	
t	19	tato	rato	
d	19	dado	amado	
g	20	gato	maragato	
m	21	mato	comer	
n	19	no	ano	

ipa	viseme	Example 1	Example 2	Example 3
ɲ	19	nhoque	ninho	
f	18	faca	afago	
v	18	vaca	cavar	
r	19		para	amar
s	15	satisfeito	amassado	casados
z	15	zebra	azar	
ʃ	16	cheirar	machado	
ʒ	16	jaca	injusta	
x	12	rota	carreta	
tʃ	19,16	tirar	atirar	
dʒ	19,16	dia	adiar	
l	14	lata	aleto	
ʎ	14	lhama	malhado	
ĩ	6		inabalavelmente	hífen
j	6		caixa	sai
k	20	casa	ensacado	

pt-PT

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	ábdito	consular	medirá
e	4	abacaxi	domação	longa
ej	4,6	eidético	direita	detectei
ẽ	4	anverso	viajante	afã
ẽj	4,6	angels	viagens	também

ipa	viseme	Example 1	Example 2	Example 3
ẽ̄w	4,7	hão	significaçãozinha	gabão
éw	4,7		saudar	hello
áj	2,6	airosa	culturais	vai
ɔ̄	3	hora	depósito	ló
ɔ̄j	3,6	óis	heróico	dói
áw	2,7	outlook	incauto	pau
ə̄	1	extremo	sapremar	noite
ē	4	eclipse	haver	buffet
ε̄	4	eco	hibérmios	paté
éw̄	4,7		pirinéus	escarcéu
ẽ̄	4	embaçado	dirimente	ámen
éw̄	4,7	eu	deus	bebêu
ī	6	igreja	aplaudido	escrevi
í̄	6	impaciente	espinçar	manequim
í̄w̄	6,7		niue	garantiu
ō	8	ofir	consumidor	stacatto
ój̄	8,6	oirar	noite	foi
ȭ	8	ombrão	barronda	dom
ȭj̄	8,6		ocupações	expõe
ū	7	ubi	facultativo	fado
új̄	7,6	uivar	arruivado	fui
ũ̄	7	umbilical	funcionar	fórum
ũ̄j̄	7,6		muito	

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	bacalhau	tabaco	club
d	19	dado	dado	band
r	19	rename	verás	chutar
f	18	fim	eficácia	golf
g	20	gadinho	apego	blog
j	6	iode	desassociado	substitui
k	20	kiwi	traficado	snack
l	14	laborar	pelada	full
ɫ	14		polvo	brasil
ʎ	14	lhancemente	antilhas	
m	21	maça	amanhã	modem
n	19	nutritivo	campana	scan
ɲ	19	nhambu-grande	toalhinha	penh
p	21	pai	crápula	laptop
R	13	recordar	guerra	chauffeur
s	15	seco	grosseira	boss
ʃ	16	chuva	duchar	médios
t	19	tabaco	pelota	input
v	18	vaca	combatível	pavlov
w	7	waffle	restituir	katofio
z	15	zâmbia	prazer	jazz
ʒ	16	gelada	infligir	cuj

ro-RO

Vowels

ipa	viseme	Example 1	Example 2	Example 3
ə	1	ăsta	abătut	fizică
ɨ	6	înspre	hotărâre	îhî
a	2	absolut	prematur	Praga
e	4	educație	tablete	alianțe
ea	4,2		studentească	badea
eo	4,8		bleumarin	vreo
i	6	Italia	aripi	aberații
o	8	oricum	catacombe	radio
oa	8,2	oară	închisoare	șamoa
u	7	umble	gradul	Alexandru

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	băi	vizibil	arab
b ^j	21			microbi
d	19	doctor	video	miliard
dʒ	19,16	ger	vegetație	
dʒ ^j	19,16			colegi
f	18	furtună	efort	ecograf
f ^j	18			filosofi
g	20	galeria	egal	filolog
g ^j	20		neghină	unghi
h	12	hexagon	arhitect	monarh
j	6	ieri	băiat	ditai
k	20	cadru	baricadat	pitic

ipa	viseme	Example 1	Example 2	Example 3
k ^j	20			urechi
l	14	laptop	alint	Paul
l ^j	14			circuli
m	21	mandarine	camera	atom
m ^j	21			consumi
n	19	nepot	Canada	Eden
ŋ	20		banca	plâng
n ^j	19			dragoni
p	21	pai	opera	Filip
p ^j	21			ocupi
r	13	real	mere	distribuitor
r ^j	13			palmieri
s	15	sertar	căsătorit	exclus
ʃ	16	şine	cuşetă	greş
ʃ ^j	16			groşi
t	19	teracota	material	abonament
t ^j	19			foştii
ts	19,15	tar	cuştit	vorbăreşti
tʃ	19,16	circa	meciuri	
ts ^j	19,15			uscaţi
tʃ ^j	19,16			indici
v	18	vaccin	gravidă	fugitiv
v ^j	18			nervi
w	7	uau	cuantificare	pliu
z	15	zoologică	frază	parbriz

ipa	viseme	Example 1	Example 2	Example 3
ʒ	16	jar	abajur	pasaj
z ^j	15			semnezi
ʒ ^j	16			dârji

ru-RU

Vowels

ipa	viseme	Example 1	Example 2	Example 3
а	2	адрес	радость	беда
ʌ	1	облаков	застенчивость	внучка
ə	1		яблочного	
ε	4	эпос	белка	кафе
и	6	иней	лист	соловьи
ɪ	6	игра	медведь	мгновенье
ɔ	6	энергия	лысый	весы
ɔ̄	3	окрик	мот	весло
ʊ	7	ужин	куст	пойду

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	профессор	поплавок	укроп
p ^j	21	Петербург	ослепительно	степь
b	21	большой	собака	
b ^j	21	белый	убедить	
t	19	тайна	старенький	твид
t ^j	19	тепло	учитель	синеть

ipa	viseme	Example 1	Example 2	Example 3
d	19	доверчиво	недалеко	
d ^j	19	дядя	единица	
k	20	крыло	кукуруза	кустарник
k ^j	20	кипяток	неяркий	
g	20	гроза	немного	
g ^j	20	герань	помогите	
x	12	хороший	поход	дух
x ^j	12	хилый	хихиканье	
f	18	фантазия	шкафах	кров
f ^j	18	фестиваль	кофе	верфь
v	18	внучка	синева	
v ^j	18	вертеть	свет	
s	15	сказочник	лесной	карапуз
s ^j	15	сеять	посередине	зажглись
z	15	заяц	звезда	
z ^j	15	земляника	созерцал	
ʂ	15	шуметь	пшено	мышь
ʐ	15	жилище	кружевной	
tʂ	19,15	целитель	Венеция	незнакомец
tç	19,16	часы	очарование	мяч
ç:	16	щелчок	ощущать	лещ
m	21	молодежь	несмотря	том
m ^j	21	меч	дымить	семь
n	19	начало	оконце	сон
n ^j	19	небо	линялый	тюлень

ipa	viseme	Example 1	Example 2	Example 3
l	14	лужа	долгожитель	мел
l ^j	14	лицо	недалеко	соль
r	13	радость	сорока	двор
r ^j	13	рябина	набережная	дверь
j	6	есть	маяк	игрушечный

sk-SK

Vowels

ipa	viseme	Example 1	Example 2	Example 3
i	6	idea	efektivita	éry
e	4	edícia	absencia	farme
a	2	abnormálne	eskapáda	ária
o	8	oba	banková	esperanto
u	7	udial	február	babu
ɛ	6		mäsité	
i:	6	ílu	edícií	druhý
e:	4	éra	anamnézy	ázijské
a:	2	áno	animácia	druhová
o:	8	ódy	bilión	haló
u:	7	úbočí	absolútna	druhú
ɪa	6,2		piatkové	maškrtia
ɪe	6,4		domnienka	námestie
ɪu	6,7			väčšiu
ʊo	7,8	ôsma	jahôd	malinô
au	2,7	audio	aplaudovalo	sredau

ipa	viseme	Example 1	Example 2	Example 3
ou	8,7			
ə	1			

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	pád	apelu	cap
b	21	babička	abiogenézy	dub
t	19	tabak	foto	art
d	19	delta	editor	backhand
c	16	ťahač	docenti	farnosť
ž	16	ďalej	strede	lod'
k	20	kabaret	bábkový	chudáčik
g	20	galón	demagógia	mozog
čs	19,15	certifikácie	cicavce	bác
ďz	19,15		medzí	
čž	19,16	čajový	frčí	boháč
ďž	19,16	džúsu	brandžé	
f	18	fabrík	biografie	fotograf
v	18	vokály	evanjelické	
s	15	sekunda	esá	algoritmus
z	15	zábal	fazuľu	aníz
ʃ	16	šabľa	duševná	chápeš
ʒ	16	žaba	veži	kaluž
x	12	charakter	biochémie	bohatých
h	12	háčik	bohmi	

ipa	viseme	Example 1	Example 2	Example 3
r	13	rabat	eróziou	éter
r̥	13		chatrče	leicester
r̥:	13		vŕtal	
l̥	14	lampa	električka	čakal
l̥:	14		dlhý	nóbl
l̥:	14		jabík	
ʎ	14	ľad	citeľné	byľ
m̥	21	meter	emisný	aktom
m̥	21		amfiteáter	
n̥	19	nábeh	colnému	fajn̥
N̥	19		slovinský	
ŋ̥	20		banket	
n̥	19	ňom	ani	jačmeň
u̥	7		cestovní	aktív
i̥	6		fajka	chatovej
j̥	6	ja	eseje	
w̥	7	vzbudí	krivdí	

sl-SI

Vowels

ipa	viseme	Example 1	Example 2	Example 3
ə	1	rjava	pisemski	december
a	2	azilantov	hišam	delniška
a:	2	avto	marketplace	lucia
ɛ	4	edicija	priděm	ničle

ipa	viseme	Example 1	Example 2	Example 3
e:	4	ebola	pridevnik	prepove
ɛ:	4	ena	producenta	janže
i	6	ideja	poudarim	poudariti
i:	6	igla	ilirska	jedmi
ɔ	3	oba	morfološke	Marko
ɔ:	3	oče	črnomorskem	
o	8	občina	reformam	seno
u	7	ulova	mamut	mandatu
u:	7	ura	dramaturgom	intervju

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	brez	bober	lebdeča
d	19	dajal	degradacija	navedbah
d ¹	19	dleto		navedla
dn	19,19	dna		dohodne
đž	19,16	džezovske	bridža	menedžment
đz	19,15	odživajo		Kocbek
f	18	fagota	fotografa	golf
ŋ	21			nimfa
ɣ	20			aħdeloja
g	20	gaber	lagal	ragbi
ħ	6		pojdi	emisij
j	6	jadra	kajak	najostreje
k	20	kabel	akademija	alkoholik

ipa	viseme	Example 1	Example 2	Example 3
l	14	labirint	olajša	šal
l ^j	14			poljski
m	21	maček	omara	potem
ŋ	20	Ngaliemski_slapovi		banka
n	19	nabave	obarvana	obarvan
n ^j	19			konjska
p	21	pada	sapa	sestop
r	13	rabilia	sorazmerna	spor
s	15	srajca	virusa	virus
ʃ	16	šah	sušijo	tovariš
t	19	tabela	gojiti	flavtist
t ^l	19	tla	škatla	desetletne
tn	19,19			devetnajst
tʃ	19,16	čaj	gneča	hlač
tʂ	19,15	car	raca	rejec
u	7			avto
v	18	veja	avanture	
w	7	vgradila		slabokrvnost
M	7	vstatí		
x	12	hiša	jahači	jamah
z	16	žaba	ježa	možgani
z̥	15	zmaj	doza	izvoza

sv-SE

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	andas	betrakta	gryta
æ	1	ärter	smärtar	
æ:	1	ärliga	besvära	
a:	2	avig	beta	bedra
ɔ	3	åtta	kontrollen	jaså
a_u	2,7	aura	pauser	Olau
ə	1		äpplena	pojke
e	4	energi	servetten	Arjepluovve
ɛ	4	äpple	berätta	Siviä
ɛ:	4	äta	beträda	trä
e:	4	eka	konkreta	café
œ	8	ört	störta	
œ:	4	ören	beröra	
ø	4	öppen	Alströmer	Päiviö
ɸ:	1	öl	belöning	adjö
ɪ	6	idé	vitsippa	kiwi
i:	6	ivrig	kris	parti
ʊ	4	oas	betrodda	konto
u:	7	oro	förtroende	bero
ø:	8	åtala	telefonen	nivå
ə	1	uppenbar	förundrad	farstu
ʉ:	6	ute	bestulen	intervju
y	4	ytterst	rykte	Tommy
y:	4	yta	förtydliga	paraply

Consonant

ipa	viseme	Example 1	Example 2	Example 3
p	21	pil	apa	topp
t	19	tal	matta	akut
k	20	kål	jacka	tak
b	21	bil	klubba	jobb
d	19	dal	lida	stad
g	20	gås	såga	lag
f	18	fil	klaffa	klaff
h	12	hal	behålla	
s	15	sil	visa	viss
ɧ	16	sjuk	maskin	dusch
ç	16	tjock	åkkänslan	coach
v	18	vår	leva	torv
m	21	mil	kamma	arm
n	19	nål	minnas	sömn
ŋ	20		ringa	ung
l	14	lös	måla	tal
r	13	ris	kärra	borr
j	6	jag	tröja	haj
ɖ	19		borda	bord
ɫ	14		porlande	kärl
ɳ	19		gärna	barn
ʂ	15		forsa	fors
t̪	19		skjorta	gjort

th-TH

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2		ນະ	
a:	2		ຫາດ	ໜາ
e	4		ເຕະ	
e:	4		ເທ	ເຕັ
i	6		ປິດ	
i:	6		ປຶກ	ປີ
ia	6,2		ເງື່ອນ	
o	8		ບດ	
o:	8		ແລກໂຕສ	ຈາວໂລ່
ə	1		ໃບຝາກເງິນ	
ə:	1		ເກີນ	ເໜຸ່ອ
u	7		ຈຸດ	
u:	7		ຊູດ	ຊີ
ua	7,2		ກວນ	ຮ້າ
ɯ	6		ຢືດ	
ɯ:	6		ມືດ	ມືອ
wa	6,2		ເຮືອນ	ເຮືອ
ɛ	4		ຄຸ່ແຂ່ງ	
ɛ:	4		ແບນ	ຄ່າແປ່ລ
ɔ	3		ນົອຕ	
ɔ:	3		ນອນ	ເຈັບຄອ

Consonant

ipa	viseme	Example 1	Example 2	Example 3
-----	--------	-----------	-----------	-----------

ipa	viseme	Example 1	Example 2	Example 3
b	21	บิน	กระบี่	ยมน
tč	19,16	เจ็ด	กระโจน	
tčʰ	19,16	เช้าๆ	วันอาสาพหุชา	
d	19	เดช	วิดีโอ	
f	18	ฟ้า	เพื่องฟู	
h	12	หา	เศษอาหาร	
j	6	ยา	เรือยนต์	ยาย
k	20	กາ	เนื้อไก่	มาก
kʰ	20	คາ	เลข	
l	14	ลາ	หุ่นลาม	
m	21	มา	เสมอ	รัดกุม
n	19	นา	แอมโนเนีย	กาล
ŋ	20	งາ	แต่งงาน	แหล่ง
p	21	ปາ	โดยทั่วไปแล้ว	กับ
pʰ	21	พາ	ล็อกกี้	
r	13	รา	ไข่แมลาเรีย	
s	15	ສາມ	ไม่มีสติ	
t	19	ຕາ	กติกາ	อาจ
tʰ	19	ທາ	คุณธรรม	
w	7	ຈົ່ງ	ກວິ	ແກ້ວ
?!	19	ອາ	ສະອາດ	

Tone

ipa	Description	Example
-	Mid	ໄປ bpaɪ ^M (to go)

ipa	Description	Example
˥	Low	ໄຂ້ khai ^L (egg)
˧	Falling	ໃຈ່ chai ^F (yes; agreement)
˦	High	ຄຮັບ khrap ^H ([spoken by a male] yes)
˥	Rising	ໜັງ nang ^R (cinema film)

tr-TR

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2	armut	fal	elma
a:	2	ağrı	kağıńı	dağ
e	4	erik	kel	akide
e:	4	eğri	değnek	yeğ
œ	4	ördek	göl	banliyö
œ̄	4,16	öğlen	açıköğretim	
i	6	ilaç	kıl	kedi
ī	6,16	iğne	mevkiiñe	tebliğ
o	8	orman	kol	vazo
ō	8,16	oğlan	doğru	doğ
u	7	uçak	kuş	koku
ū	7,16	uçur	buğra	başbuğ
ɯ	6	ıhlamur	tıp	kazı
ɯ̄	6,16	ığdır	sığlık	tiğ
y	4	ülke	gül	öykü
ȳ	4,16	düğme		

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	balık	ebe	sertab
c	16	keçi	ekşi	böyük
tʃ	19,16	çöp	uçak	ilgeç
d	19	dere	badem	ad
f	18	fil	kefen	çarşaf
g	20	gaz	yorgun	diyalog
ɣ	20		ağır	
ʒ	16	gavur	kevgir	
h	12	halat	ahır	külah
j	6	yer	ayva	olay
dʒ	19,16	cezve	evcimen	hac
k	20	kabak	bakla	çocuk
l	14	leylek	delik	kıl
ɫ	6	lala	kalın	pul
m	21	muz	yemek	kalem
n	19	nar	inek	sorun
ŋ	20		mangal	ring
p	21	para	kapı	rakip
r	19	renk	iri	minder
s	15	sal	kısa	bahis
ʃ	16	şekil	koşu	afiş
t	19	terlik	kutup	adalet
v	18	vadi	tava	ev
w	7		tavuk	

ipa	viseme	Example 1	Example 2	Example 3
z	15	zemin	gezi	kaz
ʒ	16	jöle	angajman	refüp

vi-VN

Vowels

ipa	viseme	Example 1	Example 2	Example 3
a	2		ban	ma
ɛ	4		hép	mẹ
i	6			chí
ɔ	3		học	tò
u	7		hung	thù
ua	7,2			chúa
aj	2,6			tài
ɛj	4,6			lãy
əj	1,6			chơi
o	8			xô
œw	6,4,7			riêú
ɛθ	6,1		khướt	
ɔi	3,6			mọi
ə	1		phần	
ie	6,4		biển	
uj	7,6			mùi
aw	2,7			bảo
ɨ	6		chứng	từ
e	4		nhặt	

ipa	viseme	Example 1	Example 2	Example 3
ăw	2,7			màu
ăj	2,6			ngày
ɛəj	6,1,6			lưới
o̪j	8,6			hồi
ə:	1		lợn	sõ
e	4		sên	té
ɔ̄w	3,7			tấu
ɛw	4,7			béo
ɪw	6,7			tịu
ɛ̄w	6,7			tựu
ēj	4,6			bêu
ɛ̄w̄	6,1,7			bươú
ɛ̄j	6,6			cửi
i	6	inh	tinh	
iə	6,1			kìa
āj	2	ach	hóach	

Consonant

ipa	viseme	Example 1	Example 2	Example 3
b	21	ba		
k	20	canh		diệc
z	15	diễn		
j	6	giặc		
ɹ	13	róc		
f	18	phụ		

ipa	viseme	Example 1	Example 2	Example 3
ɣ	20	gà		
h	12	hoa		
l	14	làm		
m	21	mười		năm
n	19	núi		tiền
p	21	pín		pháp
s	15	xào		
ʂ	15	son		
t	19	tuổi		hết
v	18	vàng		
ɖ	19	đón		
ɳ	20	ngủn		lung
x	12	khùng		
ɲ	19	nhàm		sanh
tʰ	19	thông		
t̚	19	trùng		
tʃ	19,16	chim		chỉ
w	7		hoang	

Tone

ipa	Description	Example
˥	tone ngang	xem
˧	tone huyền	làm
˨	tone sắc	sống
˩	tone hỏi	phải

ipa	Description	Example
ጀ	tone ngā	cũng
ጀጀ	tone nặng	một

zh-CN

The Speech service phone set for zh-CN is based on the native phone [Pinyin](#) ↗ set.

Pinyin Initials

Pinyin	viseme	Character example	sapi example
b	21	玻	bo 1
p	21	坡	po 1
m	21	摸	mo 1
f	18	佛	fo 2
d	19	得	de 2
t	19	特	te 4
n	19	呢	ne 5
l	14	乐	le 4
g	20	哥	ge 1
k	20	科	ke 1
h	12	喝	he 1
j	16	基	ji 1
q	16	欺	qi 1
x	16	希	xi 1
zh	19, 15	知	zhi 1
ch	19, 15	吃	chi 1
sh	15	诗	shi 1
r	15	日	ri 4

Pinyin	viseme	Character example	sapi example
z	15	资	zi 1
c	15	此	ci 3
s	15	思	si 1
y	6	衣	yi 1
w	7	屋	wu 1

Pinyin Finals

Pinyin	viseme	Character example	sapi example
a	2, 2 (19, 2, 2 for no initials)	法	fa 3
o	7, 8, 8 (19, 8, 8 for no initials)	泼	po 1
e	1, 1 (19, 1, 1 for no initials)	歌	ge 1
i	6, 6, 6	理	li 3
u	7, 7, 7	步	bu 4
v	7, 4, 4	女	nv 3
ai	2, 4 (19, 2, 4 for no initials)	百	bai 3
ei	4, 6 (19, 4, 6 for no initials)	北	bei 3
ui	7, 4, 6	对	dui 4
ao	2, 8 (19, 2, 8 for no initials)	号	hao 4
ou	8, 7 (19, 8, 7 for no initials)	走	zou 3
iu	6, 8, 7	牛	niu 2
ie	6, 4, 4	谢	xie 4
ue	7, 4, 4	略	lue 4
er	19, 1, 1	耳	er 3
an	2, 19 (19, 2, 19 for no initials)	喊	han 3
en	1, 19 (19, 1, 19 for no initials)	肯	ken 3
in	6, 6, 19	宾	bin 1

Pinyin	viseme	Character example	sapi example
un	7, 1, 19	尊	zun 1
ang	2, 20 (19, 2, 20 for no initials)	朗	lang 3
eng	1, 20	恒	heng 2
ing	6, 1, 20	赢	ying 2
ong	7, 7, 20	红	hong 2
ia	6, 2, 2	家	jia 1
ian	6, 2, 19	面	mian 4
iang	6, 2, 20	象	xiang 4
iao	6, 2, 8	表	biao 3
iong	7, 7, 20	兄	xiong 1
ua	7, 2, 2	花	hua 1
uai	7, 2, 4	帅	shuai 4
uan	7, 2, 19	短	duan 3
uang	7, 2, 20	广	guang 3
uo	7, 8, 8	多	duo 1

Pinyin Whole syllable

Pinyin	viseme	Character example	sapi example
zhi	19, 15, 6, 6	知	zhi 1
chi	19, 15, 6, 6	吃	chi 1
shi	15, 6, 6	诗	shi 1
ri	15, 6, 6	日	ri 1
zi	15, 6, 6	资	zi 1
ci	15, 6, 6	词	ci 2
si	15, 6, 6	斯	si 1
yi	6, 6, 6	衣	yi 1

Pinyin	viseme	Character example	sapi example
wu	7, 7, 7	屋	wu 1
yu	7, 4, 4	鱼	yu 2
ye	6, 4, 4	叶	ye 4
yue	7, 4, 4	月	yue 4
yuan	7, 2, 19	圆	yuan 2
yin	6, 6, 19	音	yin 1
yun	7, 1, 19	云	yun 1
ying	6, 1, 20	英	ying 1
a	19, 2, 2	啊	a 1
o	19, 8, 8	噢	o 1
e	19, 1, 1	鹅	e 2
ai	19, 2, 4	爱	ai 4
ei	19, 4, 6	欸	ei 1
ao	19, 2, 8	奥	ao 4
ou	19, 8, 7	偶	ou 3
an	19, 2, 19	安	an 1
en	19, 1, 19	恩	en 1
ang	19, 2, 20	昂	ang 2

Pinyin Tone

Pinyin	Character example	sapi example
bā	八	ba 1
bá	拔	ba 2
bǎ	把	ba 3
bà	坝	ba 4
ba	吧	ba 5

Example

Character	Speech service
组织关系	zu 3 - zhi 1 - guan 1 - xi 5
累进	lei 3 - jin 4
西宅巷	xi 1 - zhai 2 - xiang 4
一会儿	yi 2 - hui r 4

zh-HK

The Speech service phone set for zh-HK is based on the native phone [Jyutping](#) ↗ set.

Jyutping Initials

Jyutping	Character example	sapi example	VisemeID
p	怕	paa 3	21
b	巴	baa 1	21
t	他	taa 1	19
d	打	daa 2	19
k	卡	kaa 1	20
g	家	gaa 1	20
f	花	faa 1	18
s	沙	saa 1	15
h	蝦	haa 1	12
m	媽	maa 1	21
n	那	naa 5	19
ng	牙	ngaa 4	20
c	叉	caa 1	19 15
z	渣	zaa 1	19 15
l	啦	laa 1	14

Jyutping	Character example	sapi example	VisemeID
kw	誇	kwa 1	20
gw	瓜	gwaa 1	20
w	蛙	waa 1	7
j	甘	jaa 6	6

Jyutping Middle

Jyutping	Character example	sapi example	VisemeID
aa	沙	saa 1	2
e	些	se 1	4
i	詩	si 1	6
o	疏	so 1	3 In [ou], [o] corresponding viseme is 8
u	夫	fu 1	7
oe	鋸	goe 3	4
yu	書	syu 1	4
a	新	san 1	4
eo	律	leot 6	1

Jyutping Ending

Jyutping	Character example	sapi example	VisemeID
i	西	sai 1	6 (In [eo], [i] corresponding viseme is 4)
u	收	sau 1	7
p	夾	gep 6	21
t	不	bat 1	19
k	策	caak 3	20
m	心	sam 1	21
n	新	san 1	19

Jyutping	Character example	sapi example	VisemeID
ng	敬	ging 3	20

Jyutping Tone

Tone number	Description	Character example	sapi example
1	High level/High falling or Entering High Level	詩	si 1
2	Mid Rising	史	si 2
3	Mid Level or Entering Mid Level	試	si 3
4	Low Falling	時	si 4
5	Low Rising	市	si 5
6	Low Level or Entering Low Level	是	si 6

zh-TW

The Speech service phone set for zh-TW is based on the native phone [Bopomofo ↗](#) set.

Bopomofo Initials

Bopomofo	Pinyin	sapi Example (Bopomofo, Pinyin)
ㄅ	b	玻 (ㄅㄞ, bo 1)
ㄆ	p	坡 (ㄆㄞ, po 1)
ㄇ	m	摸 (ㄇㄞ, mo 1)
ㄈ	f	佛 (ㄈㄞ, fo 2)
ㄉ	d	得 (ㄉㄞ, de 2)
ㄊ	t	特 (ㄊㄞ, te 4)
ㄋ	n	呢 (ㄋㄞ, ne 5)
ㄌ	l	樂 (ㄌㄞ, le 4)

Bopomofo	Pinyin	sapi Example (Bopomofo, Pinyin)
ㄍ	g	哥 (ㄍㄜ, ge 1)
ㄎ	k	科 (ㄎㄜ, ke 1)
ㄏ	h	喝 (ㄏㄢ, he 1)
ㄐ	j	基 (ㄐㄧ, ji 1)
ㄑ	q	欺 (ㄑㄧ, qi 1)
ㄒ	x	希 (ㄒㄧ, xi 1)
ㄓ	zh	知 (ㄓㄧ, zhi 1)
ㄔ	ch	吃 (ㄔㄧ, chi 1)
ㄕ	sh	詩 (ㄕㄧ, shi 1)
ㄖ	r	日 (ㄖㄧ, ri 4)
ㄗ	z	资 (ㄗㄧ, zi 1)
ㄘ	c	此 (ㄘㄧ, ci 3)
ㄙ	s	思 (ㄙㄧ, si 1)
ㄧ	y	衣 (ㄧㄧ, yi 1)
ㄨ	w	屋 (ㄨㄩ, wu 1)

Bopomofo Finals

Bopomofo	Pinyin	sapi Example (Bopomofo, Pinyin)
ㄚ	a	法 (ㄅㄚ, fa 3)
ㄛ	o	潑 (ㄉㄛ, po 1)
ㄜ	e	歌 (ㄍㄜ, ge 1)
ㄧ	i	理 (ㄌㄧˋ, li 3)
ㄨ	u	步 (ㄉㄨˋ, bu 4)
ㄩ	v	女 (ㄉㄩˊ, nv 3)
ㄞ	ai	百 (ㄉㄞˇ, bai 3)

Bopomofo	Pinyin	sapi Example (Bopomofo, Pinyin)
ㄞ	ei	北(ㄞㄞ, bei 3)
ㄨㄞ	ui	對(ㄨㄞㄞ, dui 4)
ㄠ	ao	號(ㄏㄠㄠ, hao 4)
ㄡ	ou	走(ㄡㄡㄉ, zou 3)
ㄧㄡ	iu	牛(ㄐㄧㄡㄉ, niu 2)
ㄧㄢ	ie	謝(ㄒㄧㄢㄉ, xie 4)
ㄩㄢ	ue	略(ㄌㄩㄢㄉ, lue 4)
ㄢ	an	喊(ㄏㄢㄉ, han 3)
ㄣ	en	肯(ㄻㄣㄉ, ken 3)
ㄧㄣ	in	賓(ㄦㄧㄣ, bin 1)
ㄨㄣ	un	尊(ㄩㄨㄣ, zun 1)
ㄤ	ang	朗(ㄌㄤㄉ, lang 3)
ㄥ	eng	恆(ㄏㄥㄉ, heng 2)
ㄧㄥ	ing	贏(ㄧㄥㄉ, ying 2)
ㄨㄥ	ong	紅(ㄏㄨㄥㄉ, hong 2)
ㄧㄚ	ia	家(ㄩㄧㄚ, jia 1)
ㄧㄢ	ian	麵(ㄇㄧㄢ, mian 4)
ㄧㄤ	iang	象(ㄒㄧㄤㄉ, xiang 4)
ㄧㄾ	iao	表(ㄦㄧㄾ, biao 3)
ㄩㄾ	iong	兄(ㄒㄩㄾ, xiong 1)
ㄨㄚ	ua	花(ㄏㄨㄚ, hua 1)
ㄨㄞ	uai	帥(ㄩㄨㄞ, shuai 4)
ㄨㄢ	uan	短(ㄩㄨㄢ, duan 3)
ㄨㄤ	uang	廣(ㄍㄨㄤ, guang 3)
ㄨㄛ	uo	多(ㄩㄨㄛ, duo 1)

Bopomofo Whole syllable

Bopomofo	Pinyin	sapi Example (Bopomofo, Pinyin)
ㄓ	zhi	知 (ㄓ, zhi 1)
ㄔ	chi	吃 (ㄔ, chi 1)
ㄕ	shi	诗 (ㄕ, shi 1)
ㄖ	ri	日 (ㄖ, ri 1)
ㄗ	zi	资 (ㄗ, zi 1)
ㄎ	ci	词 (ㄎ, ci 2)
ㄘ	si	斯 (ㄘ, si 1)
ㄧ	yi	衣 (ㄧ, yi 1)
ㄨㄛ	wo	我 (ㄨㄛ, wo 3)
ㄨ	wu	屋 (ㄨ, wu 1)
ㄩ	yu	鱼 (ㄩ, yu 2)
ㄧㄢ	ye	叶 (ㄧㄢ, ye 4)
ㄩㄢ	yue	月 (ㄩㄢ, yue 4)
ㄦ	er	耳 (ㄦ, er 3)
ㄩㄤ	yuan	圆 (ㄩㄤ, yuan 2)
ㄧㄣ	yin	音 (ㄧㄣ, yin 1)
ㄩㄣ	yun	云 (ㄩㄣ, yun 1)
ㄧㄥ	ying	英 (ㄧㄥ, ying 1)
ㄩㄥ	yong	擁 (ㄩㄥ, yong 1)
ㄚ	a	啊 (ㄚ, a 1)
ㄛ	o	噢 (ㄛ, o 1)
ㄜ	e	鹅 (ㄜ, e 2)
ㄞ	ai	爱 (ㄞ, ai 4)
ㄟ	ei	欸 (ㄟ, ei 1)

Bopomofo	Pinyin	sapi Example (Bopomofo, Pinyin)
ㄠ	ao	奥 (ㄠ, ao 4)
ㄡ	ou	偶 (ㄡ, ou 3)
ㄢ	an	安 (ㄢ, an 1)
ㄣ	en	恩 (ㄣ, en 1)
ㄤ	ang	昂 (ㄤ, ang 2)
ㄥ	eng	鞞 (ㄥ, eng 1)
ㄦ	ê	ㄦ (ㄦ, ê 1)

Bopomofo tone

Bopomofo	Tone number	sapi Example (Bopomofo, Pinyin)
-	1	八 (ㄩㄚ or ㄩㄚˉ, ba 1)
ˊ	2	拔 (ㄩㄚˊ, ba 2)
ˇ	3	把 (ㄩㄚˇ, ba 3)
ˋ	4	坝 (ㄩㄚˋ, ba 4)
˙	5	吧 (ㄩㄚ˙, ba 5)

Map X-SAMPA to IPA

The table below shows a mapping relationship between X-SAMPA (Extended Speech Assessment Methods Phonetic Alphabet) and IPA alphabets. The X-SAMPA symbols are shown at left, with the corresponding IPA symbols to the right.

txt	
x-sampa (L)	ipa (R)
a	a
b	b
b_<	þ
c	c
d	d
d`	ð
d_<	ɸ

e	e
f	f
g	g
g_<	g̊
h	h
h\	h̄
i	i
j	j
j\	j̄
k	k
l	l
l`	l̄
l\	l̄
m	m
n	n
n`	n̄
o	o
p	p
p\	p̄
q	q
r	r
r`	r̄
r\`	r̄̄
r\`^	r̄̄̄
s	s
s`	s̄
s\`	s̄̄
t	t
t`	t̄
u	u
v	v
P	ø
v\`	ø̄
w	w
x	x
x\`	x̄
y	y
z	z
z`	z̄
z\`	z̄̄
A	ɑ
B	β
B\`	β̄
C	ç
D	ð
E	ɛ
F	ɱ
G	ɣ
G\`	ɣ̄
G\`_<	ɣ̄̄
H	ɥ
H\`	ɥ̄
I	ɪ
I\`	ɛ
J	ɲ

J\	ጀ
J_<	ጀጀ
K	ጀጀ
K\	ጀጀ
L	ጀጀ
L\	ጀጀ
M	ጀጀ
M\	ጀጀ
N	ጀጀ
N\	ጀጀ
O	ጀጀ
O\	ጀጀ
Q	ጀጀ
R	ጀጀ
R\	ጀጀ
S	ጀጀ
T	ጀጀ
U	ጀጀ
U\	ጀጀ
V	ጀጀ
W	ጀጀ
X	ጀጀ
X\	ጀጀ
Y	ጀጀ
Z	ጀጀ
.	ጀጀ
"	ጀጀ
%	ጀጀ
_j	ጀጀ
'	ጀጀ
:	ጀጀ
:\'	ጀጀ
@	ጀጀ
@\`	ጀጀ
@`	ጀጀ
{	ጀጀ
}	ጀጀ
1	ጀጀ
2	ጀጀ
3	ጀጀ
3\	ጀጀ
4	ጀጀ
5	ጀጀ
6	ጀጀ
7	ጀጀ
8	ጀጀ
9	ጀጀ
&	ጀጀ
?	ጀጀ
?\'	ጀጀ
<\`	ጀጀ
>\`	ጀጀ
^	ጀጀ
!	ጀጀ
!\`	ጀጀ

_	_ \	_ =	_ \ \	_ = \	_ - \	_ "	_ +	_ -	_ /	_ 0	_ =	_ =	_ >	_ ? \	_ \	_ ^	_ }	_ `	_ ~	_ ~ A	_ a	_ B	_ B_L	_ c	_ d	_ e	<F>	_ F	_ G	_ H	_ H_T	_ h	_ k	_ L	_ l	_ M	_ m	_ N	_ n	_ O	_ o	_ q	<R>	_ R	_ R_F	_ r	_ T	_ t	_ v	_ w	_ X	_ x
-	- -	- =	- \ :	- =	- -	- "	- +	- -	- /	- 0	- =	- =	- >	- ? :	- \ ^	- } ,	- ` ~	- ~	- ~ -	- a	- B	- B_L	- c	- d	- e	- >	- F	- G	- H	- H_T	- h	- k	- L	- l	- M	- m	- N	- n	- O	- o	- q	- >	- R	- R_F	- r	- T	- t	- v	- w	- X	- x	

Additional resources

Documentation

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Voice and sound with Speech Synthesis Markup Language \(SSML\) - Speech service - Azure Cognitive Services](#)

Learn about Speech Synthesis Markup Language (SSML) elements to determine what your output audio will sound like.

[Speech Synthesis Markup Language \(SSML\) document structure and events - Speech service - Azure Cognitive Services](#)

Learn about the Speech Synthesis Markup Language (SSML) document structure.

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Troubleshoot the Speech SDK - Speech service - Azure Cognitive Services](#)

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Show 5 more](#)

Lower speech synthesis latency using Speech SDK

Article • 04/08/2022 • 7 minutes to read

The synthesis latency is critical to your applications. In this article, we will introduce the best practices to lower the latency and bring the best performance to your end users.

Normally, we measure the latency by `first byte latency` and `finish latency`, as follows:

Latency	Description	SpeechSynthesisResult property key
first byte latency	Indicates the time delay between the start of the synthesis task and receipt of the first chunk of audio data.	SpeechServiceResponse_SynthesisFirstByteLatencyMs
finish latency	Indicates the time delay between the start of the synthesis task and the receipt of the whole synthesized audio data.	SpeechServiceResponse_SynthesisFinishLatencyMs

The Speech SDK puts the latency durations in the Properties collection of [SpeechSynthesisResult](#). The following sample code shows these values.

C#

```
var result = await synthesizer.SpeakTextAsync(text);
Console.WriteLine($"first byte latency:
\{result.Properties.GetProperty(PropertyId.SpeechServiceResponse_SynthesisFirstByteLatencyMs)} ms");
Console.WriteLine($"finish latency:
\{result.Properties.GetProperty(PropertyId.SpeechServiceResponse_SynthesisFinishLatencyMs)} ms");
// you can also get the result id, and send to us when you need help for diagnosis
var resultId = result.ResultId;
```

The first byte latency is much lower than finish latency in most cases. The first byte latency is independent from text length, while finish latency increases with text length.

Ideally, we want to minimize the user-experienced latency (the latency before user hears the sound) to one network route trip time plus the first audio chunk latency of the speech synthesis service.

Streaming

Streaming is critical to lowering latency. Client code can start playback when the first audio chunk is received. In a service scenario, you can forward the audio chunks immediately to your clients instead of waiting for the whole audio.

You can use the [PullAudioOutputStream](#), [PushAudioOutputStream](#), [Synthesizing event](#), and [AudioDataStream](#) of the Speech SDK to enable streaming.

Taking [AudioDataStream](#) as an example:

C#

```
using (var synthesizer = new SpeechSynthesizer(config, null as AudioConfig))
{
    using (var result = await synthesizer.StartSpeakingTextAsync(text))
    {
        using (var audioDataStream = AudioDataStream.FromResult(result))
        {
            byte[] buffer = new byte[16000];
            uint filledSize = 0;
            while ((filledSize = audioDataStream.ReadData(buffer)) > 0)
            {
                Console.WriteLine($"{filledSize} bytes received.");
            }
        }
    }
}
```

Pre-connect and reuse SpeechSynthesizer

The Speech SDK uses a websocket to communicate with the service. Ideally, the network latency should be one route trip time (RTT). If the connection is newly established, the network latency will include extra time to establish the connection. The establishment of a websocket connection needs the TCP handshake, SSL handshake, HTTP connection, and protocol upgrade, which introduces time delay. To avoid the connection latency, we recommend pre-connecting and reusing the [SpeechSynthesizer](#).

Pre-connect

To pre-connect, establish a connection to the Speech service when you know the connection will be needed soon. For example, if you are building a speech bot in client, you can pre-connect to the speech synthesis service when the user starts to talk, and call [SpeakTextAsync](#) when the bot reply text is ready.

C#

```
using (var synthesizer = new SpeechSynthesizer(uspConfig, null as
AudioConfig))
{
    using (var connection = Connection.FromSpeechSynthesizer(synthesizer))
    {
        connection.Open(true);
    }
    await synthesizer.SpeakTextAsync(text);
}
```

ⓘ Note

If the synthesize text is available, just call `SpeakTextAsync` to synthesize the audio. The SDK will handle the connection.

Reuse SpeechSynthesizer

Another way to reduce the connection latency is to reuse the `SpeechSynthesizer` so you don't need to create a new `SpeechSynthesizer` for each synthesis. We recommend using object pool in service scenario, see our sample code for [C# ↗](#) and [Java ↗](#).

Transmit compressed audio over the network

When the network is unstable or with limited bandwidth, the payload size will also impact latency. Meanwhile, a compressed audio format helps to save the users' network bandwidth, which is especially valuable for mobile users.

We support many compressed formats including `opus`, `webm`, `mp3`, `silk`, and so on, see the full list in [SpeechSynthesisOutputFormat](#). For example, the bitrate of `Riff24Khz16BitMonoPcm` format is 384 kbps, while `Audio24Khz48KBitRateMonoMp3` only costs 48 kbps. Our Speech SDK will automatically use a compressed format for transmission when a `pcm` output format is set. For Linux and Windows, `GStreamer` is required to enable this feature. Refer [this instruction](#) to install and configure `GStreamer` for Speech SDK. For Android, iOS and macOS, no extra configuration is needed starting version 1.20.

Others tips

Cache CRL files

The Speech SDK uses CRL files to check the certification. Caching the CRL files until expired helps you avoid downloading CRL files every time. See [How to configure OpenSSL for Linux](#) for details.

Use latest Speech SDK

We keep improving the Speech SDK's performance, so try to use the latest Speech SDK in your application.

Load test guideline

You may use load test to test the speech synthesis service capacity and latency. Here are some guidelines.

- The speech synthesis service has the ability to autoscale, but takes time to scale out. If the concurrency is increased in a short time, the client may get long latency or `429` error code (too many requests). So, we recommend you increase your concurrency step by step in load test. [See this article](#) for more details, especially [this example of workload patterns](#).
- You can leverage our sample using object pool ([C# ↗](#) and [Java ↗](#)) for load test and getting the latency numbers. You can modify the test turns and concurrency in the sample to meet your target concurrency.
- The service has quota limitation based on the real traffic, therefore, if you want to perform load test with the concurrency much higher than your real traffic, connect before your test.

Next steps

- [See the samples ↗](#) on GitHub

Get facial position with viseme

Article • 01/11/2023 • 7 minutes to read

ⓘ Note

Viseme ID supports neural voices in **all viseme-supported locales**. Scalable Vector Graphics (SVG) only supports neural voices in `en-US` locale, and blend shapes supports neural voices in `en-US` and `zh-CN` locales.

A *viseme* is the visual description of a phoneme in spoken language. It defines the position of the face and mouth while a person is speaking. Each viseme depicts the key facial poses for a specific set of phonemes.

You can use visemes to control the movement of 2D and 3D avatar models, so that the facial positions are best aligned with synthetic speech. For example, you can:

- Create an animated virtual voice assistant for intelligent kiosks, building multi-mode integrated services for your customers.
- Build immersive news broadcasts and improve audience experiences with natural face and mouth movements.
- Generate more interactive gaming avatars and cartoon characters that can speak with dynamic content.
- Make more effective language teaching videos that help language learners understand the mouth behavior of each word and phoneme.
- People with hearing impairment can also pick up sounds visually and "lip-read" speech content that shows visemes on an animated face.

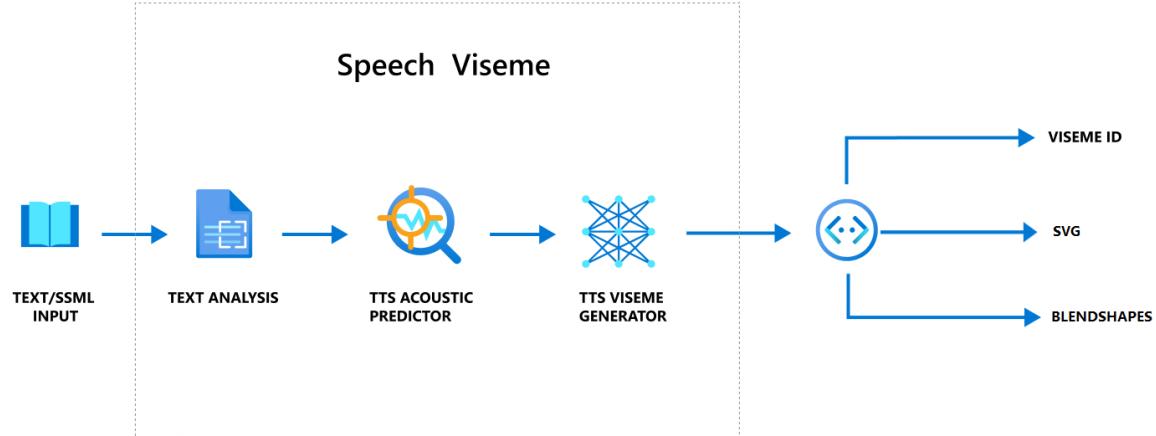
For more information about visemes, view this [introductory video](#).

<https://www.youtube-nocookie.com/embed/ui9XT47uwxs>

Overall workflow of producing viseme with speech

Neural Text-to-Speech (Neural TTS) turns input text or SSML (Speech Synthesis Markup Language) into lifelike synthesized speech. Speech audio output can be accompanied by viseme ID, Scalable Vector Graphics (SVG), or blend shapes. Using a 2D or 3D rendering engine, you can use these viseme events to animate your avatar.

The overall workflow of viseme is depicted in the following flowchart:



Viseme ID

Viseme ID refers to an integer number that specifies a viseme. We offer 22 different visemes, each depicting the mouth position for a specific set of phonemes. There's no one-to-one correspondence between visemes and phonemes. Often, several phonemes correspond to a single viseme, because they look the same on the speaker's face when they're produced, such as **s** and **z**. For more specific information, see the table for [mapping phonemes to viseme IDs](#).

Speech audio output can be accompanied by viseme IDs and `Audio offset`. The `Audio offset` indicates the offset timestamp that represents the start time of each viseme, in ticks (100 nanoseconds).

Map phonemes to visemes

Visemes vary by language and locale. Each locale has a set of visemes that correspond to its specific phonemes. The [SSML phonetic alphabets](#) documentation maps viseme IDs to the corresponding International Phonetic Alphabet (IPA) phonemes. The table below shows a mapping relationship between viseme IDs and mouth positions, listing typical IPA phonemes for each viseme ID.

Viseme ID	IPA	Mouth position
-----------	-----	----------------

Viseme ID	IPA	Mouth position
0	Silence	
1	æ, ə,ʌ	
2	ɑ	
3	ɔ	

Viseme ID	IPA	Mouth position
4	ɛ, ʊ	
5	ɔ	
6	j, i, ɪ	
7	w, u	

Viseme ID	IPA	Mouth position
8	o	
9	aʊ	
10	ɔɪ	
11	aɪ	

Viseme ID	IPA	Mouth position
12	h	
13	u	
14	ɪ	
15	s, z	

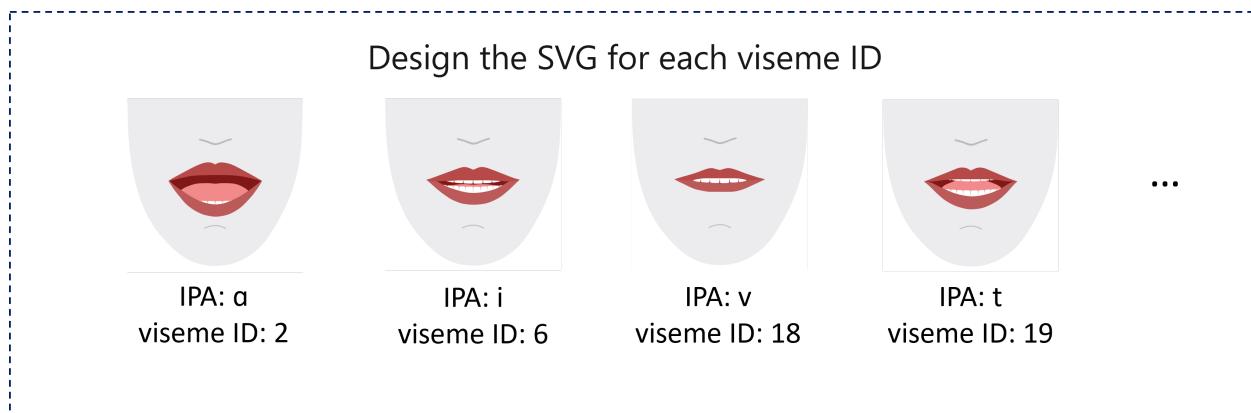
Viseme ID	IPA	Mouth position
16	ʃ, tʃ, dʒ, ʒ	
17	ð	
18	f, v	
19	d, t, n, θ	

Viseme ID	IPA	Mouth position
20	k, g, ŋ	
21	p, b, m	

2D SVG animation

For 2D characters, you can design a character that suits your scenario and use Scalable Vector Graphics (SVG) for each viseme ID to get a time-based face position.

With temporal tags that are provided in a viseme event, these well-designed SVGs will be processed with smoothing modifications, and provide robust animation to the users. For example, the following illustration shows a red-lipped character that's designed for language learning.



3D blend shapes animation

You can use blend shapes to drive the facial movements of a 3D character that you designed.

The blend shapes JSON string is represented as a 2-dimensional matrix. Each row represents a frame. Each frame (in 60 FPS) contains an array of 55 facial positions.

Get viseme events with the Speech SDK

To get viseme with your synthesized speech, subscribe to the `VisemeReceived` event in the Speech SDK.

ⓘ Note

To request SVG or blend shapes output, you should use the `mstts:viseme` element in SSML. For details, see [how to use viseme element in SSML](#).

The following snippet shows how to subscribe to the viseme event:

C#

```
using (var synthesizer = new SpeechSynthesizer(speechConfig, audioConfig))
{
    // Subscribes to viseme received event
    synthesizer.VisemeReceived += (s, e) =>
    {
        Console.WriteLine($"Viseme event received. Audio offset: " +
            $"{e.AudioOffset / 10000}ms, viseme id: {e.VisemeId}.");

        // `Animation` is an xml string for SVG or a json string for blend
        // shapes
        var animation = e.Animation;
    };

    // If VisemeID is the only thing you want, you can also use
    `SpeakTextAsync()`
    var result = await synthesizer.SpeakSsmlAsync(ssml);
}
```

Here's an example of the viseme output.

Viseme ID

text

(Viseme), Viseme ID: 1, Audio offset: 200ms.

(Viseme), Viseme ID: 5, Audio offset: 850ms.

.....

(Viseme), Viseme ID: 13, Audio offset: 2350ms.

After you obtain the viseme output, you can use these events to drive character animation. You can build your own characters and automatically animate them.

Next steps

- [SSML phonetic alphabets](#)
 - [How to improve synthesis with SSML](#)
-

Additional resources

Documentation

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[How to lower speech synthesis latency using Speech SDK - Azure Cognitive Services](#)

How to lower speech synthesis latency using Speech SDK, including streaming, pre-connection, and so on.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[Quickstart: The Speech CLI - Speech service - Azure Cognitive Services](#)

In this Azure Speech CLI quickstart, you interact with speech-to-text, text-to-speech, and speech translation without having to write code.

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[Show 5 more](#)

What is Custom Neural Voice?

Article • 01/11/2023 • 5 minutes to read

Custom Neural Voice (CNV) is a text-to-speech feature that lets you create a one-of-a-kind, customized, synthetic voice for your applications. With Custom Neural Voice, you can build a highly natural-sounding voice for your brand or characters by providing human speech samples as training data.

Important

Custom Neural Voice access is **limited** based on eligibility and usage criteria. Request access on the [intake form](#).

Out of the box, [text-to-speech](#) can be used with prebuilt neural voices for each [supported language](#). The prebuilt neural voices work very well in most text-to-speech scenarios if a unique voice isn't required.

Custom Neural Voice is based on the neural text-to-speech technology and the multilingual, multi-speaker, universal model. You can create synthetic voices that are rich in speaking styles, or adaptable cross languages. The realistic and natural sounding voice of Custom Neural Voice can represent brands, personify machines, and allow users to interact with applications conversationally. See the [supported languages](#) for Custom Neural Voice.

How does it work?

To create a custom neural voice, use [Speech Studio](#) to upload the recorded audio and corresponding scripts, train the model, and deploy the voice to a custom endpoint.

Tip

Try [Custom Neural Voice \(CNV\) Lite](#) to demo and evaluate CNV before investing in professional recordings to create a higher-quality voice.

Creating a great custom neural voice requires careful quality control in each step, from voice design and data preparation, to the deployment of the voice model to your system.

Before you get started in Speech Studio, here are some considerations:

- **Design a persona** of the voice that represents your brand by using a persona brief document. This document defines elements such as the features of the voice, and the character behind the voice. This helps to guide the process of creating a custom neural voice model, including defining the scripts, selecting your voice talent, training, and voice tuning.
- **Select the recording script** to represent the user scenarios for your voice. For example, you can use the phrases from bot conversations as your recording script if you're creating a customer service bot. Include different sentence types in your scripts, including statements, questions, and exclamations.

Here's an overview of the steps to create a custom neural voice in Speech Studio:

1. **Create a project** to contain your data, voice models, tests, and endpoints. Each project is specific to a country and language. If you are going to create multiple voices, it's recommended that you create a project for each voice.
2. **Set up voice talent**. Before you can train a neural voice, you must submit a recording of the voice talent's consent statement. The voice talent statement is a recording of the voice talent reading a statement that they consent to the usage of their speech data to train a custom voice model.
3. **Prepare training data** in the right **format**. It's a good idea to capture the audio recordings in a professional quality recording studio to achieve a high signal-to-noise ratio. The quality of the voice model depends heavily on your training data. Consistent volume, speaking rate, pitch, and consistency in expressive mannerisms of speech are required.
4. **Train your voice model**. Select at least 300 utterances to create a custom neural voice. A series of data quality checks are automatically performed when you upload them. To build high-quality voice models, you should fix any errors and submit again.
5. **Test your voice**. Prepare test scripts for your voice model that cover the different use cases for your apps. It's a good idea to use scripts within and outside the training dataset, so you can test the quality more broadly for different content.
6. **Deploy and use your voice model** in your apps.

You can tune, adjust, and use your custom voice, similarly as you would use a prebuilt neural voice. Convert text into speech in real time, or generate audio content offline with text input. You can do this by using the [REST API](#), the [Speech SDK](#), or the [Speech Studio](#).

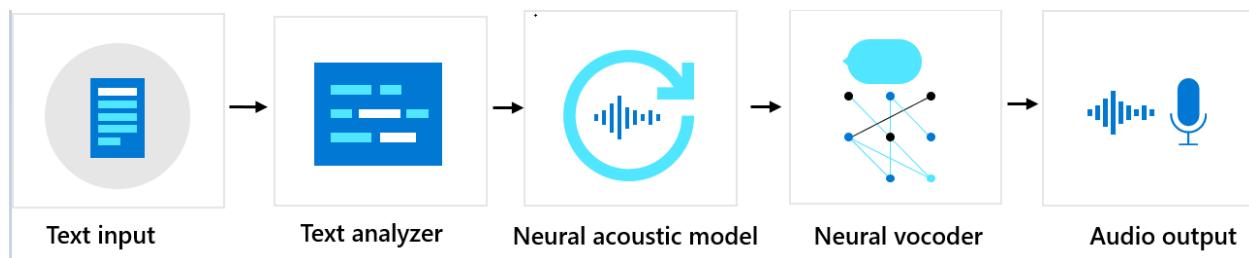
The style and the characteristics of the trained voice model depend on the style and the quality of the recordings from the voice talent used for training. However, you can make several adjustments by using [SSML \(Speech Synthesis Markup Language\)](#) when you make the API calls to your voice model to generate synthetic speech. SSML is the

markup language used to communicate with the text-to-speech service to convert text into audio. The adjustments you can make include change of pitch, rate, intonation, and pronunciation correction. If the voice model is built with multiple styles, you can also use SSML to switch the styles.

Components sequence

Custom Neural Voice consists of three major components: the text analyzer, the neural acoustic model, and the neural vocoder. To generate natural synthetic speech from text, text is first input into the text analyzer, which provides output in the form of phoneme sequence. A *phoneme* is a basic unit of sound that distinguishes one word from another in a particular language. A sequence of phonemes defines the pronunciations of the words provided in the text.

Next, the phoneme sequence goes into the neural acoustic model to predict acoustic features that define speech signals. Acoustic features include the timbre, the speaking style, speed, intonations, and stress patterns. Finally, the neural vocoder converts the acoustic features into audible waves, so that synthetic speech is generated.



Neural text-to-speech voice models are trained by using deep neural networks based on the recording samples of human voices. For more information, see [this Microsoft blog post](#). To learn more about how a neural vocoder is trained, see [this Microsoft blog post](#).

Migrate to Custom Neural Voice

If you're using the old version of Custom Voice (which is scheduled to be retired in February 2024), see [How to migrate to Custom Neural Voice](#).

Responsible use of AI

To learn how to use Custom Neural Voice responsibly, check the following articles.

- [Transparency note and use cases for Custom Neural Voice](#)
- [Characteristics and limitations for using Custom Neural Voice](#)

- Limited access to Custom Neural Voice
- Guidelines for responsible deployment of synthetic voice technology
- Disclosure for voice talent
- Disclosure design guidelines
- Disclosure design patterns
- Code of Conduct for Text-to-Speech integrations
- Data, privacy, and security for Custom Neural Voice

Next steps

- Create a project
 - Prepare training data
 - Train model
-

Additional resources

Documentation

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Speech Studio overview - Speech service - Azure Cognitive Services](#)

Speech Studio is a set of UI-based tools for building and integrating features from Azure Speech service in your applications.

[Batch synthesis API \(Preview\) for text to speech - Speech service - Azure Cognitive Services](#)

Learn how to use the batch synthesis API for asynchronous synthesis of long-form text to speech.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[How to lower speech synthesis latency using Speech SDK - Azure Cognitive Services](#)

How to lower speech synthesis latency using Speech SDK, including streaming, pre-connection, and so on.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[How to synthesize speech from text - Speech service - Azure Cognitive Services](#)

Learn how to convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Show 5 more](#)

Custom Neural Voice Lite (preview)

Article • 01/11/2023 • 6 minutes to read

Speech Studio provides two Custom Neural Voice (CNV) project types: CNV Lite and CNV Pro.

- Custom Neural Voice (CNV) Pro allows you to upload your training data collected through professional recording studios and create a higher-quality voice that is nearly indistinguishable from its human samples. CNV Pro access is limited based on eligibility and usage criteria. Request access on the [intake form](#).
- Custom Neural Voice (CNV) Lite is a project type in public preview. You can demo and evaluate Custom Neural Voice before investing in professional recordings to create a higher-quality voice. No application is required. Microsoft restricts and selects the recording and testing samples for use with CNV Lite. You must apply for full access to CNV Pro in order to deploy and use the CNV Lite model for business purpose.

With a CNV Lite project, you record your voice online by reading 20-50 pre-defined scripts provided by Microsoft. After you've recorded at least 20 samples, you can start to train a model. Once the model is trained successfully, you can review the model and check out 20 output samples produced with another set of pre-defined scripts.

See the [supported languages](#) for Custom Neural Voice.

Compare project types

The following table summarizes key differences between the CNV Lite and CNV Pro project types.

Items	Lite (Preview)	Pro
Target scenarios	Demonstration or evaluation	Professional scenarios like brand and character voices for chat bots, or audio content reading.
Training data	Record online using Speech Studio	Bring your own data. Recording in a professional studio is recommended.
Scripts for recording	Provided in Speech Studio	Use your own scripts that match the use case scenario. Microsoft provides example scripts for reference.

Items	Lite (Preview)	Pro
Required data size	20-50 utterances	300-2000 utterances
Training time	Less than one compute hour	Approximately 20-40 compute hours
Voice quality	Moderate quality	High quality
Availability	Anyone can record samples online and train a model for demo and evaluation purpose. Full access to Custom Neural Voice is required if you want to deploy the CNV Lite model for business use.	Data upload isn't restricted, but you can only train and deploy a CNV Pro model after access is approved. CNV Pro access is limited based on eligibility and usage criteria. Request access on the intake form .
Pricing	Per unit prices apply equally for both the CNV Lite and CNV Pro projects. Check the pricing details here .	Per unit prices apply equally for both the CNV Lite and CNV Pro projects. Check the pricing details here .

Create a Custom Neural Voice Lite project

To create a Custom Neural Voice Lite project, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select the subscription and Speech resource to work with.

Important

Custom Neural Voice training is currently only available in some regions. See footnotes in the [regions](#) table for more information.

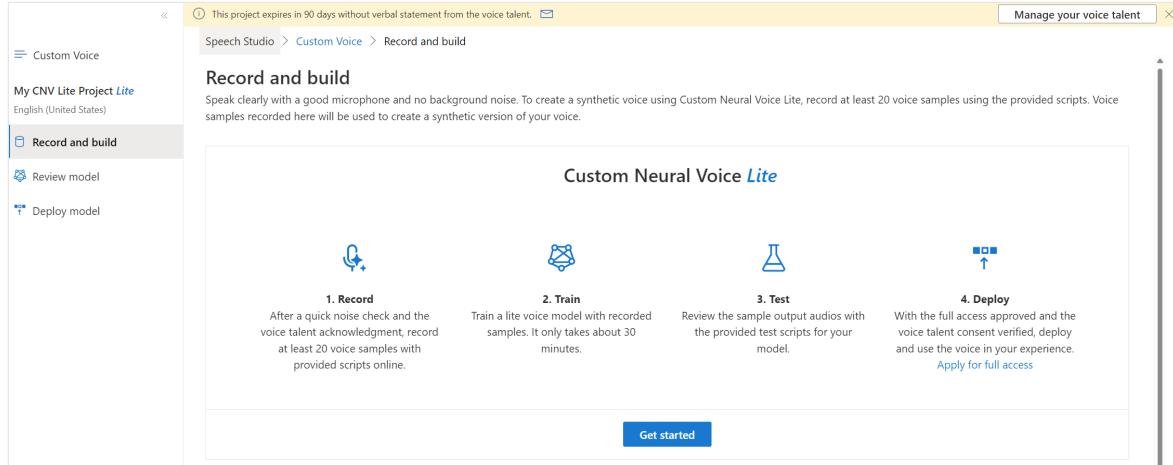
3. Select **Custom Voice > Create a project**.
4. Select **Custom Neural Voice Lite > Next**.

Note

To create a Custom Neural Voice Pro project, see [Create a project for Custom Neural Voice](#).

5. Follow the instructions provided by the wizard to create your project.

6. Select the new project by name or select **Go to project**. You'll see these menu items in the left panel: **Record and build**, **Review model**, and **Deploy model**.



The CNV Lite project expires after 90 days unless the [verbal statement](#) recorded by the voice talent is submitted.

Record and build a CNV Lite model

Record at least 20 voice samples (up to 50) with provided scripts online. Voice samples recorded here will be used to create a synthetic version of your voice.

Here are some tips to help you record your voice samples:

- Use a good microphone. Increase the clarity of your samples by using a high-quality microphone. Speak about 8 inches away from the microphone to avoid mouth noises.
- Avoid background noise. Record in a quiet room without background noise or echoing.
- Relax and speak naturally. Allow yourself to express emotions as you read the sentences.
- Record in one take. To keep a consistent energy level, record all sentences in one session.
- Pronounce each word correctly, and speak clearly.

To record and build a CNV Lite model, follow these steps:

1. Select **Custom Voice** > Your project name > **Record and build**.
2. Select **Get started**.
3. Read the Voice talent terms of use carefully. Select the checkbox to acknowledge the terms of use.
4. Select **Accept**
5. Press the microphone icon to start the noise check. This noise check will take only a few seconds, and you won't need to speak during it.

6. If noise was detected, you can select **Check again** to repeat the noise check. If no noise was detected, you can select **Done** to proceed to the next step.

Noise check

Make sure you're in a quiet room without background noise or echoing. This noise check will take only a few seconds, and you won't need to speak during it.



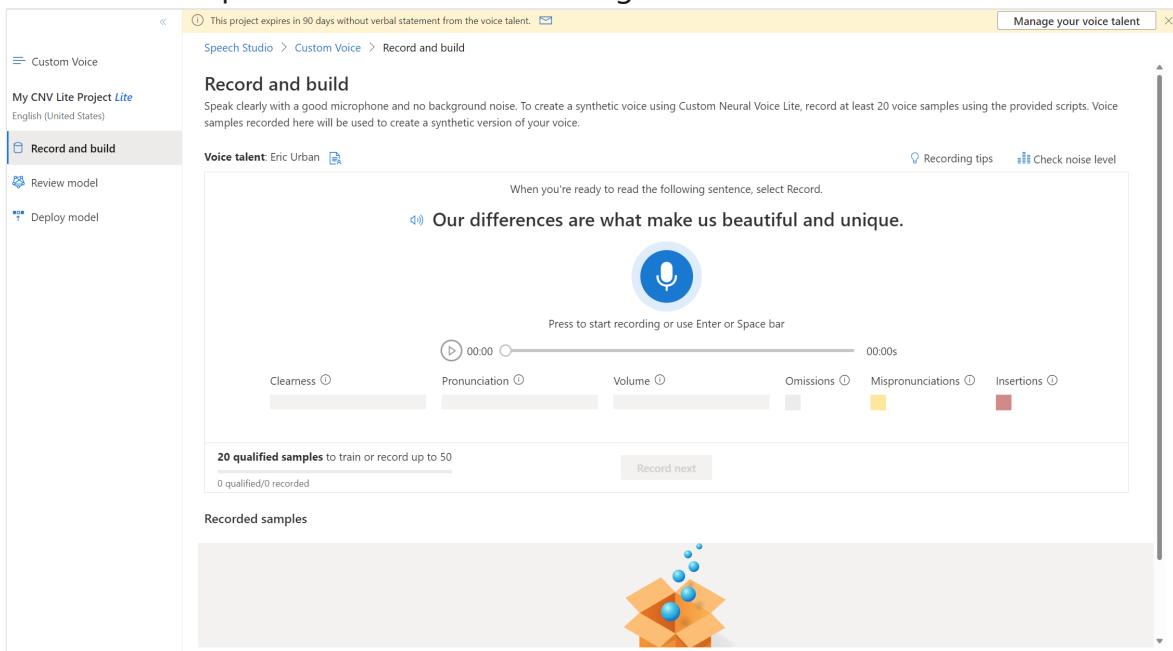
Check again

⚠️ Noise detected

For the best results, we recommend moving to a quieter area without background noise before recording your voice samples.

Done Cancel

7. Review the recording tips and select **Got it**. For the best results, go to a quiet area without background noise before recording your voice samples.
8. Press the microphone icon to start recording.



9. Press the stop icon to stop recording.
10. Review quality metrics. After recording each sample, check its quality metric before continuing to the next one.
11. Record more samples. Although you can create a model with just 20 samples, it's recommended that you record up to 50 to get better quality.
12. Select **Train model** to start the training process.

The training process takes approximately one compute hour. You can check the progress of the training process in the **Review model** page.

Review model

To review the CNV Lite model and listen to your own synthetic voice, follow these steps:

1. Select **Custom Voice** > Your project name > **Review model**. Here you can review the voice model name, model language, sample data size, and training progress. The voice name is composed of the word "Neural" appended to your project name.
2. Select the voice model name to review the model details and listen to the sample text-to-speech results.
3. Select the play icon to hear your voice speak each script.

The screenshot shows the 'Review model' section of the Azure Custom Voice interface. It displays the following information:

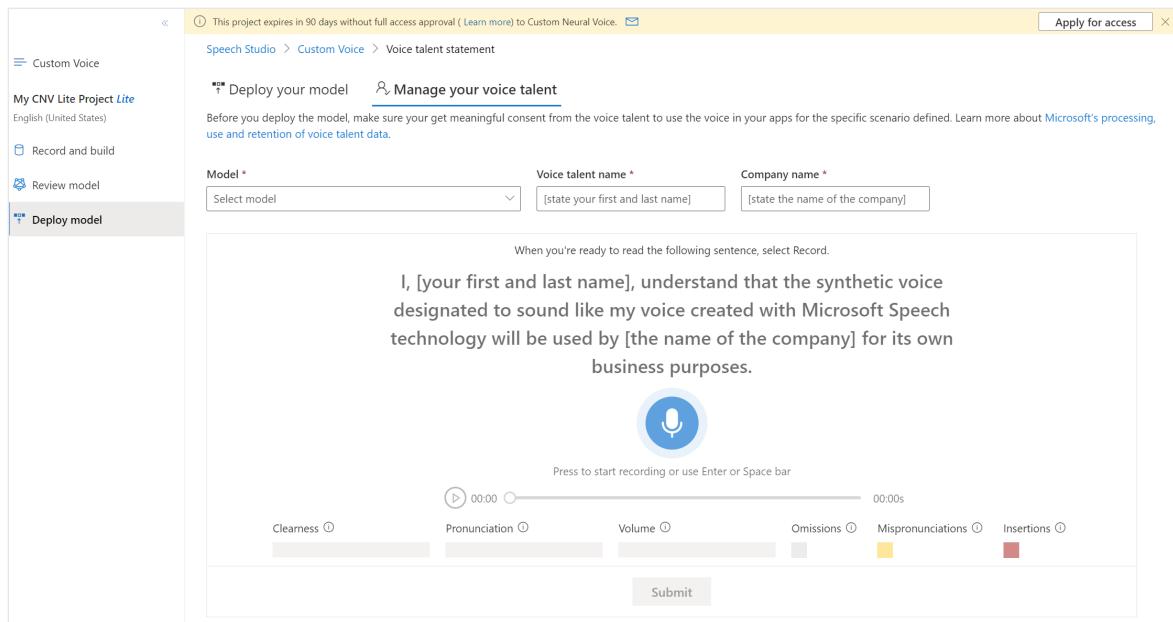
- Project Details:** My CNV Lite Project *Lite* (English (United States)).
- Model Information:** Gender: Male, Model language: English (United States), Model type: Neural, Utterances: 20, Sampling rate: 24000, Status: Succeeded (green circle).
- Model ID:** ae6db871-18cd-42ab-856f-90282d9c5fb3.
- Created:** 10/15/2022 8:18 AM.
- Sample Output:** A section titled 'Text input' shows a message: 'Congratulations! You have successfully created your synthetic voice with Custom Neural Voice Lite!' Below it, there are three audio playback icons labeled 'Audio'.
- Instructions:** Text at the bottom encourages users to 'To deploy your voice model and use it in your business applications, you must get the full access to Custom Neural Voice and the explicit consent from your voice talent.'
- Note:** Text at the bottom states: 'With the full access, you can create an even more natural voice by bringing your own data recorded in professional studios, using Custom Neural Voice Pro.'

Submit verbal statement

A verbal statement recorded by the voice talent is required before you can [deploy the model](#) for your business use.

To submit the voice talent verbal statement, follow these steps:

1. Select **Custom Voice** > Your project name > **Deploy model** > **Manage your voice talent**.



2. Select the model.
3. Enter the voice talent name and company name.
4. Read and record the statement. Select the microphone icon to start recording.
Select the stop icon to stop recording.
5. Select **Submit** to submit the statement.
6. Check the processing status in the script table at the bottom of the dashboard.

Once the status is **Succeeded**, you can [deploy the model](#).

Deploy model

To deploy your voice model and use it in your applications, you must get the full access to Custom Neural Voice. Request access on the [intake form](#). Within approximately 10 business days, you'll receive an email with the approval status. A [verbal statement](#) recorded by the voice talent is also required before you can deploy the model for your business use.

To deploy a CNV Lite model, follow these steps:

1. Select **Custom Voice** > Your project name > **Deploy model** > **Deploy model**.
2. Select a voice model name and then select **Next**.
3. Enter a name and description for your endpoint and then select **Next**.
4. Select the checkbox to agree to the terms of use and then select **Next**.
5. Select **Deploy** to deploy the model.

From here, you can use the CNV Lite voice model similarly as you would use a CNV Pro voice model. For example, you can [suspend or resume](#) an endpoint after it's created, to limit spend and conserve resources that aren't in use. You can also access the voice in the [Audio Content Creation tool](#) in the [Speech Studio](#).

Create a project for Custom Neural Voice

Article • 01/11/2023 • 2 minutes to read

Content for [Custom Neural Voice](#) like data, models, tests, and endpoints are organized into projects in Speech Studio. Each project is specific to a country and language, and the gender of the voice you want to create. For example, you might create a project for a female voice for your call center's chat bots that use English in the United States.

💡 Tip

Try [Custom Neural Voice \(CNV\) Lite](#) to demo and evaluate CNV before investing in professional recordings to create a higher-quality voice.

All it takes to get started are a handful of audio files and the associated transcriptions. See if Custom Neural Voice supports your [language](#) and [region](#).

Create a Custom Neural Voice Pro project

To create a Custom Neural Voice Pro project, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select the subscription and Speech resource to work with.

ⓘ Important

Custom Neural Voice training is currently only available in some regions. After your voice model is trained in a supported region, you can copy it to a Speech resource in another region as needed. See footnotes in the [regions](#) table for more information.

3. Select **Custom Voice > Create a project**.
4. Select **Custom Neural Voice Pro > Next**.
5. Follow the instructions provided by the wizard to create your project.

Select the new project by name or select **Go to project**. You'll see these menu items in the left panel: **Set up voice talent**, **Prepare training data**, **Train model**, and **Deploy model**.

Next steps

- [Set up voice talent](#)
- [Prepare data for custom neural voice](#)
- [Train your voice model](#)
- [Deploy and use your voice model](#)

Set up voice talent for Custom Neural Voice

Article • 10/27/2022 • 2 minutes to read

A voice talent is an individual or target speaker whose voices are recorded and used to create neural voice models.

Before you can train a neural voice, you must submit a recording of the voice talent's consent statement. The voice talent statement is a recording of the voice talent reading a statement that they consent to the usage of their speech data to train a custom voice model. The consent statement is also used to verify that the voice talent is the same person as the speaker in the training data.

Tip

Before you get started in Speech Studio, define your voice **persona** and **choose the right voice talent**.

You can find the verbal consent statement in multiple languages on [GitHub](#). The language of the verbal statement must be the same as your recording. See also the [disclosure for voice talent](#).

Add voice talent

To add a voice talent profile and upload their consent statement, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Voice** > Your project name > **Set up voice talent** > **Add voice talent**.
3. In the **Add new voice talent** wizard, describe the characteristics of the voice you're going to create. The scenarios that you specify here must be consistent with what you provided in the application form.
4. Select **Next**.
5. On the **Upload voice talent statement** page, follow the instructions to upload the voice talent statement you've recorded beforehand. Make sure the verbal statement was [recorded](#) with the same settings, environment, and speaking style

as your training data.

Add new voice talent

Define voice characteristics
 Upload voice talent statement
 Review and create

Upload voice talent statement

Upload an audio recording of your voice talent saying the sentence below. This audio file will be used to create a voice signature of your voice talent and to verify against your training data when you create a voice model. Learn more about [Microsoft's processing, use and retention of voice talent data](#).

I [state your first and last name] am aware that recordings of my voice will be used by [state the name of the company] to create and use a synthetic version of my voice.

Voice talent name *

Provide the first and last name of your voice talent

Company name *

Provide your company name

Recorded statement from your voice talent *

Browse files...

Only .wav and .mp3 files are accepted.

[Back](#) [Next](#) [Cancel](#)

6. Enter the voice talent name and company name. The voice talent name must be the name of the person who recorded the consent statement. The company name must match the company name that was spoken in the recorded statement.

7. Select **Next**.

8. Review the voice talent and persona details, and select **Submit**.

After the voice talent status is *Succeeded*, you can proceed to [train your custom voice model](#).

Next steps

- [Prepare data for custom neural voice](#)
- [Train your voice model](#)
- [Deploy and use your voice model](#)

Prepare training data for Custom Neural Voice

Article • 10/27/2022 • 7 minutes to read

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. For details on recording voice samples, see [the tutorial](#). 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.

All data you upload must meet the requirements for the data type that you choose. It's important to correctly format your data before it's uploaded, which ensures the data will be accurately processed by the Speech service. To confirm that your data is correctly formatted, see [Training data types](#).

Note

- Standard subscription (S0) users can upload five data files simultaneously. If you reach the limit, wait until at least one of your data files finishes importing. Then try again.
- The maximum number of data files allowed to be imported per subscription is 500 .zip files for standard subscription (S0) users. Please see out [Speech service quotas and limits](#) for more details.

Upload your data

When you're ready to upload your data, go to the **Prepare training data** tab to add your first training set and upload data. A *training set* is a set of audio utterances and their mapping scripts used for training a voice model. You can use a training set to organize your training data. The service checks data readiness per each training set. You can import multiple data to a training set.

To upload training data, follow these steps:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Voice** > Your project name > **Prepare training data** > **Upload data**.
3. In the **Upload data** wizard, choose a [data type](#) and then select **Next**.

4. Select local files from your computer or enter the Azure Blob storage URL to upload data.
5. Under **Specify the target training set**, select an existing training set or create a new one. If you created a new training set, make sure it's selected in the drop-down list before you continue.
6. Select **Next**.
7. Enter a name and description for your data and then select **Next**.
8. Review the upload details, and select **Submit**.

 **Note**

Duplicate IDs are not accepted. Utterances with the same ID will be removed.

Duplicate audio names are removed from the training. Make sure the data you select don't contain the same audio names within the .zip file or across multiple .zip files. If utterance IDs (either in audio or script files) are duplicates, they're rejected.

Data files are automatically validated when you select **Submit**. Data validation includes series of checks on the audio files to verify their file format, size, and sampling rate. If there are any errors, fix them and submit again.

After you upload the data, you can check the details in the training set detail view. On the **Overview** tab, you can further check the pronunciation scores and the noise level for each of your data. The pronunciation score ranges from 0-100. A score below 70 normally indicates a speech error or script mismatch. A heavy accent can reduce your pronunciation score and affect the generated digital voice.

Resolve data issues online

After upload, you can check the data details of the training set. Before continuing to [train your voice model](#), you should try to resolve any data issues.

You can resolve data issues per utterance in Speech Studio.

1. On the **Data details** page, select individual utterances you want to edit, then click **Edit**.

Overview Data details

[Download](#) [Upload data](#) [Analyze data](#) [Edit](#) [Delete](#)

Name	Audio	Transcript	Score ⓘ	SNR ⓘ	Duration	Issue ⓘ
0034900001		When he takes classes to visit the memorial, he shows his students his father's name.	97.40	40.84	00:00:04	Silence auto
0035100001		The National Love affair with stealthy weapons will endure for several reasons.	100.00	37.78	00:00:04	Silence auto
0035200001		But Altman doesn't let the audiences outrage mount.	98.80	41.67	00:00:03	
<input checked="" type="checkbox"/> 0035400001		There is also a warehouse consolidation planned for 2000.	100.00	39.86	00:00:03	Non-normalized text

2. Edit window will be displayed.

Edit transcript and recording file

Name
0035400001

Issue
Non-normalized text

Transcript and recording

00:00 00:00s

There is also a warehouse consolidation planned for 2000.

Update recording file

3. Update transcript or recording file according to issue description on the edit window.

You can edit transcript in the text box, then click **Done**

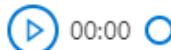
Edit transcript and recording file

X

Name

0035400001

Transcript and recording



00:00



00:00s

There is also a warehouse consolidation planned for two thousand.

Update recording file

Done

Cancel



If you need to update recording file, select **Update recording file**, then upload the fixed recording file (.wav).

Edit transcript and recording file

X

Name

0041700001

Issue

End silence issue

Transcript and recording



00:00



00:00s

In the face of today's nonviolent protests, he sent his police in to beat us.

Update recording file



Drag and drop.

[Browse for a file](#)



Done

Cancel

4. After the data in a training set are updated, you need to check the data quality by clicking **Analyze data** before using this training set for training.

You can't select this training set for training model before the analysis is complete.

You can also delete utterances with issues by selecting them and clicking **Delete**.

Typical data issues

The issues are divided into three types. Refer to the following tables to check the respective types of errors.

Auto-rejected

Data with these errors won't be used for training. Imported data with errors will be ignored, so you don't need to delete them. You can resubmit the corrected data for training.

Category	Name	Description
Script	Invalid separator	You must separate the utterance ID and the script content with a Tab character.
Script	Invalid script ID	The script line ID must be numeric.
Script	Duplicated script	Each line of the script content must be unique. The line is duplicated with {}.
Script	Script too long	The script must be less than 1,000 characters.
Script	No matching audio	The ID of each utterance (each line of the script file) must match the audio ID.
Script	No valid script	No valid script is found in this dataset. Fix the script lines that appear in the detailed issue list.
Audio	No matching script	No audio files match the script ID. The name of the .wav files must match with the IDs in the script file.

Category	Name	Description
Audio	Invalid audio format	The audio format of the .wav files is invalid. Check the .wav file format by using an audio tool like SoX .
Audio	Low sampling rate	The sampling rate of the .wav files can't be lower than 16 KHz.
Audio	Too long audio	Audio duration is longer than 30 seconds. Split the long audio into multiple files. It's a good idea to make utterances shorter than 15 seconds.
Audio	No valid audio	No valid audio is found in this dataset. Check your audio data and upload again.
Mismatch	Low scored utterance	Sentence-level pronunciation score is lower than 70. Review the script and the audio content to make sure they match.

Auto-fixed

The following errors are fixed automatically, but you should review and confirm the fixes are made correctly.

Category	Name	Description
Mismatch	Silence auto fixed	The start silence is detected to be shorter than 100 ms, and has been extended to 100 ms automatically. Download the normalized dataset and review it.
Mismatch	Silence auto fixed	The end silence is detected to be shorter than 100 ms, and has been extended to 100 ms automatically. Download the normalized dataset and review it.

Manual check required

Unresolved errors listed in the next table affect the quality of training, but data with these errors won't be excluded during training. For higher-quality training, it's a good idea to fix these errors manually.

Category	Name	Description
Script	Non-normalized text	This script contains digits. Expand them to normalized words, and match with the audio. For example, normalize <i>123</i> to <i>one hundred and twenty-three</i> .
Script	Non-normalized text	This script contains symbols. Normalize the symbols to match the audio. For example, normalize <i>50%</i> to <i>fifty percent</i> .

Category	Name	Description
Script	Not enough question utterances	At least 10 percent of the total utterances should be question sentences. This helps the voice model properly express a questioning tone.
Script	Not enough exclamation utterances	At least 10 percent of the total utterances should be exclamation sentences. This helps the voice model properly express an excited tone.
Script	No valid end punctuation	Add one of the following at the end of the line: full stop (half-width '.' or full-width '。'), exclamation point (half-width '!' or full-width '！'), or question mark (half-width '?' or full-width '？').
Audio	Low sampling rate for neural voice	It's recommended that the sampling rate of your .wav files should be 24 KHz or higher for creating neural voices. If it's lower, it will be automatically raised to 24 KHz.
Volume	Overall volume too low	Volume shouldn't be lower than -18 dB (10 percent of max volume). Control the volume average level within proper range during the sample recording or data preparation.
Volume	Volume overflow	Overflowing volume is detected at {}s. Adjust the recording equipment to avoid the volume overflow at its peak value.
Volume	Start silence issue	The first 100 ms of silence isn't clean. Reduce the recording noise floor level, and leave the first 100 ms at the start silent.
Volume	End silence issue	The last 100 ms of silence isn't clean. Reduce the recording noise floor level, and leave the last 100 ms at the end silent.
Mismatch	Low scored words	Review the script and the audio content to make sure they match, and control the noise floor level. Reduce the length of long silence, or split the audio into multiple utterances if it's too long.
Mismatch	Start silence issue	Extra audio was heard before the first word. Review the script and the audio content to make sure they match, control the noise floor level, and make the first 100 ms silent.

Category	Name	Description
Mismatch	End silence issue	Extra audio was heard after the last word. Review the script and the audio content to make sure they match, control the noise floor level, and make the last 100 ms silent.
Mismatch	Low signal-noise ratio	Audio SNR level is lower than 20 dB. At least 35 dB is recommended.
Mismatch	No score available	Failed to recognize speech content in this audio. Check the audio and the script content to make sure the audio is valid, and matches the script.

Next steps

- [Train your voice model](#)
- [Deploy and use your voice model](#)
- [How to record voice samples](#)

Train your voice model

Article • 01/11/2023 • 12 minutes to read

In this article, you learn how to train a custom neural voice through the Speech Studio portal.

ⓘ Important

Custom Neural Voice training is currently only available in some regions. After your voice model is trained in a supported region, you can [copy](#) it to a Speech resource in another region as needed. See footnotes in the [regions](#) table for more information.

Training duration varies depending on how much data you're training. It takes about 40 compute hours on average to train a custom neural voice. Standard subscription (S0) users can train four voices simultaneously. If you reach the limit, wait until at least one of your voice models finishes training, and then try again.

ⓘ Note

Although the total number of hours required per [training method](#) will vary, the same unit price applies to each. For more information, see the [Custom Neural training pricing details ↗](#).

Choose a training method

After you validate your data files, you can use them to build your Custom Neural Voice model. When you create a custom neural voice, you can choose to train it with one of the following methods:

- [Neural](#): Create a voice in the same language of your training data, select [Neural](#) method.
- [Neural - cross lingual \(Preview\)](#): Create a secondary language for your voice model to speak a different language from your training data. For example, with the `zh-CN` training data, you can create a voice that speaks `en-us`. The language of the training data and the target language must both be one of the [languages that are supported](#) for cross lingual voice training. You don't need to prepare training data in the target language, but your test script must be in the target language.

- **Neural - multi style** (Preview): Create a custom neural voice that speaks in multiple styles and emotions, without adding new training data. Multi-style voices are particularly useful for video game characters, conversational chatbots, audiobooks, content readers, and more. To create a multi-style voice, you just need to prepare a set of general training data (at least 300 utterances), and select one or more of the preset target speaking styles. You can also create up to 10 custom styles by providing style samples (at least 100 utterances per style) as additional training data for the same voice.

The language of the training data must be one of the [languages that are supported](#) for custom neural voice neural, cross-lingual, or multi-style training.

Train your Custom Neural Voice model

To create a custom neural voice in Speech Studio, follow these steps for one of the following [methods](#):

1. Sign in to the [Speech Studio](#).

2. Select **Custom Voice** > Your project name > **Train model** > **Train a new model**.

3. Select **Neural** as the [training method](#) for your model and then select **Next**. To use a different training method, see [Neural - cross lingual](#) or [Neural - multi style](#).

<input checked="" type="radio"/> Select training method <input type="radio"/> Choose data <input type="radio"/> Choose test script <input type="radio"/> Name your voice <input type="radio"/> Review and train	Select the training method for your model <div style="background-color: #e0f2ff; padding: 10px;"> Neural Create a highly realistic voice in the same language of your training data. Start with 300 utterances. </div> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> Neural - cross lingual <i>Preview</i> Create a voice that speaks a different language from your training data. Start with 300 utterances. </div> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> Neural - multi style <i>Preview</i> Create a voice that speaks in multiple styles. Start with 300 utterances for the default and up to 10 preset styles. Customize additional styles with 100 utterances each. </div>
---	---

4. Select a version of the training recipe for your model. The latest version is selected by default. The supported features and training time can vary by

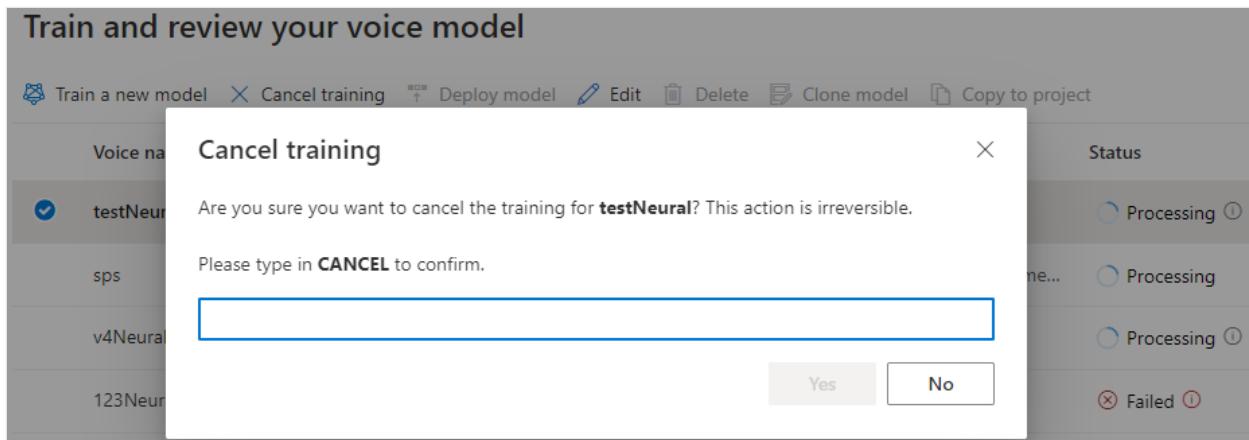
version. Normally, the latest version is recommended for the best results. In some cases, you can choose an older version to reduce training time.

5. Select the data that you want to use for training. Duplicate audio names will be removed from the training. Make sure the data you select don't contain the same audio names across multiple .zip files. Only successfully processed datasets can be selected for training. Check your data processing status if you do not see your training set in the list.
6. Select a speaker file with the voice talent statement that corresponds to the speaker in your training data.
7. Select **Next**.
8. Optionally, you can check the box next to **Add my own test script** and select test scripts to upload. Each training generates 100 sample audio files automatically, to help you test the model with a default script. You can also provide your own test script with up to 100 utterances for the default style. The generated audio files are a combination of the automatic test scripts and custom test scripts. For more information, see [test script requirements](#).
9. Enter a **Name** and **Description** to help you identify the model. Choose a name carefully. The model name will be used as the voice name in your [speech synthesis request](#) via the SDK and SSML input. Only letters, numbers, and a few punctuation characters are allowed. Use different names for different neural voice models.
10. Optionally, enter the **Description** to help you identify the model. A common use of the description is to record the names of the data that you used to create the model.
11. Select **Next**.
12. Review the settings and check the box to accept the terms of use.
13. Select **Submit** to start training the model.

The **Train model** table displays a new entry that corresponds to this newly created model. The status reflects the process of converting your data to a voice model, as described in this table:

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 failed in training. The cause of the failure might be, for example, unseen data problems or network issues.
Canceled	The training for your voice model was canceled.

While the model status is **Processing**, you can select **Cancel training** to cancel your voice model. You're not charged for this canceled training.

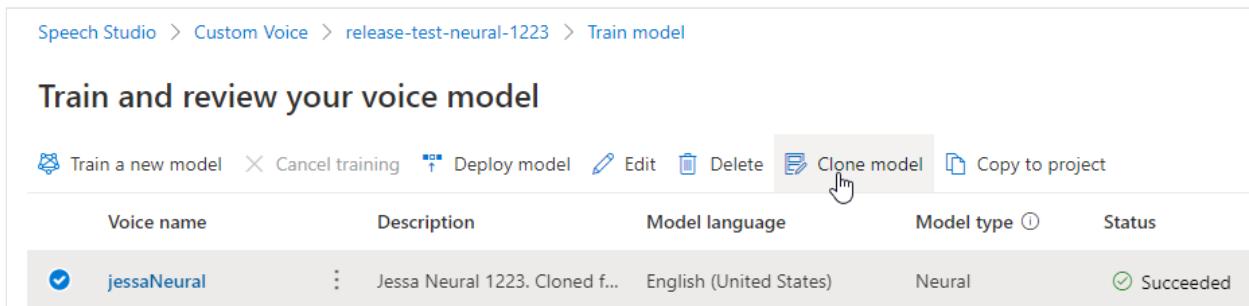


After you finish training the model successfully, you can review the model details and [test the model](#).

You can use the [Audio Content Creation](#) tool in [Speech Studio](#) to create audio and fine-tune your deployed voice. If applicable for your voice, one of multiple styles can also be selected.

Rename your model

If you want to rename the model you built, you can select **Clone model** to create a clone of the model with a new name in the current project.



Enter the new name on the **Clone voice model** window, then select **Submit**. The text 'Neural' will be automatically added as a suffix to your new model name.

Clone voice model

X

Create a clone of your model in the current project, under a new name.

Enter the new name *

New-jessa

Note: You must rename your model for a clone. The text 'Neural' will be automatically added as a suffix to your new model name.

Description

Jessa Neural 1223. Cloned from: Jessa Neural 1223Neural (470efb5b-13b6-4c1d-b860-0b95...)

Note: An additional description will be automatically added to indicate the source model for your clone.

Submit

Cancel



Test your voice model

After your voice model is successfully built, you can use the generated sample audio files to test it before deploying it for use.

The quality of the voice depends on many factors, such as:

- The size of the training data.
- The quality of the recording.
- The accuracy of the transcript file.
- How well the recorded voice in the training data matches the personality of the designed voice for your intended use case.

Select **DefaultTests** under **Testing** to listen to the sample audios. The default test samples include 100 sample audios generated automatically during training to help you test the model. In addition to these 100 audios provided by default, your own test script (at most 100 utterances) provided during training are also added to **DefaultTests** set. You're not charged for the testing with **DefaultTests**.

Testing Training set

+ Add test scripts Delete

File name	Utterances	Created
DefaultTests	100	5/25/2022 3:36 PM

If you want to upload your own test scripts to further test your model, select **Add test scripts** to upload your own test script.

Testing			Training set
File name	Utterances	Created	
DefaultTests	100	5/25/2022 3:36 PM	 Delete

Before uploading test script, check the [test script requirements](#). You'll be charged for the additional testing with the batch synthesis based on the number of billable characters. See [pricing page](#).

On Add test scripts window, select **Browse for a file** to select your own script, then select **Add** to upload it.

Add test scripts

Upload your own text script to further test the model. [View data requirements](#)



Drag and drop.
(.txt file < 1MB)

[Browse for a file](#)

Language 

English (United States)

Note: You will be charged for the additional testing with the batch synthesis (a.k.a., 'Long audio API') based on the number of billable characters. [See pricing. Check how 'billable characters' are calculated.](#)

Add Cancel

Test script requirements

The test script must be a .txt file, less than 1 MB. Supported encoding formats include ANSI/ASCII, UTF-8, UTF-8-BOM, UTF-16-LE, or UTF-16-BE.

Unlike the [training transcription files](#), the test script should exclude the utterance ID (filenames of each utterance). Otherwise, these IDs are spoken.

Here's an example set of utterances in one .txt file:

```
text

This is the waistline, and it's falling.
We have trouble scoring.
It was Janet Maslin.
```

Each paragraph of the utterance results in a separate audio. If you want to combine all sentences into one audio, make them a single paragraph.

ⓘ Note

The generated audio files are a combination of the automatic test scripts and custom test scripts.

Update engine version for your voice model

Azure Text-to-Speech engines are updated from time to time to capture the latest language model that defines the pronunciation of the language. After you've trained your voice, you can apply your voice to the new language model by updating to the latest engine version.

When a new engine is available, you're prompted to update your neural voice model.

Voice name	Description	Model language	Model type ⓘ	Status	Engine version	Training hour ⓘ
InJoon_unitsV3Neural		Korean (Korea)	Neural	✓ Succeeded	2021.08.31.0 ⓘ	35.09
InJoon_conformerNeural		Korean (Korea)	Neural	✓ Succeeded	2021.08.31.0 ⓘ	49.66
InJoonNeural	Unittsv3_0308+hfnet500k	Korean (Korea)	Neural	✓ Succeeded	New engine is available. Update your model to get the best experience with the latest neural TTS technology.	InitialVersion ⓘ 48.53
<input checked="" type="radio"/> InJoonNeural	: unitts2.1 update spk id. Ste...	Korean (Korea)	Neural	✓ Succeeded		
InJoonNeural	unitts2.1. Step 100000-300...	Korean (Korea)	Neural	✓ Succeeded	InitialVersion ⓘ 48.54	

Go to the model details page, select **Update** at the top to display **Update** window.

① New engine is available. Update your model to get the best experience with the latest neural TTS technology. This update comes with no additional cost.

Update

Speech Studio > Custom Voice > Train model > InJoonNeural

InJoonNeural

units2.1 update spk id. Step 100000-300000

Gender	Language	Model type	Utterances ⓘ	Status	Model ID	Created
Male	Korean (Korea)	Neural	1791	Succeeded	0f35ea7c-ff9b-4af9-ac0a-df52f879f450	1/11/2021 7:08 PM

Engine version ⓘ

InitialVersion

Then select **Update** to update your model to the latest engine version.

Speech Studio > Custom Voice > Train model > InJoonNeural

InJoonNeural

units2.1 update spk id. Step 100000-300000

Gender	Language	Model type	Utterances ⓘ	Status	Model ID	Created
Male	Korean (Korea)	Neural	1791	Succeeded	0f35ea7c-ff9b-4af9-ac0a-df52f879f450	1/11/2021 7:08 PM

Engine version ⓘ

InitialVersion

InitialVersion (Default) ⓘ

Check out the samples of the voice created for this version. You can only create additional tests for a default version.

Update

You're not charged for engine update. The previous versions are still kept. You can check all engine versions for the model from **Engine version** drop-down list, or remove one if you don't need it anymore.

Gender	Language	Model type	Utterances ⓘ	Status	Model ID	Created
Male	Korean (Korea)	Neural	1791	Succeeded	0f35ea7c-ff9b-4af9-ac0a-df52f879f450	1/11/2021 7:08 PM

Engine version ⓘ

2022.05.11.0

InitialVersion

2 2022.05.11.0 Default

Check out the samples of the voice created for this version. You can only create additional tests for a default version.

Testing **Training set**

The updated version is automatically set as default. But you can change the default version by selecting a version from the drop-down list and selecting **Set as default**.

Engine version ⓘ

InitialVersion

InitialVersion Set as default

Check out the samples of the voice created for this version. You can only create additional tests for a default version.

Testing **Training set**

If you want to test each engine version of your voice model, you can select a version from the drop-down list, then select **DefaultTests** under **Testing** to listen to the sample audios. If you want to upload your own test scripts to further test your current engine version, first make sure the version is set as default, then follow the [testing steps above](#).

After you've updated the engine version for your voice model, you need to [redeploy this new version](#). You can only deploy the default version.

For more information, [learn more about the capabilities and limits of this feature, and the best practice to improve your model quality](#).

Copy your voice model to another project

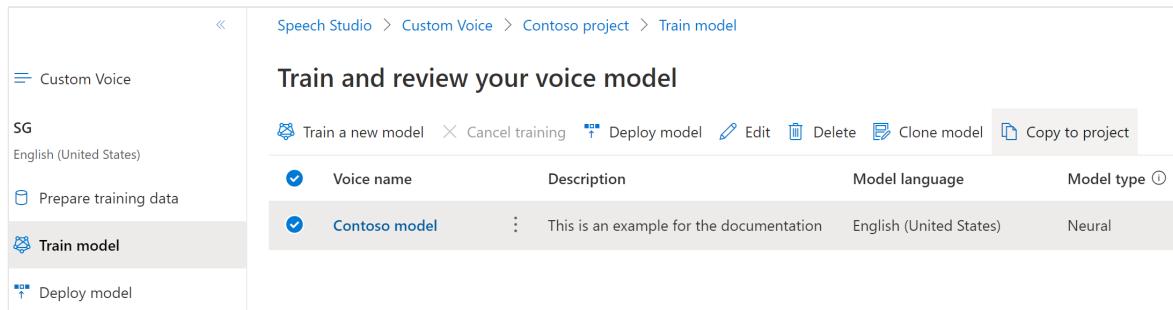
You can copy your voice model to another project for the same region or another region. For example, you can copy a neural voice model that was trained in one region, to a project for another region.

ⓘ Note

Custom Neural Voice training is currently only available in some regions. But you can easily copy a neural voice model from those regions to other regions. For more information, see the [regions for Custom Neural Voice](#).

To copy your custom neural voice model to another project:

1. On the **Train model** tab, select a voice model that you want to copy, and then select **Copy to project**.



2. Select the **Region**, **Speech resource**, and **Project** where you want to copy the model. You must have a speech resource and project in the target region, otherwise you need to create them first.

Copy voice model

X

Copy the voice model selected to another project. Easily deploy the voice to a different region or project.

Region *

East Asia

Speech Resource *

Contoso East Asia resource

Project *

Contoso model copy

+ Create a new project

Cancel

Submit

3. Select **Submit** to copy the model.

4. Select **View model** under the notification message for copy success.

Navigate to the project where you copied the model to [deploy the model copy](#).

Next steps

- [Deploy and use your voice model](#)
- [How to record voice samples](#)
- [Text-to-Speech API reference](#)

Additional resources

Documentation

[Speech Synthesis Markup Language \(SSML\) document structure and events - Speech service - Azure Cognitive Services](#)

Learn about the Speech Synthesis Markup Language (SSML) document structure.

[Text-to-speech quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Voice and sound with Speech Synthesis Markup Language \(SSML\) - Speech service - Azure Cognitive Services](#)

Learn about Speech Synthesis Markup Language (SSML) elements to determine what your output audio will sound like.

[Create a project for Custom Neural Voice - Speech service - Azure Cognitive Services](#)

Learn how to create a Custom Neural Voice project that contains data, models, tests, and endpoints in Speech Studio.

[Speech phonetic alphabets - Speech service - Azure Cognitive Services](#)

This article presents Speech service phonetic alphabet and International Phonetic Alphabet (IPA) examples.

[Text-to-speech API reference \(REST\) - Speech service - Azure Cognitive Services](#)

Learn how to use the REST API to convert text into synthesized speech.

[Get facial position with viseme - Azure Cognitive Services](#)

Speech SDK supports viseme events during speech synthesis, which represent key poses in observed speech, such as the position of the lips, jaw, and tongue when producing a particular phoneme.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Show 5 more](#)

Deploy and use your voice model

Article • 12/01/2022 • 8 minutes to read

After you've successfully created and [trained](#) your voice model, you deploy it to a custom neural voice endpoint.

Use the Speech Studio to [add a deployment endpoint](#) for your custom neural voice. You can use either the Speech Studio or text-to-speech REST API to [suspend or resume](#) a custom neural voice endpoint.

Note

You can create up to 50 endpoints with a standard (S0) Speech resource, each with its own custom neural voice.

To use your custom neural voice, you must specify the voice model name, use the custom URI directly in an HTTP request, and use the same Speech resource to pass through the authentication of the text-to-speech service.

Add a deployment endpoint

To create a custom neural voice endpoint:

1. Sign in to the [Speech Studio](#).
2. Select **Custom Voice** > Your project name > **Deploy model** > **Deploy model**.
3. Select a voice model that you want to associate with this endpoint.
4. Enter a **Name** and **Description** for your custom endpoint.
5. Select **Deploy** to create your endpoint.

After your endpoint is deployed, the endpoint name appears as a link. Select the link to display information specific to your endpoint, such as the endpoint key, endpoint URL, and sample code. When the status of the deployment is **Succeeded**, the endpoint is ready for use.

Application settings

The application settings that you use as REST API [request parameters](#) are available on the **Deploy model** tab in [Speech Studio](#).

Use custom voice in your apps

Voice model name ⓘ

Contoso model

Model created

1/12/2022 2:06 AM

Endpoint key

Endpoint URL

https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId=*****-****-****-****-*****



- The **Endpoint key** shows the Speech resource key the endpoint is associated with. Use the endpoint key as the value of your `Ocp-Apim-Subscription-Key` request header.
- The **Endpoint URL** shows your service region. Use the value that precedes `voice.speech.microsoft.com` as your service region request parameter. For example, use `eastus` if the endpoint URL is <https://eastus.voice.speech.microsoft.com/cognitiveservices/v1>.
- The **Endpoint URL** shows your endpoint ID. Use the value appended to the `?deploymentId=` query parameter as the value of your endpoint ID request parameter.

Use your custom voice

The custom endpoint is functionally identical to the standard endpoint that's used for text-to-speech requests.

One difference is that the `EndpointId` must be specified to use the custom voice via the Speech SDK. You can start with the [text-to-speech quickstart](#) and then update the code with the `EndpointId` and `SpeechSynthesisVoiceName`.

C#

```
var speechConfig = SpeechConfig.FromSubscription(speechKey, speechRegion);
speechConfig.SpeechSynthesisVoiceName = "YourCustomVoiceName";
speechConfig.EndpointId = "YourEndpointId";
```

To use a custom neural voice via [Speech Synthesis Markup Language \(SSML\)](#), specify the model name as the voice name. This example uses the `YourCustomVoiceName` voice.

XML

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis">
  <xml:lang="en-US">
    <voice name="YourCustomVoiceName">
      This is the text that is spoken.
    </voice>
  </speak>
```

Switch to a new voice model in your product

Once you've updated your voice model to the latest engine version, or if you want to switch to a new voice in your product, you need to redeploy the new voice model to a new endpoint. Redeploying new voice model on your existing endpoint is not supported. After deployment, switch the traffic to the newly created endpoint. We recommend that you transfer the traffic to the new endpoint in a test environment first to ensure that the traffic works well, and then transfer to the new endpoint in the production environment. During the transition, you need to keep the old endpoint. If there are some problems with the new endpoint during transition, you can switch back to your old endpoint. If the traffic has been running well on the new endpoint for about 24 hours (recommended value), you can delete your old endpoint.

Note

If your voice name is changed and you are using Speech Synthesis Markup Language (SSML), be sure to use the new voice name in SSML.

Suspend and resume an endpoint

You can suspend or resume an endpoint, to limit spend and conserve resources that aren't in use. You won't be charged while the endpoint is suspended. When you resume an endpoint, you can continue to use the same endpoint URL in your application to synthesize speech.

You can suspend and resume an endpoint in Speech Studio or via the REST API.

Note

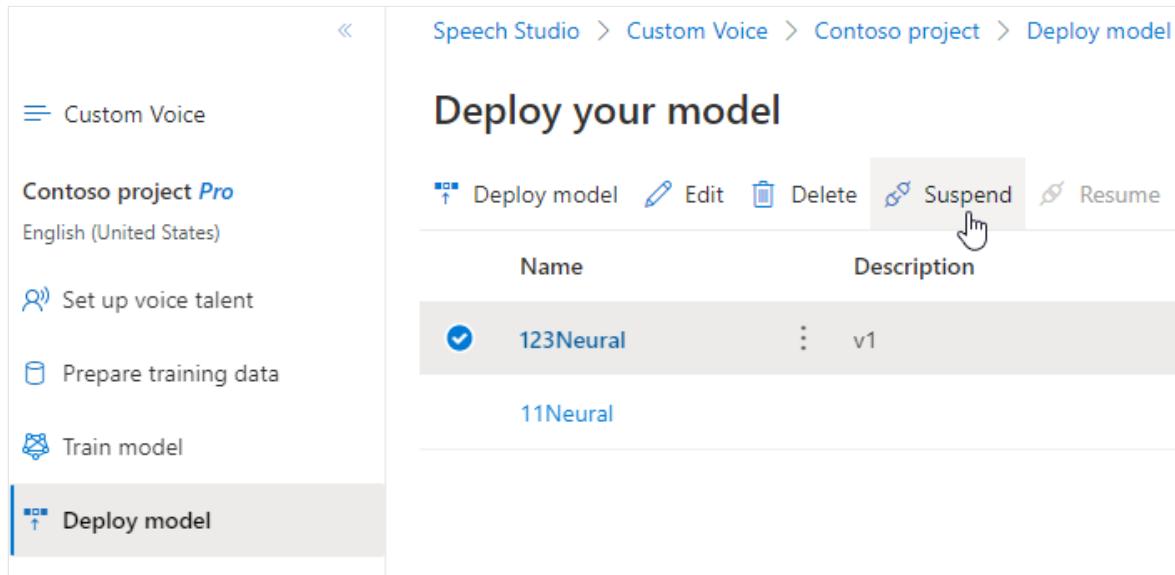
The suspend operation will complete almost immediately. The resume operation completes in about the same amount of time as a new deployment.

Suspend and resume an endpoint in Speech Studio

This section describes how to suspend or resume a custom neural voice endpoint in the Speech Studio portal.

Suspend endpoint

1. To suspend and deactivate your endpoint, select **Suspend** from the Deploy model tab in [Speech Studio](#).



The screenshot shows the 'Deploy your model' page in the Speech Studio portal. On the left, there's a sidebar with options like 'Custom Voice', 'Contoso project Pro', and 'Deploy model' (which is selected and highlighted). The main area shows a table of deployed models:

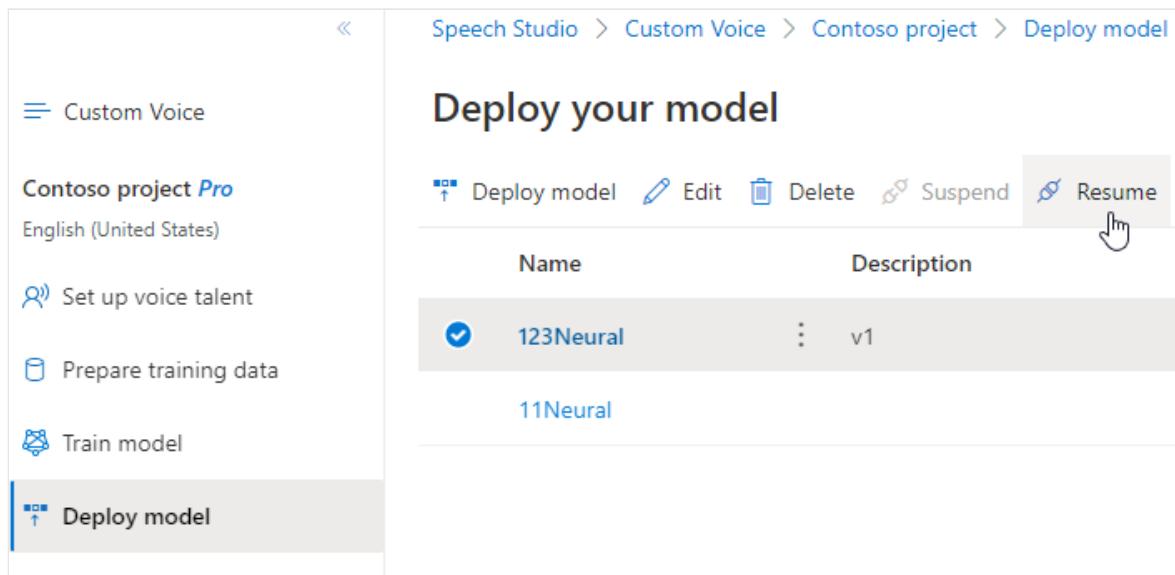
Name	Description
123Neural	v1
11Neural	

A mouse cursor is hovering over the 'Suspend' button in the top navigation bar, which is highlighted with a light gray background.

2. In the dialog box that appears, select **Submit**. After the endpoint is suspended, Speech Studio will show the **Successfully suspended endpoint** notification.

Resume endpoint

1. To resume and activate your endpoint, select **Resume** from the Deploy model tab in [Speech Studio](#).



The screenshot shows the same 'Deploy your model' page in the Speech Studio portal. The 'Deploy model' tab is selected. The table of deployed models is identical to the previous screenshot. A mouse cursor is hovering over the 'Resume' button in the top navigation bar, which is highlighted with a light gray background.

2. In the dialog box that appears, select **Submit**. After you successfully reactivate the endpoint, the status will change from **Suspended** to **Succeeded**.

Suspend and resume endpoint via REST API

This section will show you how to [get](#), [suspend](#), or [resume](#) a custom neural voice endpoint via REST API.

Get endpoint

Get the endpoint by endpoint ID. The operation returns details about an endpoint such as model ID, project ID, and status.

For example, you might want to track the status progression for [suspend](#) or [resume](#) operations. Use the `status` property in the response payload to determine the status of the endpoint.

The possible `status` property values are:

Status	Description
<code>NotStarted</code>	The endpoint hasn't yet been deployed, and it's not available for speech synthesis.
<code>Running</code>	The endpoint is in the process of being deployed or resumed, and it's not available for speech synthesis.
<code>Succeeded</code>	The endpoint is active and available for speech synthesis. The endpoint has been deployed or the resume operation succeeded.
<code>Failed</code>	The endpoint deploy or suspend operation failed. The endpoint can only be viewed or deleted in Speech Studio .
<code>Disabling</code>	The endpoint is in the process of being suspended, and it's not available for speech synthesis.
<code>Disabled</code>	The endpoint is inactive, and it's not available for speech synthesis. The suspend operation succeeded or the resume operation failed.

Tip

If the status is `Failed` or `Disabled`, check `properties.error` for a detailed error message. However, there won't be error details if the status is `Disabled` due to a successful suspend operation.

Get endpoint example

For information about endpoint ID, region, and Speech resource key parameters, see [request parameters](#).

HTTP example:

HTTP

```
GET api/texttospeech/v3.0/endpoints/<YourEndpointId> HTTP/1.1
Ocp-Apim-Subscription-Key: YourResourceKey
Host: <YourResourceRegion>.customvoice.api.speech.microsoft.com
```

cURL example:

Console

```
curl -v -X GET
"https://<YourResourceRegion>.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/endpoints/<YourEndpointId>" -H "Ocp-Apim-Subscription-Key:<YourResourceKey>"
```

Response header example:

Status code: 200 OK

Response body example:

JSON

```
{
  "model": {
    "id": "a92aa4b5-30f5-40db-820c-d2d57353de44"
  },
  "project": {
    "id": "ffc87aba-9f5f-4bfa-9923-b98186591a79"
  },
  "properties": {},
  "status": "Succeeded",
  "lastActionDateTime": "2019-01-07T11:36:07Z",
  "id": "e7ffdf12-17c7-4421-9428-a7235931a653",
  "createdDateTime": "2019-01-07T11:34:12Z",
  "locale": "en-US",
  "name": "Voice endpoint",
  "description": "Example for voice endpoint"
}
```

Suspend endpoint

You can suspend an endpoint to limit spend and conserve resources that aren't in use. You won't be charged while the endpoint is suspended. When you resume an endpoint, you can use the same endpoint URL in your application to synthesize speech.

You suspend an endpoint with its unique deployment ID. The endpoint status must be `Succeeded` before you can suspend it.

Use the [get endpoint](#) operation to poll and track the status progression from `Succeeded`, to `Disabling`, and finally to `Disabled`.

Suspend endpoint example

For information about endpoint ID, region, and Speech resource key parameters, see [request parameters](#).

HTTP example:

HTTP

```
POST api/texttospeech/v3.0/endpoints/<YourEndpointId>/suspend HTTP/1.1
Ocp-Apim-Subscription-Key: YourResourceKey
Host: <YourResourceRegion>.customvoice.api.speech.microsoft.com
Content-Type: application/json
Content-Length: 0
```

CURL example:

Console

```
curl -v -X POST
"https://<YourResourceRegion>.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/endpoints/<YourEndpointId>/suspend" -H "Ocp-Apim-Subscription-Key: <YourResourceKey >" -H "content-type: application/json" -H "content-length: 0"
```

Response header example:

Status code: 202 Accepted

For more information, see [response headers](#).

Resume endpoint

When you resume an endpoint, you can use the same endpoint URL that you used before it was suspended.

You resume an endpoint with its unique deployment ID. The endpoint status must be `Disabled` before you can resume it.

Use the [get endpoint](#) operation to poll and track the status progression from `Disabled`, to `Running`, and finally to `Succeeded`. If the resume operation failed, the endpoint status will be `Disabled`.

Resume endpoint example

For information about endpoint ID, region, and Speech resource key parameters, see [request parameters](#).

HTTP example:

HTTP

```
POST api/texttospeech/v3.0/endpoints/<YourEndpointId>/resume HTTP/1.1
Ocp-Apim-Subscription-Key: YourResourceKey
Host: <YourResourceRegion>.customvoice.api.speech.microsoft.com
Content-Type: application/json
Content-Length: 0
```

cURL example:

Console

```
curl -v -X POST
"https://<YourResourceRegion>.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/endpoints/<YourEndpointId>/resume" -H "Ocp-Apim-Subscription-Key: <YourResourceKey >" -H "content-type: application/json" -H "content-length: 0"
```

Response header example:

```
Status code: 202 Accepted
```

For more information, see [response headers](#).

Parameters and response codes

Request parameters

You use these request parameters with calls to the REST API. See [application settings](#) for information about where to get your region, endpoint ID, and Speech resource key in Speech Studio.

Name	Location	Required	Type	Description
YourResourceRegion	Path	True	string	The Azure region the endpoint is associated with.
YourEndpointId	Path	True	string	The identifier of the endpoint.
Ocp-Apim-Subscription-Key	Header	True	string	The Speech resource key the endpoint is associated with.

Response headers

Status code: 202 Accepted

Name	Type	Description
Location	string	The location of the endpoint that can be used as the full URL to get endpoint.
Retry-After	string	The total seconds of recommended interval to retry to get endpoint status.

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.
202	Accepted	The request has been accepted and is being processed.
400	Bad Request	The value of a parameter is invalid, or a required parameter is missing, empty, or null. One common issue is a header that is too long.

HTTP status code	Description	Possible reason
401	Unauthorized	The request isn't authorized. Check to make sure your Speech resource key or token is valid and in the correct region.
429	Too Many Requests	You've exceeded the quota or rate of requests allowed for your Speech resource.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

Next steps

- [How to record voice samples](#)
- [Text-to-Speech API reference](#)
- [Batch synthesis](#)

Training data for Custom Neural Voice

Article • 10/27/2022 • 9 minutes to read

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.

💡 Tip

To create a voice for production use, we recommend you use a professional recording studio and voice talent. For more information, see [record voice samples to create a custom neural voice](#).

Types of training data

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 neural voice training with available audio files, short or long, with or without transcripts.

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	Extra processing required
Individual utterances + matching transcript	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.

Data type	Description	When to use	Extra processing required
Long audio + transcript	A collection (.zip) of long, unsegmented audio files (.wav or .mp3, longer than 20 seconds, at most 1000 audio files), paired with a collection (.zip) of transcripts that contains all spoken words.	You have audio files and matching transcripts, but they aren't segmented into utterances.	Segmentation (using batch transcription). Audio format transformation wherever required.
Audio only (Preview)	A collection (.zip) of audio files (.wav or .mp3, at most 1000 audio files) without a transcript.	You only have audio files available, without transcripts.	Segmentation + transcript generation (using batch transcription). Audio format transformation wherever required.

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 500 zip files 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 model, 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.

For data format examples, refer to the sample training set on [GitHub](#). The sample training set includes the sample script and the associated audio.

Audio data for Individual utterances + matching transcript

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 aren't supported, except for the Chinese-English bi-lingual. Each audio file must have a unique filename with the filename extension .wav.

Follow these guidelines when preparing audio.

Property	Value
File format	RIFF (.wav), grouped into a .zip file
File name	File name characters supported by Windows OS, with .wav extension. The characters \ / : * ? " < > aren't allowed. It can't start or end with a space, and can't start with a dot. No duplicate file names allowed.
Sampling rate	When creating a custom neural voice, 24,000 Hz is required.
Sample format	PCM, at least 16-bit
Audio length	Shorter than 15 seconds
Archive format	.zip
Maximum archive size	2048 MB

ⓘ Note

The default sampling rate for a custom neural voice is 24,000 Hz. Audio 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. Your audio files with a sampling rate higher than 16,000 Hz and lower than 24,000 Hz will be up-sampled to 24,000 Hz to train a neural voice. It's recommended that you should use a sample rate of 24,000 Hz for your training data.

Transcription data for Individual utterances + matching transcript

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 and ASCII encoding aren't supported.
# of utterances per line	One - 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.

Long audio + transcript (Preview)

ⓘ Note

For **Long audio + transcript (Preview)**, only these languages are supported: Chinese (Mandarin, Simplified), English (India), English (United Kingdom), English (United States), French (France), German (Germany), Italian (Italy), Japanese (Japan), Portuguese (Brazil), and Spanish (Mexico).

In some cases, you may not have segmented audio available. The Speech Studio can help you segment long audio files and create transcriptions. The long-audio segmentation service will use the [Batch Transcription API](#) feature of speech-to-text.

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.

ⓘ Note

This service will be charged toward your speech-to-text subscription usage. The long-audio segmentation service is only supported with standard (S0) Speech resources.

Audio data for Long audio + transcript

Follow these guidelines when preparing audio for segmentation.

Property	Value
File format	RIFF (.wav) or .mp3, grouped into a .zip file
File name	File name characters supported by Windows OS, with .wav extension. The characters \ / : * ? " < > aren't allowed. It can't start or end with a space, and can't start with a dot. No duplicate file names allowed.
Sampling rate	When creating a custom neural voice, 24,000 Hz is required.
Sample format	RIFF(.wav): PCM, at least 16-bit mp3: at least 256 KBps bit rate
Audio length	Longer than 20 seconds
Archive format	.zip
Maximum archive size	2048 MB, at most 1000 audio files included

ⓘ Note

The default sampling rate for a custom neural voice is 24,000 Hz. Audio files with a sampling rate lower than 16,000 Hz will be rejected. Your audio files with a sampling rate higher than 16,000 Hz and lower than 24,000 Hz will be up-sampled to 24,000 Hz to train a neural voice. It's recommended that you should use a sample rate of 24,000 Hz for your training data.

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.

Transcription data for Long audio + transcript

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	ANSI, ASCII, UTF-8, UTF-8-BOM, UTF-16-LE, or UTF-16-BE. For zh-CN, ANSI and ASCII encoding aren't supported.
# 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've uploaded a zip file containing an audio file named 'kingstory.wav', 45 seconds long, and another one named 'queenstory.mp3', 200 seconds long. You'll need to upload another zip file containing two transcripts, one named 'kingstory.txt', the other one 'queenstory.txt'. Within each plain text file, you'll provide the full correct transcription for the matching audio.

After your dataset is successfully uploaded, we'll 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 (Preview)

 Note

For Audio only (Preview), only these languages are supported: Chinese (Mandarin, Simplified), English (India), English (United Kingdom), English (United States), French (France), German (Germany), Italian (Italy), Japanese (Japan), Portuguese (Brazil), and Spanish (Mexico).

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 be charged 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) or .mp3, grouped into a .zip file
File name	File name characters supported by Windows OS, with .wav extension. The characters \ / : * ? " < > aren't allowed. It can't start or end with a space, and can't start with a dot. No duplicate file names allowed.
Sampling rate	When creating a custom neural voice, 24,000 Hz is required.
Sample format	RIFF(.wav): PCM, at least 16-bit mp3: at least 256 KBps bit rate
Audio length	No limit
Archive format	.zip
Maximum archive size	2048 MB, at most 1000 audio files included

ⓘ Note

The default sampling rate for a custom neural voice is 24,000 Hz. Your audio files with a sampling rate higher than 16,000 Hz and lower than 24,000 Hz will be up-sampled to 24,000 Hz to train a neural voice. It's recommended that you should use a sample rate of 24,000 Hz for your training data.

All audio files should be grouped into a zip file. Once your dataset is successfully uploaded, we'll 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

- [Train your voice model](#)
- [Deploy and use your voice model](#)
- [How to record voice samples](#)

Recording voice samples for Custom Neural Voice

Article • 12/29/2022 • 23 minutes to read

This article provides you instructions on preparing high-quality voice samples for creating a professional voice model using the Custom Neural Voice Pro project.

Creating a high-quality production custom neural voice from scratch isn't a casual undertaking. The central component of a custom neural 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 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.

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.

Tips for preparing data for a high-quality voice

A highly-natural custom neural voice depends on several factors, like the quality and size of your training data.

The quality of your training data is a primary factor. For example, in the same training set, consistent volume, speaking rate, speaking pitch, and speaking style are essential to create a high-quality custom neural voice. You should also avoid background noise in the recording and make sure the script and recording match. To ensure the quality of your data, you need to follow [script selection criteria](#) and [recording requirements](#).

Regarding the size of the training data, in most cases you can build a reasonable custom neural voice with 500 utterances. According to our tests, adding more training data in most languages does not necessarily improve naturalness of the voice itself (tested using the MOS score), however, with more training data that covers more word instances, you have higher possibility to reduce the DSAT (dis-satisfied part of the speech, for example, the glitches) ratio for the voice.

In some cases, you may want a voice persona with unique characteristics. For example, a cartoon persona needs a voice with a special speaking style, or a voice that is very dynamic in intonation. For such cases, we recommend that you prepare at least 1000 (preferably 2000) utterances, and record them at a professional recording studio. To

learn more about how to improve the quality of your voice model, see [characteristics and limitations for using Custom Neural Voice](#).

Voice recording roles

There are four basic roles in a custom neural voice recording project:

Role	Purpose
Voice talent	This person's voice will form the basis of the custom neural 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 Speech Studio

An individual may fill more than one role. This guide assumes that you'll be 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 recording session, and can be performed by the director or the recording engineer.

Choose your voice talent

Actors with experience in voiceover, voice character work, announcing or newsreading make good voice talent. Choose voice talent whose natural voice you like. It's 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. The single most important factor for choosing voice talent is consistency. Your recordings for the same voice style 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 must be able to speak with consistent rate, volume level, pitch, and tone with clear dictation. They also need to be able to control their pitch variation, emotional affect, and speech mannerisms. Recording voice samples can be more fatiguing than other kinds of voice work, so most voice talent can usually only record for two or three hours a day. Limit sessions to three or four days a week, with a day off in-between if possible.

Work with your voice talent to develop a persona that defines the overall sound and emotional tone of the custom neural voice, making sure to pinpoint what "neutral"

sounds like for that persona. Using the Custom Neural Voice capability, you can train a model that speaks with emotion, so define the speaking styles of your persona and ask your voice talent to read the script in a way that resonates with the styles you want.

For example, a persona with a naturally upbeat personality would carry a note of optimism even when they speak neutrally. However, this 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 neural 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. Building a custom neural voice requires at least 300 recorded utterances as training data.

The utterances in your script can come from anywhere: fiction, non-fiction, transcripts of speeches, news reports, and anything else available in printed form. 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, the same kind of source, or have anything to do with each other. However, if you use set phrases (for example, "You have successfully logged in") in your speech application, make sure to include them in your script. It will give your custom neural voice a better chance of pronouncing those phrases well.

We recommend the recording scripts include both general sentences and domain-specific sentences. For example, if you plan to record 2,000 sentences, 1,000 of them could be general sentences, another 1,000 of them could be sentences from your target domain or the use case of your application.

We provide [sample scripts in the 'General', 'Chat' and 'Customer Service' domains for each language](#) to help you prepare your recording scripts. You can use these Microsoft shared scripts for your recordings directly or use them as a reference to create your own.

Script selection criteria

Below are some general guidelines that you can follow to create a good corpus (recorded audio samples) for custom neural voice training.

- Balance your script to cover different sentence types in your domain including statements, questions, exclamations, long sentences, and short sentences.

Each sentence should contain 4 words to 30 words, and no duplicate sentences should be included in your script.

For how to balance the different sentence types, refer to the following table:

Sentence types	Coverage
Statement sentences	Statement sentences should be 70-80% of the script.
Question sentences	Question sentences should be about 10%-20% of your domain script, including 5%-10% of rising and 5%-10% of falling tones.
Exclamation sentences	Exclamation sentences should be about 10%-20% of your script.
Short word/phrase	Short word/phrase scripts should be about 10% of total utterances, with 5 to 7 words per case.

① Note

Short words/phrases should be separated with a commas. They help remind your voice talent to pause briefly when reading them.

Best practices include:

- Balanced coverage for Parts of Speech, like verbs, nouns, adjectives, and so on.
- Balanced coverage for pronunciations. Include all letters from A to Z so the Text-to-Speech engine learns how to pronounce each letter in your style.
- Readable, understandable, common-sense scripts for the speaker to read.
- Avoid too many similar patterns for words/phrases, like "easy" and "easier".
- Include different formats of numbers: address, unit, phone, quantity, date, and so on, in all sentence types.
- Include spelling sentences if it's something your custom neural voice will be used to read. For example, "The spelling of Apple is A P P L E".

- Don't put multiple sentences into one line/one utterance. Separate each line by utterance.
- Make sure the sentence is clean. Generally, don't include too many non-standard words like numbers or abbreviations as they're usually hard to read. Some applications may require the reading of many numbers or acronyms. In these cases, you can include these words, but normalize them in their spoken form.

Below are some best practices for example:

- For lines with abbreviations, instead of "BTW", write "by the way".
- For lines with digits, instead of "911", write "nine one one".
- For lines with acronyms, instead of "ABC", write "A B C".

With that, make sure your voice talent pronounces these words in an expected way. Keep your script and recordings matched during the training process.

- Your script should include many different words and sentences with different kinds of sentence lengths, structures, and moods.
- Check the script carefully for errors. If possible, have someone else check it too. When you run through the script with your voice talent, you may catch more mistakes.

Difference between voice talent script and training script

The training script can differ from the voice talent script, especially for scripts that contain digits, symbols, abbreviations, date, and time. Scripts prepared for the voice talent must follow native reading conventions, such as 50% and \$45. The scripts used for training must be normalized to match the audio recording, such as *fifty percent* and *forty-five dollars*.

Note

We provide some example scripts for the voice talent on [GitHub](#). To use the example scripts for training, you must normalize them according to the recordings of your voice talent before uploading the file.

The following table shows the difference between scripts for voice talent and the normalized script for training.

Category	Voice talent script example	Training script example (normalized)
----------	-----------------------------	--------------------------------------

Category	Voice talent script example	Training script example (normalized)
Digits	123	one hundred and twenty-three
Symbols	50%	fifty percent
Abbreviation	ASAP	as soon as possible
Date and time	March 3rd at 5:00 PM	March third at five PM

Typical defects of a script

The script's poor quality can adversely affect the training results. To achieve high-quality training results, it's crucial to avoid defects.

Script defects generally fall into the following categories:

Category	Example
Meaningless content.	"Colorless green ideas sleep furiously."
Incomplete sentences.	<ul style="list-style-type: none"> - "This was my last eve" (no subject, no specific meaning) - "He's obviously already funny (no quote mark in the end, it's not a complete sentence)
Typo in the sentences.	<ul style="list-style-type: none"> - Start with a lower case - No ending punctuation if needed - Misspelling - Lack of punctuation: no period in the end (except news title) - End with symbols, except comma, question, exclamation - Wrong format, such as: <ul style="list-style-type: none"> - 45\$ (should be \$45) - No space or excess space between word/punctuation
Duplication in similar format, one per each pattern is enough.	<ul style="list-style-type: none"> - "Now is 1pm in New York" - "Now is 2pm in New York" - "Now is 3pm in New York" - "Now is 1pm in Seattle" - "Now is 1pm in Washington D.C."
Uncommon foreign words: only commonly used foreign words are acceptable in the script.	In English one might use the French word "faux" in common speech, but a French expression such as "coincer la bulle" would be uncommon.
Emoji or any other uncommon symbols	

Script format

The script is for use during recording sessions, so you can set it up any way you find easy to work with. Create the text file that's required by Speech Studio 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 Microsoft 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 voice talent, one for the recording 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.

Voice talent statement

To train a neural voice, you must [create a voice talent profile](#) with an audio file recorded by the voice talent consenting to the usage of their speech data to train a custom voice model. When preparing your recording script, make sure you include the statement sentence.

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 won't be recognizable in the final product, the custom neural voice. Even so, the legality of using a copyrighted work for this purpose isn't well established. Microsoft can't provide legal advice on this issue; consult your own legal counsel.

Fortunately, it's 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, Project Gutenberg ↗ 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 neural 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's recommended 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's critical that the audio has consistent volume and a high signal-to-noise ratio, while being free of unwanted sounds.

Recording requirements

To achieve high-quality training results, follow the following requirements during recording or data preparation:

- Clear and well pronounced
- Natural speed: not too slow or too fast between audio files.
- Appropriate volume, prosody and break: stable within the same sentence or between sentences, correct break for punctuation.
- No noise during recording
- Fit your persona design
- No wrong accent: fit to the target design
- No wrong pronunciation

You can refer to below specification to prepare for the audio samples as best practice.

Property	Value
File format	*.wav, Mono
Sampling rate	24 KHz
Sample format	16 bit, PCM
Peak volume levels	-3 dB to -6 dB
SNR	> 35 dB

Property	Value
Silence	<ul style="list-style-type: none"> - There should have some silence (recommend 100 ms) at the beginning and ending, but no longer than 200 ms - Silence between words or phrases < -30 dB - Silence in the wave after last word is spoken <-60 dB
Environment noise, echo	<ul style="list-style-type: none"> - The level of noise at start of the wave before speaking < -70 dB

ⓘ Note

You can record at higher sampling rate and bit depth, for example in the format of 48 KHz 24 bit PCM. During the custom neural voice training, we'll down sample it to 24 KHz 16 bit PCM automatically.

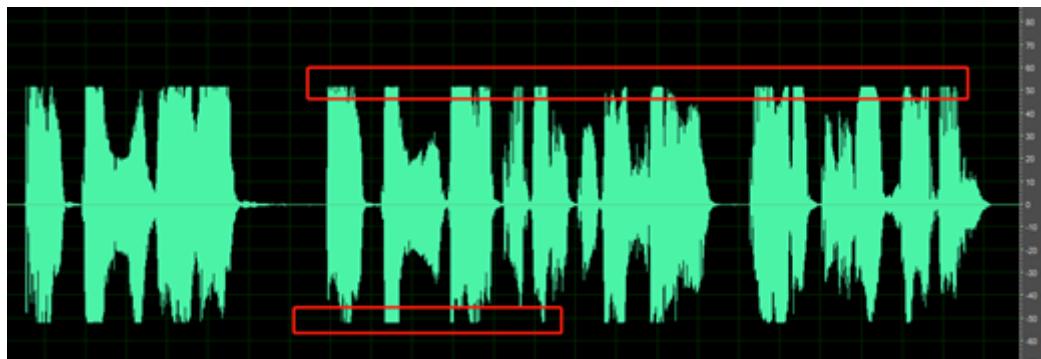
A higher signal-to-noise ratio (SNR) indicates lower noise in your audio. You can typically reach a 35+ SNR by recording at professional studios. Audio with an SNR below 20 can result in obvious noise in your generated voice.

Consider re-recording any utterances with low pronunciation scores or poor signal-to-noise ratios. If you can't re-record, consider excluding those utterances from your data.

Typical audio errors

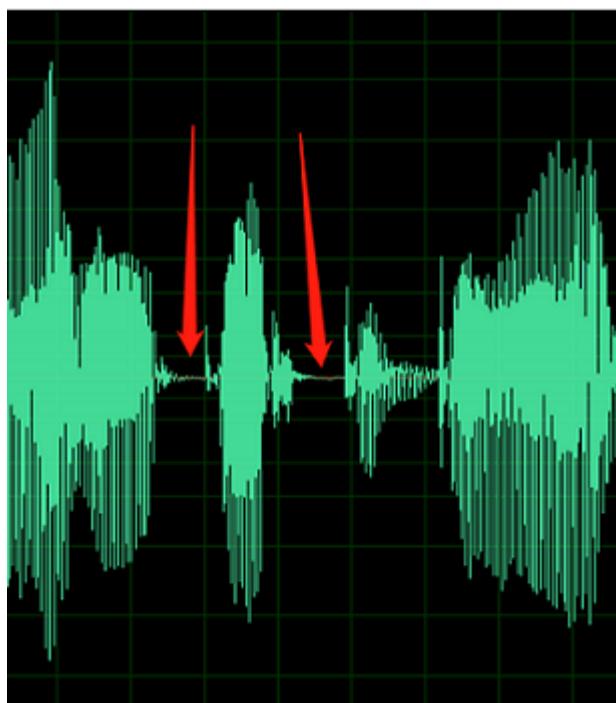
For high-quality training results, avoiding audio errors is highly recommended. Audio errors can be usually within following categories:

- Audio file name doesn't match the script ID.
- WAR file has an invalid format and can't be read.
- Audio sampling rate is lower than 16 KHz. It's recommended that the .wav file sampling rate be equal or higher than 24 KHz for high-quality neural voice.
- Volume peak isn't within the range of -3 dB (70% of max volume) to -6 dB (50%).
- Waveform overflow: the waveform is cut at its peak value and is thus not complete.

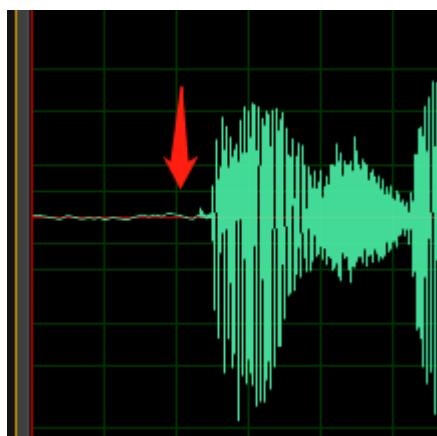


- The silent parts of the recording aren't clean; you can hear sounds such as ambient noise, mouth noise and echo.

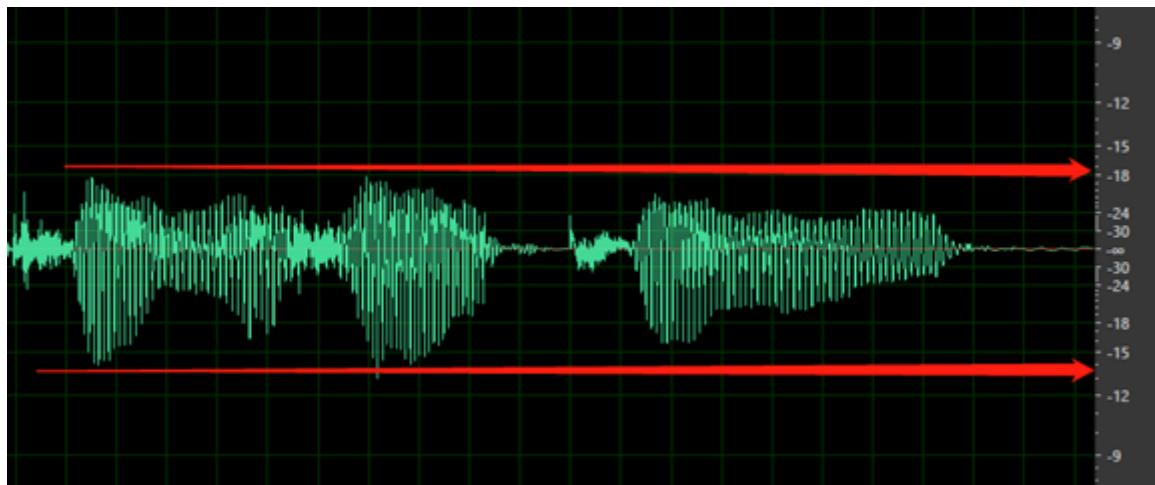
For example, below audio contains the environment noise between speeches.



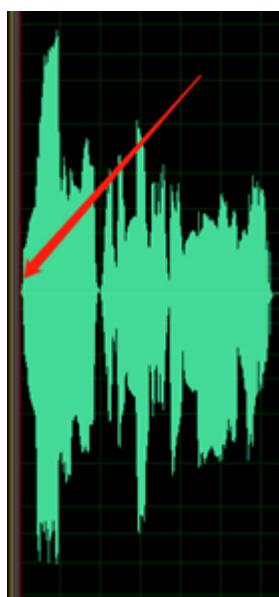
Below sample contains signs of DC offset or echo.



- The overall volume is too low. Your data will be tagged as an issue if the volume is lower than -18 dB (10% of max volume). Make sure all audio files should be consistent at the same level of volume.



- No silence before the first word or after the last word. Also, the start or end silence shouldn't be longer than 200 ms or shorter than 100 ms.



Do it yourself

If you want to make the recording yourself, instead of 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 known 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 recording 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*).

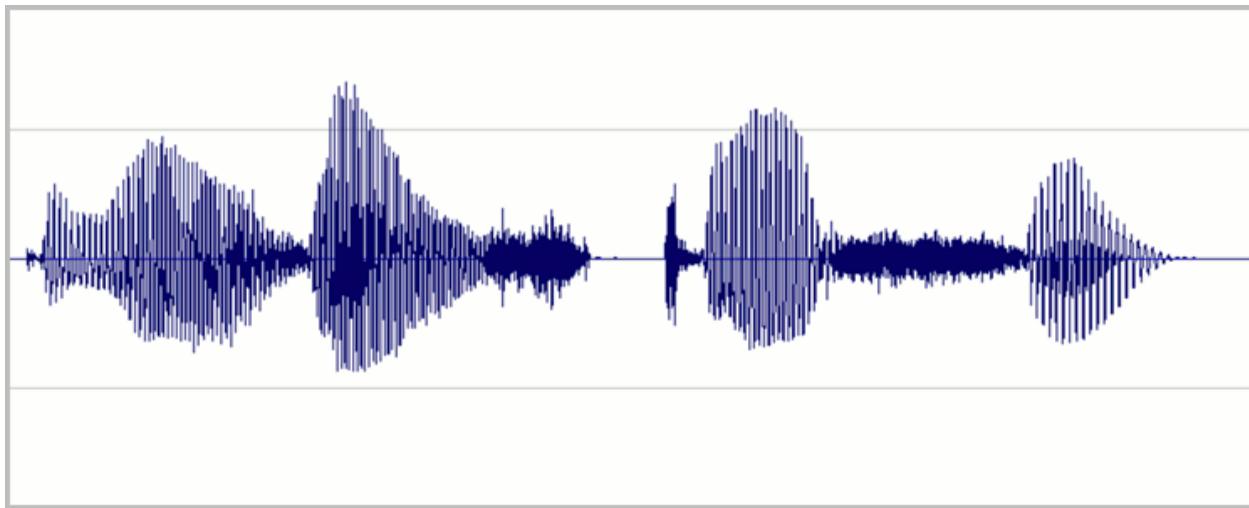
The recording should contain as little noise as possible, with a goal of -80 dB.

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 don't 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'll down-sample your audio to 24 KHz 16-bit before you submit it to Speech Studio. Still, it pays to have a high-quality original recording in the event that 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 shouldn't 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 shouldn't 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. Play it a few times for the talent and have them repeat it each time until they're 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 approximately five seconds of silence before the first recording to capture the "room tone". This practice helps Speech Studio compensate for noise in the recordings.

💡 Tip

All you 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 Speech Studio.

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 neural voice. It might be more expedient to simply note any utterances with issues, exclude them from your dataset, and see how your custom neural 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. 24 KHz 16-bit is common and desirable. The default sampling rate for a custom neural voice is 24 KHz. It's recommended that you should use a sample rate of 24 KHz for your training data. Higher sampling rates, such as 96 KHz, are generally not needed.

Speech Studio 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.

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've done so.

Convert each file to 16 bits and a sample rate of 24 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. [Train your voice model](#) 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 neural voice.

[Train your voice model](#)

Speech synthesis with the Audio Content Creation tool

Article • 01/11/2023 • 10 minutes to read

You can use the [Audio Content Creation](#) tool in Speech Studio for text-to-speech synthesis without writing any code. You can use the output audio as-is, or as a starting point for further customization.

Build highly natural audio content for a variety of scenarios, such as audiobooks, news broadcasts, video narrations, and chat bots. With Audio Content Creation, you can efficiently fine-tune text-to-speech voices and design customized audio experiences.

The tool is based on [Speech Synthesis Markup Language \(SSML\)](#). It allows you to adjust text-to-speech output attributes in real time or batch synthesis, such as voice characters, voice styles, speaking speed, pronunciation, and prosody.

- No-code approach: You can use the Audio Content Creation tool for text-to-speech synthesis without writing any code. The output audio might be the final deliverable that you want. For example, you can use the output audio for a podcast or a video narration.
- Developer-friendly: You can listen to the output audio and adjust the SSML to improve speech synthesis. Then you can use the [Speech SDK](#) or [Speech CLI](#) to integrate the SSML into your applications. For example, you can use the SSML for building a chat bot.

You have easy access to a broad portfolio of [languages and voices](#). These voices include state-of-the-art prebuilt neural voices and your custom neural voice, if you've built one.

To learn more, view the Audio Content Creation tutorial video [on YouTube](#).

Get started

The Audio Content Creation tool in Speech Studio is free to access, but you'll pay for Speech service usage. To work with the tool, you need to sign in with an Azure account and create a Speech resource. For each Azure account, you have free monthly speech quotas, which include 0.5 million characters for prebuilt neural voices (referred to as *Neural* on the [pricing page](#)). The monthly allotted amount is usually enough for a small content team of around 3-5 people.

The next sections cover how to create an Azure account and get a Speech resource.

Step 1: Create an Azure account

To work with Audio Content Creation, you need a [Microsoft account](#) and an [Azure account](#).

The [Azure portal](#) is the centralized place for you to manage your Azure account. You can create the Speech resource, manage the product access, and monitor everything from simple web apps to complex cloud deployments.

Step 2: Create a Speech resource

After you sign up for the Azure account, you need to create a Speech resource in your Azure account to access Speech services. Create a Speech resource on the [Azure portal](#). For more information, see [Create a new Azure Cognitive Services resource](#).

It takes a few moments to deploy your new Speech resource. After the deployment is complete, you can start using the Audio Content Creation tool.

Note

If you plan to use neural voices, make sure that you create your resource in a region that supports neural voices.

Step 3: Sign in to Audio Content Creation with your Azure account and Speech resource

1. After you get the Azure account and the Speech resource, sign in to [Speech Studio](#), and then select **Audio Content Creation**.
2. Select the Azure subscription and the Speech resource you want to work with, and then select **Use resource**.

The next time you sign in to Audio Content Creation, you're linked directly to the audio work files under the current Speech resource. You can check your Azure subscription details and status in the [Azure portal](#).

If you don't have an available Speech resource and you're the owner or admin of an Azure subscription, you can create a Speech resource in Speech Studio by selecting **Create a new resource**.

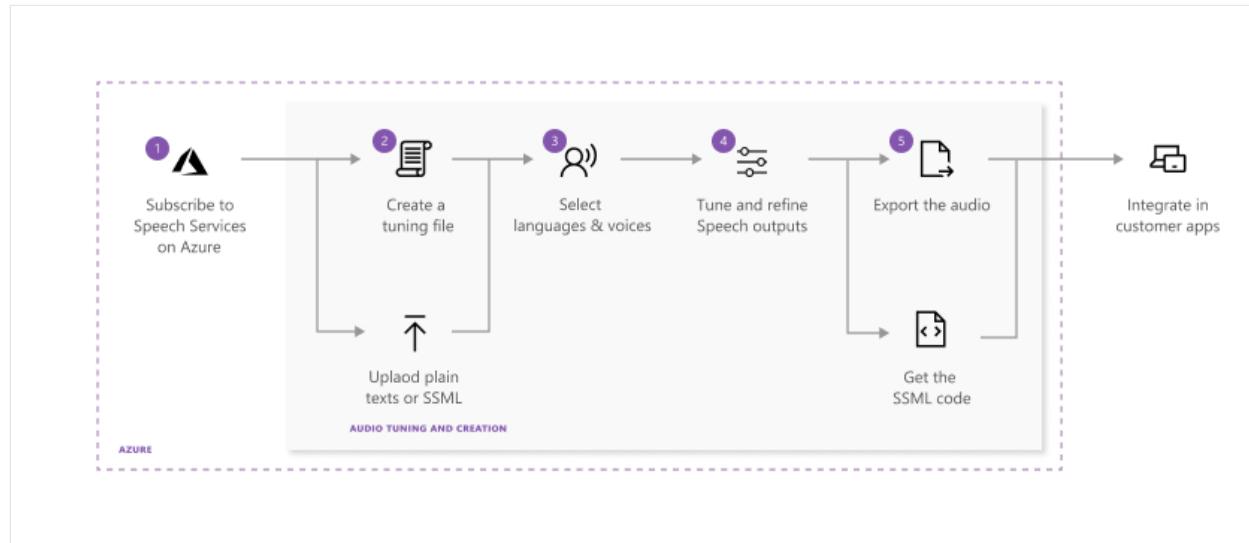
If you have a user role for a certain Azure subscription, you might not have permissions to create a new Speech resource. To get access, contact your admin.

To switch your Speech resource at any time, select **Settings** at the top of the page.

To switch directories, select **Settings** or go to your profile.

Use the tool

The following diagram displays the process for fine-tuning the text-to-speech outputs.



Each step in the preceding diagram is described here:

1. Choose the Speech resource you want to work with.
2. [Create an audio tuning file](#) by using plain text or SSML scripts. Enter or upload your content into Audio Content Creation.
3. Choose the voice and the language for your script content. Audio Content Creation includes all of the [prebuilt text-to-speech voices](#). You can use prebuilt neural voices or a custom neural voice.

(!) Note

Gated access is available for Custom Neural Voice, which allows you to create high-definition voices that are similar to natural-sounding speech. For more information, see [Gating process](#).

4. Select the content you want to preview, and then select **Play** (triangle icon) to preview the default synthesis output.

If you make any changes to the text, select the **Stop** icon, and then select **Play** again to regenerate the audio with changed scripts.

Improve the output by adjusting pronunciation, break, pitch, rate, intonation, voice style, and more. For a complete list of options, see [Speech Synthesis Markup Language](#).

For more information about fine-tuning speech output, view the [How to convert Text to Speech using Microsoft Azure AI voices](#) video.

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.

Create an audio tuning file

You can get your content into the Audio Content Creation tool in either of two ways:

- **Option 1**

1. Select **New > Text file** to create a new audio tuning file.
2. Enter or paste your content into the editing window. The allowable number of characters for each file is 20,000 or fewer. If your script contains more than 20,000 characters, you can use Option 2 to automatically split your content into multiple files.
3. Select **Save**.

- **Option 2**

1. Select **Upload > Text file** to import one or more text files. Both plain text and SSML are supported.

If your script file is more than 20,000 characters, split the content by paragraphs, by characters, or by regular expressions.

2. When you upload your text files, make sure that they meet these requirements:

Property	Description
File format	Plain text (.txt)* SSML text (.txt)** Zip files aren't supported.

Property	Description
Encoding format	UTF-8
File name	Each file must have a unique name. Duplicate files aren't supported.
Text length	Character limit is 20,000. If your files exceed the limit, split them according to the instructions in the tool.
SSML restrictions	Each SSML file can contain only a single piece of SSML.

* Plain text example:

txt

Welcome to use Audio Content Creation to customize audio output for your products.

** SSML text example:

XML

```
<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="en-US-JennyNeural">
        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. Select **Export** to create an audio creation task.

We recommend **Export to Audio library** to easily store, find, and search audio output in the cloud. You can better integrate with your applications through Azure blob storage. You can also download the audio to your local disk directly.

2. Choose the output format for your tuned audio. The **supported audio formats and sample rates** are listed in the following table:

Format	8 kHz sample rate	16 kHz sample rate	24 kHz sample rate	48 kHz sample rate
wav	riff-8khz-16bit-mono-pcm	riff-16khz-16bit-mono-pcm	riff-24khz-16bit-mono-pcm	riff-48khz-16bit-mono-pcm
mp3	N/A	audio-16khz-128kbitrate-mono-mp3	audio-24khz-160kbitrate-mono-mp3	audio-48khz-192kbitrate-mono-mp3

3. To view the status of the task, select the **Task list** 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** pane.

5. Select the file you want to download and **Download**.

Now you're ready to use your custom tuned audio in your apps or products.

Add or remove Audio Content Creation users

If more than one user wants to use Audio Content Creation, you can grant them access to the Azure subscription and the Speech resource. If you add users to an Azure subscription, they can access all the resources under the Azure subscription. But if you add users to a Speech resource only, they'll have access only to the Speech resource and not to other resources under this Azure subscription. Users with access to the Speech resource can use the Audio Content Creation tool.

The users you grant access to need to set up a [Microsoft account](#). If they don't have a Microsoft account, they can create one in just a few minutes. They can use their existing email and link it to a Microsoft account, or they can create and use an Outlook email address as a Microsoft account.

Add users to a Speech resource

To add users to a Speech resource so that they can use Audio Content Creation, do the following:

1. In the [Azure portal](#), select **All services**.

2. Then select the **Cognitive Services**, and navigate to your specific Speech resource.

Note

You can also set up Azure RBAC for whole resource groups, subscriptions, or management groups. Do this by selecting the desired scope level and then navigating to the desired item (for example, selecting **Resource groups** and then clicking through to your wanted resource group).

3. Select **Access control (IAM)** on the left navigation pane.
4. Select **Add -> Add role assignment**.
5. On the **Role** tab on the next screen, select a role you want to add (in this case, **Owner**).
6. On the **Members** tab, enter a user's email address and select the user's name in the directory. The email address must be linked to a Microsoft account that's trusted by Azure Active Directory. Users can easily sign up for a [Microsoft account](#) by using their personal email address.
7. On the **Review + assign** tab, select **Review + assign** to assign the role.

Here is what happens next:

An email invitation is automatically sent to users. They can accept it by selecting **Accept invitation > Accept to join Azure** in their email. They're then redirected to the Azure portal. They don't need to take further action in the Azure portal. After a few moments, users are assigned the role at the Speech resource scope, which gives them access to this Speech resource. If users don't receive the invitation email, you can search for their account under **Role assignments** and go into their profile. Look for **Identity > Invitation accepted**, and select **(manage)** to resend the email invitation. You can also copy and send the invitation link to them.

Users now visit or refresh the [Audio Content Creation](#) product page, and sign in with their Microsoft account. They select **Audio Content Creation** block among all speech products. They choose the Speech resource in the pop-up window or in the settings at the upper right.

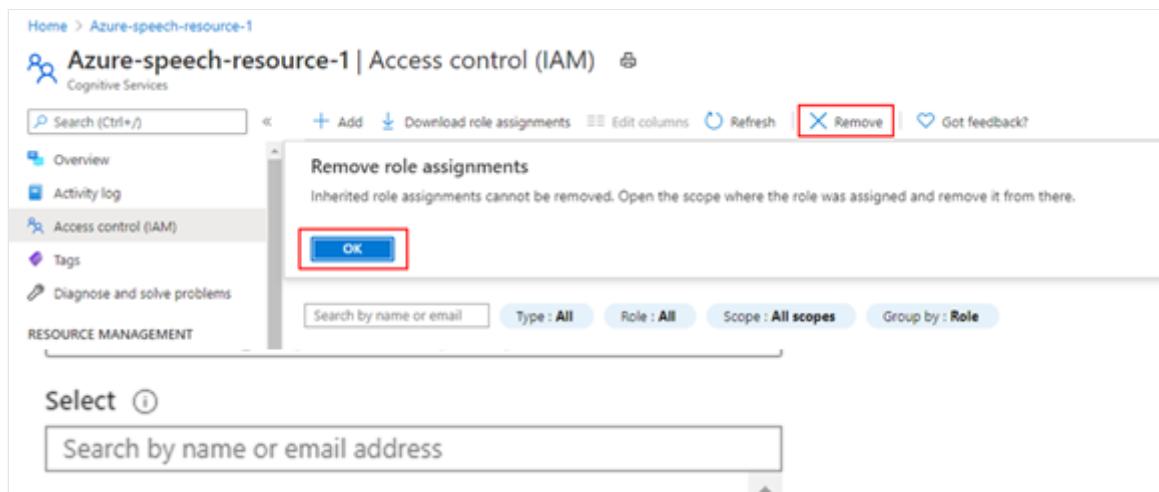
If they can't find the available Speech resource, they can check to ensure that they're in the right directory. To do so, they select the account profile at the upper right and then select **Switch** next to **Current directory**. If there's more than one directory available, it means they have access to multiple directories. They can switch to different directories and go to **Settings** to see whether the right Speech resource is available.

Users who are in the same Speech resource will see each other's work in the Audio Content Creation tool. If you want each individual user to have a unique and private

workplace in Audio Content Creation, [create a new Speech resource](#) for each user and give each user the unique access to the Speech resource.

Remove users from a Speech resource

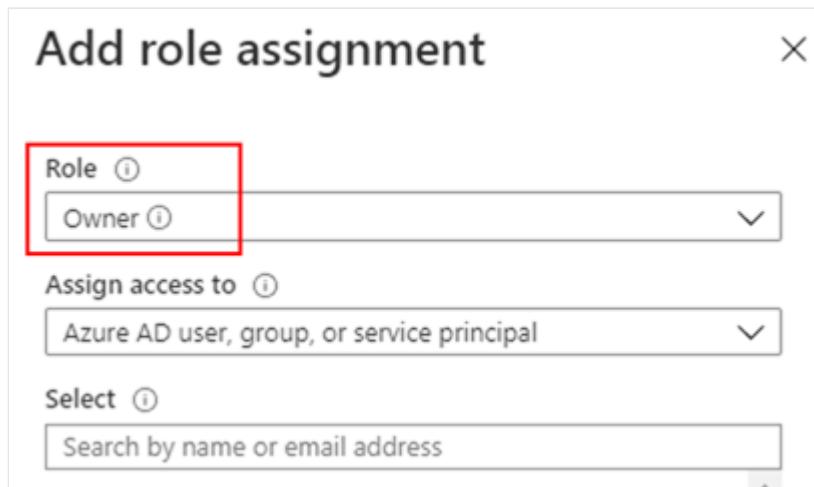
1. Search for **Cognitive services** in the Azure portal, select the Speech resource that you want to remove users from.
2. Select **Access control (IAM)**, and then select the **Role assignments** tab to view all the role assignments for this Speech resource.
3. Select the users you want to remove, select **Remove**, and then select **OK**.



Enable users to grant access to others

If you want to allow a user to grant access to other users, you need to assign them the owner role for the Speech resource and set the user as the Azure directory reader.

1. Add the user as the owner of the Speech resource. For more information, see [Add users to a Speech resource](#).



2. In the [Azure portal](#), select the collapsed menu at the upper left, select **Azure Active Directory**, and then select **Users**.
3. Search for the user's Microsoft account, go to their detail page, and then select **Assigned roles**.
4. Select **Add assignments** > **Directory Readers**. If the **Add assignments** button is unavailable, it means that you don't have access. Only the global administrator of this directory can add assignments to users.

Next steps

- [Speech Synthesis Markup Language \(SSML\)](#)
- [Batch synthesis](#)

What is speech translation?

Article • 09/20/2022 • 2 minutes to read

In this article, you learn about the benefits and capabilities of the speech translation service, which enables real-time, multi-language speech-to-speech and speech-to-text translation of audio streams.

By using the Speech SDK or Speech CLI, you can give your applications, tools, and devices access to source transcriptions and translation outputs for the provided audio. Interim transcription and translation results are returned as speech is detected, and the final results can be converted into synthesized speech.

For a list of languages supported for speech translation, see [Language and voice support](#).

Core features

- Speech-to-text translation with recognition results.
- Speech-to-speech translation.
- Support for translation to multiple target languages.
- Interim recognition and translation results.

Get started

As your first step, try the [Speech translation quickstart](#). The speech translation service is available via the [Speech SDK](#) and the [Speech CLI](#).

You'll find [Speech SDK speech-to-text and translation samples](#) ↗ on GitHub. These samples cover common scenarios, such as reading audio from a file or stream, continuous and single-shot recognition and translation, and working with custom models.

Next steps

- Try the [speech translation quickstart](#)
- Install the [Speech SDK](#)
- Install the [Speech CLI](#)

Quickstart: Recognize and translate speech to text

Article • 09/20/2022 • 27 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this quickstart, you run an application to translate speech from one language to text in another language.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

Set up the environment

The Speech SDK is available as a [NuGet package](#) and implements .NET Standard 2.0. You install the Speech SDK later in this guide, but first check the [SDK installation guide](#) for any more requirements.

Set environment variables

Your application must be authenticated to access Cognitive Services resources. For production, use a secure way of storing and accessing your credentials. For example, after you [get a key for your Speech resource](#), write it to a new environment variable on the local machine running the application.

Tip

Don't include the key directly in your code, and never post it publicly. See the Cognitive Services **security** article for more authentication options like **Azure Key Vault**.

To set the environment variable for your Speech resource key, open a console window, and follow the instructions for your operating system and development environment. To

set the `SPEECH_KEY` environment variable, replace `your-key` with one of the keys for your resource.

Windows

Console

```
setx SPEECH_KEY your-key
```

 **Note**

If you only need to access the environment variable in the current running console, you can set the environment variable with `set` instead of `setx`.

After you add the environment variable, you may need to restart any running programs that will need to read the environment variable, including the console window. For example, if you are using Visual Studio as your editor, restart Visual Studio before running the example.

To set the environment variable for your Speech resource region, follow the same steps. Set `SPEECH_REGION` to the region of your resource. For example, `westus`.

Translate speech from a microphone

Follow these steps to create a new console application and install the Speech SDK.

1. Open a command prompt where you want the new project, and create a console application with the .NET CLI. The `Program.cs` file should be created in the project directory.

.NET CLI

```
dotnet new console
```

2. Install the Speech SDK in your new project with the .NET CLI.

.NET CLI

```
dotnet add package Microsoft.CognitiveServices.Speech
```

3. Replace the contents of `Program.cs` with the following code.

C#

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Translation;

class Program
{
    // This example requires environment variables named "SPEECH_KEY"
    // and "SPEECH_REGION"
    static string speechKey =
        Environment.GetEnvironmentVariable("SPEECH_KEY");
    static string speechRegion =
        Environment.GetEnvironmentVariable("SPEECH_REGION");

    static void
    OutputSpeechRecognitionResult(TranslationRecognitionResult
        translationRecognitionResult)
    {
        switch (translationRecognitionResult.Reason)
        {
            case ResultReason.TranslatedSpeech:
                Console.WriteLine($"RECOGNIZED: Text={translationRecognitionResult.Text}");
                foreach (var element in
                    translationRecognitionResult.Translations)
                {
                    Console.WriteLine($"TRANSLATED into '{element.Key}': {element.Value}");
                }
                break;
            case ResultReason.NoMatch:
                Console.WriteLine($"NOMATCH: Speech could not be
                    recognized.");
                break;
            case ResultReason.Canceled:
                var cancellation =
                    CancellationDetails.FromResult(translationRecognitionResult);
                Console.WriteLine($"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 set the
                        speech resource key and region values?");
```

```

        }
        break;
    }

    async static Task Main(string[] args)
{
    var speechTranslationConfig =
SpeechTranslationConfig.FromSubscription(speechKey, speechRegion);
    speechTranslationConfig.SpeechRecognitionLanguage = "en-US";
    speechTranslationConfig.AddTargetLanguage("it");

    using var audioConfig =
AudioConfig.FromDefaultMicrophoneInput();
    using var translationRecognizer = new
TranslationRecognizer(speechTranslationConfig, audioConfig);

    Console.WriteLine("Speak into your microphone.");
    var translationRecognitionResult = await
translationRecognizer.RecognizeOnceAsync();
    OutputSpeechRecognitionResult(translationRecognitionResult);
}
}

```

4. To change the speech recognition language, replace `en-US` with another [supported language](#). Specify the full locale with a dash (-) separator. For example, `es-ES` for Spanish (Spain). The default language is `en-US` if you don't specify a language. For details about how to identify one of multiple languages that might be spoken, see [language identification](#).

5. To change the translation target language, replace `it` with another [supported language](#). With few exceptions you only specify the language code that precedes the locale dash (-) separator. For example, use `es` for Spanish (Spain) instead of `es-ES`. The default language is `en` if you don't specify a language.

Run your new console application to start speech recognition from a microphone:

Console

`dotnet run`

Speak into your microphone when prompted. What you speak should be output as translated text in the target language:

Console

Speak into your microphone.
RECOGNIZED: Text=I'm excited to try speech translation.

TRANSLATED into 'it': Sono entusiasta di provare la traduzione vocale.

Remarks

Now that you've completed the quickstart, here are some additional considerations:

- This example uses the `RecognizeOnceAsync` operation to transcribe utterances of up to 30 seconds, or until silence is detected. For information about continuous recognition for longer audio, including multi-lingual conversations, see [How to translate speech](#).
- To recognize speech from an audio file, use `FromWavFileInput` instead of `FromDefaultMicrophoneInput`:

C#

```
using var audioConfig =
    AudioConfig.FromWavFileInput("YourAudioFile.wav");
```

- For compressed audio files such as MP4, install GStreamer and use `PullAudioInputStream` or `PushAudioInputStream`. For more information, see [How to use compressed input audio](#).

Clean up resources

You can use the [Azure portal](#) or [Azure Command Line Interface \(CLI\)](#) to remove the Speech resource you created.

Next steps

[Learn more about speech translation](#)

How to recognize and translate speech

Article • 06/10/2022 • 38 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this how-to guide, you learn how to recognize human speech and translate it to another language.

See the speech translation [overview](#) for more information about:

- Translating speech to text
- Translating speech to multiple target languages
- Performing direct speech-to-speech translation

Sensitive data and environment variables

The example source code in this article depends on environment variables for storing sensitive data, such as the Speech resource's subscription key and region. The `Program` class contains two `static readonly string` values that are assigned from the host machine's environment variables: `SPEECH_SUBSCRIPTION_KEY` and `SPEECH_SERVICE_REGION`. Both of these fields are at the class scope, so they're accessible within method bodies of the class:

C#

```
public class Program
{
    static readonly string SPEECH_SUBSCRIPTION_KEY =
        Environment.GetEnvironmentVariable(nameof(SPEECH_SUBSCRIPTION_KEY));

    static readonly string SPEECH_SERVICE_REGION =
        Environment.GetEnvironmentVariable(nameof(SPEECH_SERVICE_REGION));

    static Task Main() => Task.CompletedTask;
}
```

For more information on environment variables, see [Environment variables and application configuration](#).

Create a speech translation configuration

To call the Speech service by using the Speech SDK, you need to create a `SpeechTranslationConfig` instance. This class includes information about your subscription, like your key and associated region, endpoint, host, or authorization token.

Tip

Regardless of whether you're performing speech recognition, speech synthesis, translation, or intent recognition, you'll always create a configuration.

You can initialize `SpeechTranslationConfig` in a few ways:

- With a subscription: pass in a key and the associated region.
- With an endpoint: pass in a Speech service endpoint. A key or authorization token is optional.
- With a host: pass in a host address. A key or authorization token is optional.
- With an authorization token: pass in an authorization token and the associated region.

Let's look at how you create a `SpeechTranslationConfig` instance by using a key and region. Get the Speech resource key and region in the [Azure portal](#).

C#

```
public class Program
{
    static readonly string SPEECH_SUBSCRIPTION_KEY =
        Environment.GetEnvironmentVariable(nameof(SPEECH_SUBSCRIPTION_KEY));

    static readonly string SPEECH_SERVICE_REGION =
        Environment.GetEnvironmentVariable(nameof(SPEECH_SERVICE_REGION));

    static Task Main() => TranslateSpeechAsync();

    static async Task TranslateSpeechAsync()
    {
        var translationConfig =
            SpeechTranslationConfig.FromSubscription(SPEECH_SUBSCRIPTION_KEY,
            SPEECH_SERVICE_REGION);
    }
}
```

Change the source language

One common task of speech translation is specifying the input (or source) language. The following example shows how you would change the input language to Italian. In your code, interact with the `SpeechTranslationConfig` instance by assigning it to the `SpeechRecognitionLanguage` property:

```
C#  
  
static async Task TranslateSpeechAsync()  
{  
    var translationConfig =  
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,  
SPEECH__SERVICE__REGION);  
  
    // Source (input) language  
    translationConfig.SpeechRecognitionLanguage = "it-IT";  
}
```

The `SpeechRecognitionLanguage` property expects a language-locale format string. Refer to the [list of supported speech-to-text locales](#).

Add a translation language

Another common task of speech translation is to specify target translation languages. At least one is required, but multiples are supported. The following code snippet sets both French and German as translation language targets:

```
C#  
  
static async Task TranslateSpeechAsync()  
{  
    var translationConfig =  
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,  
SPEECH__SERVICE__REGION);  
  
    translationConfig.SpeechRecognitionLanguage = "it-IT";  
  
    // Translate to languages. See https://aka.ms/speech/sttt-languages  
    translationConfig.AddTargetLanguage("fr");  
    translationConfig.AddTargetLanguage("de");  
}
```

With every call to `AddTargetLanguage`, a new target translation language is specified. In other words, when speech is recognized from the source language, each target translation is available as part of the resulting translation operation.

Initialize a translation recognizer

After you've created a [SpeechTranslationConfig](#) instance, the next step is to initialize [TranslationRecognizer](#). When you initialize `TranslationRecognizer`, you need to pass it your `translationConfig` instance. The configuration object provides the credentials that the Speech service requires to validate your request.

If you're recognizing speech by using your device's default microphone, here's what the `TranslationRecognizer` instance should look like:

C#

```
static async Task TranslateSpeechAsync()
{
    var translationConfig =
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,
SPEECH__SERVICE__REGION);

    var fromLanguage = "en-US";
    var toLanguages = new List<string> { "it", "fr", "de" };
    translationConfig.SpeechRecognitionLanguage = fromLanguage;
    toLanguages.ForEach(translationConfig.AddTargetLanguage);

    using var recognizer = new TranslationRecognizer(translationConfig);
}
```

If you want to specify the audio input device, then you need to create an [AudioConfig](#) class instance and provide the `audioConfig` parameter when initializing `TranslationRecognizer`.



Learn how to get the device ID for your audio input device.

First, reference the `AudioConfig` object as follows:

C#

```
static async Task TranslateSpeechAsync()
{
    var translationConfig =
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,
SPEECH__SERVICE__REGION);

    var fromLanguage = "en-US";
    var toLanguages = new List<string> { "it", "fr", "de" };
```

```

    translationConfig.SpeechRecognitionLanguage = fromLanguage;
    toLanguages.ForEach(translationConfig.AddTargetLanguage);

    using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();
    using var recognizer = new TranslationRecognizer(translationConfig,
audioConfig);
}

```

If you want to provide an audio file instead of using a microphone, you still need to provide an `audioConfig` parameter. However, when you create an `AudioConfig` class instance, instead of calling `FromDefaultMicrophoneInput`, you call `FromWavFileInput` and pass the `filename` parameter:

C#

```

static async Task TranslateSpeechAsync()
{
    var translationConfig =
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,
SPEECH__SERVICE__REGION);

    var fromLanguage = "en-US";
    var toLanguages = new List<string> { "it", "fr", "de" };
    translationConfig.SpeechRecognitionLanguage = fromLanguage;
    toLanguages.ForEach(translationConfig.AddTargetLanguage);

    using var audioConfig =
        AudioConfig.FromWavFileInput("YourAudioFile.wav");
    using var recognizer = new TranslationRecognizer(translationConfig,
audioConfig);
}

```

Translate speech

To translate speech, the Speech SDK relies on a microphone or an audio file input. Speech recognition occurs before speech translation. After all objects have been initialized, call the recognize-once function and get the result:

C#

```

static async Task TranslateSpeechAsync()
{
    var translationConfig =
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,
SPEECH__SERVICE__REGION);

    var fromLanguage = "en-US";
    var toLanguages = new List<string> { "it", "fr", "de" };

```

```
translationConfig.SpeechRecognitionLanguage = fromLanguage;
toLanguages.ForEach(translationConfig.AddTargetLanguage);

using var recognizer = new TranslationRecognizer(translationConfig);

Console.WriteLine($"Say something in '{fromLanguage}' and ");
Console.WriteLine($"we'll translate into '{string.Join("", "", toLanguages)}'.\n");

var result = await recognizer.RecognizeOnceAsync();
if (result.Reason == ResultReason.TranslatedSpeech)
{
    Console.WriteLine($"Recognized: \'{result.Text}\':");
    foreach (var element in result.Translations)
    {
        Console.WriteLine($"      TRANSLATED into '{element.Key}':
{element.Value}");
    }
}
```

For more information about speech-to-text, see [the basics of speech recognition](#).

Synthesize translations

After a successful speech recognition and translation, the result contains all the translations in a dictionary. The [Translations](#) dictionary key is the target translation language, and the value is the translated text. Recognized speech can be translated and then synthesized in a different language (speech-to-speech).

Event-based synthesis

The `TranslationRecognizer` object exposes a `Synthesizing` event. The event fires several times and provides a mechanism to retrieve the synthesized audio from the translation recognition result. If you're translating to multiple languages, see [Manual synthesis](#).

Specify the synthesis voice by assigning a `VoiceName` instance, and provide an event handler for the `Synthesizing` event to get the audio. The following example saves the translated audio as a .wav file.

Important

The event-based synthesis works only with a single translation. *Do not add multiple target translation languages.* Additionally, the `VoiceName` value should be the same

language as the target translation language. For example, "de" could map to "de-DE-Hedda".

C#

```
static async Task TranslateSpeechAsync()
{
    var translationConfig =
        SpeechTranslationConfig.FromSubscription(SPEECH__SUBSCRIPTION__KEY,
SPEECH__SERVICE__REGION);

    var fromLanguage = "en-US";
    var toLanguage = "de";
    translationConfig.SpeechRecognitionLanguage = fromLanguage;
    translationConfig.AddTargetLanguage(toLanguage);

    // See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
    translationConfig.VoiceName = "de-DE-Hedda";

    using var recognizer = new TranslationRecognizer(translationConfig);

    recognizer.Synthesizing += (_, e) =>
    {
        var audio = e.Result.GetAudio();
        Console.WriteLine($"Audio synthesized: {audio.Length:#,0} byte(s)
{((audio.Length == 0 ? "(Complete)" : "")})");

        if (audio.Length > 0)
        {
            File.WriteAllBytes("YourAudioFile.wav", audio);
        }
    };

    Console.Write($"Say something in '{fromLanguage}' and ");
    Console.WriteLine($"we'll translate into '{toLanguage}'.\n");

    var result = await recognizer.RecognizeOnceAsync();
    if (result.Reason == ResultReason.TranslatedSpeech)
    {
        Console.WriteLine($"Recognized: \\"{result.Text}\\\"");
        Console.WriteLine($"Translated into '{toLanguage}':
{result.Translations[toLanguage]}");
    }
}
```

Manual synthesis

You can use the [Translations](#) dictionary to synthesize audio from the translation text. Iterate through each translation and synthesize it. When you're creating a

`SpeechSynthesizer` instance, the `SpeechConfig` object needs to have its `SpeechSynthesisVoiceName` property set to the desired voice.

The following example translates to five languages. Each translation is then synthesized to an audio file in the corresponding neural language.

C#

```
static async Task TranslateSpeechAsync()
{
    var translationConfig =
        SpeechTranslationConfig.FromSubscription(SPEECH__SERVICE__KEY,
SPEECH__SERVICE__REGION);

    var fromLanguage = "en-US";
    var toLanguages = new List<string> { "de", "en", "it", "pt", "zh-Hans" };
    translationConfig.SpeechRecognitionLanguage = fromLanguage;
    toLanguages.ForEach(translationConfig.AddTargetLanguage);

    using var recognizer = new TranslationRecognizer(translationConfig);

    Console.WriteLine($"Say something in '{fromLanguage}' and ");
    Console.WriteLine($"we'll translate into '{string.Join(", ", toLanguages)}'.\n");

    var result = await recognizer.RecognizeOnceAsync();
    if (result.Reason == ResultReason.TranslatedSpeech)
    {
        // See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
        var languageToVoiceMap = new Dictionary<string, string>
        {
            ["de"] = "de-DE-KatjaNeural",
            ["en"] = "en-US-AriaNeural",
            ["it"] = "it-IT-ElsaNeural",
            ["pt"] = "pt-BR-FranciscaNeural",
            ["zh-Hans"] = "zh-CN-XiaoxiaoNeural"
        };

        Console.WriteLine($"Recognized: \"{result.Text}\")");

        foreach (var (language, translation) in result.Translations)
        {
            Console.WriteLine($"Translated into '{language}': {translation}");

            var speechConfig =
                SpeechConfig.FromSubscription(
                    SPEECH__SERVICE__KEY, SPEECH__SERVICE__REGION);
            speechConfig.SpeechSynthesisVoiceName =
                languageToVoiceMap[language];

            using var audioConfig = AudioConfig.FromWavFileOutput($"
```

```
{language}-translation.wav");
        using var synthesizer = new SpeechSynthesizer(speechConfig,
audioConfig);

        await synthesizer.SpeakTextAsync(translation);
    }
}
}
```

For more information about speech synthesis, see [the basics of speech synthesis](#).

Multi-lingual translation with language identification

In many scenarios, you might not know which input languages to specify. Using language identification allows you to specify up to 10 possible input languages and automatically translate to your target languages.

The following example uses continuous translation from an audio file. It automatically detects the input language, even if the language being spoken is changing. When you run the sample, `en-US` and `zh-CN` will be automatically detected because they're defined in `AutoDetectSourceLanguageConfig`. Then, the speech will be translated to `de` and `fr` as specified in the calls to `AddTargetLanguage()`.

ⓘ Important

This feature is currently in *preview*.

C#

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

public static async Task MultiLingualTranslation()
{
    var region = "<paste-your-region>";
    // Currently, the v2 endpoint is required for this design pattern
    var endpointString =
$"wss://{{region}}.stt.speech.microsoft.com/speech/universal/v2";
    var endpointUrl = new Uri(endpointString);

    var config = SpeechTranslationConfig.FromEndpoint(endpointUrl, "<paste-
your-subscription-key>");

    // Source language is required, but is currently NoOp
    string fromLanguage = "en-US";
```

```

config.SpeechRecognitionLanguage = fromLanguage;

config.AddTargetLanguage("de");
config.AddTargetLanguage("fr");

config SetProperty(PropertyId.SpeechServiceConnection_ContinuousLanguageIdPriority, "Latency");
var autoDetectSourceLanguageConfig =
AutoDetectSourceLanguageConfig.FromLanguages(new string[] { "en-US", "zh-CN" });

var stopTranslation = new TaskCompletionSource<int>();
using (var audioInput = AudioConfig.FromWavFileInput(@"path-to-your-
audio-file.wav"))
{
    using (var recognizer = new TranslationRecognizer(config,
autoDetectSourceLanguageConfig, audioInput))
    {
        recognizer.Recognizing += (s, e) =>
        {
            var lidResult =
e.Result.Properties.GetProperty(PropertyId.SpeechServiceConnection_AutoDetec-
tSourceLanguageResult);

            Console.WriteLine($"RECOGNIZING in '{lidResult}': Text=
{e.Result.Text}");
            foreach (var element in e.Result.Translations)
            {
                Console.WriteLine($"      TRANSLATING into
'{element.Key}': {element.Value}");
            }
        };

        recognizer.Recognized += (s, e) => {
            if (e.Result.Reason == ResultReason.TranslatedSpeech)
            {
                var lidResult =
e.Result.Properties.GetProperty(PropertyId.SpeechServiceConnection_AutoDetec-
tSourceLanguageResult);

                Console.WriteLine($"RECOGNIZED in '{lidResult}': Text=
{e.Result.Text}");
                foreach (var element in e.Result.Translations)
                {
                    Console.WriteLine($"      TRANSLATED into
'{element.Key}': {element.Value}");
                }
            }
            else if (e.Result.Reason == ResultReason.RecognizedSpeech)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"      Speech not translated.");
            }
            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 set the speech
resource key and region values?");
    }

    stopTranslation.TrySetResult(0);
};

recognizer.SpeechStartDetected += (s, e) => {
    Console.WriteLine("\nSpeech start detected event.");
};

recognizer.SpeechEndDetected += (s, e) => {
    Console.WriteLine("\nSpeech end detected event.");
};

recognizer.SessionStarted += (s, e) => {
    Console.WriteLine("\nSession started event.");
};

recognizer.SessionStopped += (s, e) => {
    Console.WriteLine("\nSession stopped event.");
    Console.WriteLine($"\\nStop translation.");
    stopTranslation.TrySetResult(0);
};

// Start continuous recognition. Use
StopContinuousRecognitionAsync() to stop recognition.
Console.WriteLine("Start translation...");
await
recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

Task.WaitAny(new[] { stopTranslation.Task });
await
recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
}
}
}

```

Next steps

- Try the speech to text quickstart
- Try the speech translation quickstart
- Improve recognition accuracy with custom speech

Intent recognition documentation

Intent recognition with the Speech and Language Understanding (LUIS) services, enables real-time transcription of audio streams into text, while identifying intent and entities.

About intent recognition

QUICKSTART

[Recognize speech, intents, and entities](#)

Develop with intent recognition

HOW-TO GUIDE

[Overview](#)

[Recognize speech intents with LUIS](#)

[Recognize speech intents with simple pattern matching](#)

[Recognize speech intents with custom entity matching](#)

Reference

REFERENCE

[Language Understanding \(LUIS\) portal ↗](#)

[LUIS developer resources](#)

[Language support](#)

[Intent recognition pricing ↗](#)

Help and feedback

REFERENCE

[Support and help options](#)

What is intent recognition?

Article • 06/19/2022 • 2 minutes to read

In this overview, you will learn about the benefits and capabilities of intent recognition. The Cognitive Services Speech SDK provides two ways to recognize intents, both described below. An intent is something the user wants to do: book a flight, check the weather, or make a call. Using intent recognition, your applications, tools, and devices can determine what the user wants to initiate or do based on options you define in the Intent Recognizer or LUIS.

Pattern matching

The SDK provides an embedded pattern matcher that you can use to recognize intents in a very strict way. This is useful for when you need a quick offline solution. This works especially well when the user is going to be trained in some way or can be expected to use specific phrases to trigger intents. For example: "Go to floor seven", or "Turn on the lamp" etc. It is recommended to start here and if it no longer meets your needs, switch to using LUIS or a combination of the two.

Luis (Language Understanding Intent Service)

The Microsoft LUIS service is available as a complete AI intent service that works well when your domain of possible intents is large and you are not really sure what the user will say. It supports many complex scenarios, intents, and entities.

Luis key required

- LUIS integrates with the Speech service to recognize intents from speech. You don't need a Speech service subscription, just LUIS.
- Speech intent recognition is integrated with the Speech SDK. You can use a LUIS key with the Speech service.
- Intent recognition through the Speech SDK is [offered in a subset of regions supported by LUIS](#).

Get started

See this [how-to](#) to get started with pattern matching.

See this [quickstart](#) to get started with LUIS intent recognition.

Sample code

Sample code for intent recognition:

- Quickstart: Use prebuilt Home automation app
- Recognize intents from speech using the Speech SDK for C#
- Intent recognition and other Speech services using Unity in C# ↗
- Recognize intents using Speech SDK for Python ↗
- Intent recognition and other Speech services using the Speech SDK for C++ on Windows ↗
- Intent recognition and other Speech services using the Speech SDK for Java on Windows or Linux ↗
- Intent recognition and other Speech services using the Speech SDK for JavaScript on a web browser ↗

Reference docs

- [Speech SDK](#)

Next steps

- [Intent recognition quickstart](#)
- [Get the Speech SDK](#)

What is pattern matching?

Article • 04/20/2022 • 6 minutes to read

Pattern matching can be customized to group together pattern intents and entities inside a `PatternMatchingModel`. Using this grouping, it's possible to access more advanced entity types that will help make your intent recognition more precise.

For supported locales see [here](#).

Patterns vs. Exact Phrases

There are two types of strings used in the pattern matcher: "exact phrases" and "patterns". It's important to understand the differences.

Exact phrases are a string of the exact words that you'll want to match. For example:

"Take me to floor seven".

A pattern is a phrase that contains a marked entity. Entities are marked with "{}" to define the place inside the pattern and the text inside the "{}" references the entity ID. Given the previous example perhaps you would want to extract the floor name in an entity named "floorName". You would do so with a pattern like this:

"Take me to floor {floorName}"

Outline of a PatternMatchingModel

The `PatternMatchingModel` contains an ID to reference that model by, a list of `PatternMatchingIntent` objects, and a list of `PatternMatchingEntity` objects.

Pattern Matching Intents

`PatternMatchingIntent` objects represent a collection of phrases that will be used to evaluate speech or text in the `IntentRecognizer`. If the phrases are matched, the `IntentRecognitionResult` returned will have the ID of the `PatternMatchingIntent` that was matched.

Pattern Matching Entities

`PatternMatchingEntity` objects represent an individual entity reference and its corresponding properties that tell the `IntentRecognizer` how to treat it. All `PatternMatchingEntity` objects must have an ID that is present in a phrase or else it will never be matched.

Entity Naming restrictions

Entity names containing ':' characters will assign a role to an entity. (See below)

Types of Entities

Any Entity

The "Any" entity will match any text that appears in that slot regardless of the text it contains. If we consider our previous example using the pattern "Take me to floor {floorName}", the user might say something like:

"Take me to the floor parking 2"

In this case, the "floorName" entity would match "parking 2".

These are lazy matches that will attempt to match as few words as possible unless it appears at the beginning or end of an utterance. Consider the pattern:

"Take me to the floor {floorName1} {floorName2}"

In this case, the utterance "Take me to the floor parking 2" would match and return floorName1 = "parking" and floorName2 = "2".

It may be tricky to handle extra text if it's captured. Perhaps the user kept talking and the utterance captured more than their command. "Take me to floor parking 2 yes Janice I heard about that let's". In this case the floorName1 would be correct, but floorName2 would = "2 yes Janice I heard about that let's". It's important to be aware of the way the Entities will match and adjust to your scenario appropriately. The Any entity type is the most basic and least precise.

List Entity

The "List" entity is made up of a list of phrases that will guide the engine on how to match it. The "List" entity has two modes. "Strict" and "Fuzzy".

Let's assume we have a list of floors for our elevator. Since we're dealing with speech, we will add entries for the lexical format as well.

"1", "2", "3", "lobby", "ground floor", "one", "two", "three"

When an entity with an ID is of type "List" and is in "Strict" mode, the engine will only match if the text in the slot appears in the list.

"take me to floor one" will match.

"take me to floor 5" will not.

It's important to note that the Intent will not match, not just the entity.

When an entity is of type "List" and is in "Fuzzy" mode, the engine will still match the Intent, and will return the text that appeared in the slot in the utterance even if it's not in the list. This is useful behind the scenes to help make the speech recognition better.

Warning

Fuzzy list entities are implemented, but not integrated into the speech recognition part. Therefore, they will match entities, but not improve speech recognition.

Prebuilt Integer Entity

The "PrebuiltInteger" entity is used when you expect to get an integer in that slot. It won't match the intent if an integer cannot be found. The return value is a string representation of the number.

Examples of a valid match and return values

"Two thousand one hundred and fifty-five" -> "2155"

"first" -> "1"

"a" -> "1"

"four oh seven one" -> "4071"

If there's text that is not recognizable as a number, the entity and intent will not match.

Examples of an invalid match

"the third"

"first floor I think"

"second plus three"

"thirty-three and anyways"

Consider our elevator example.

"Take me to floor {floorName}"

If "floorName" is a prebuilt integer entity, the expectation is that whatever text is inside the slot will represent an integer. Here a floor number would match well, but a floor with a name such as "lobby" would not.

Grouping required and optional items

In the pattern it's allowed to include words or entities that may be present in the utterance or not. This is especially useful for determiners like "the", "a", or "an". This doesn't have any functional difference from hard coding out the many combinations, but can help reduce the number of patterns needed. Indicate optional items with "[" and "]". Indicate required items with "(" and ")". You may include multiple items in the same group by separating them with a '|' character.

To see how this would reduce the number of patterns needed consider the following set.

"Take me to {floorName}"

"Take me the {floorName}"

"Take me {floorName}"

"Take me to {floorName} please"

"Take me the {floorName} please"

"Take me {floorName} please"

"Bring me {floorName} please"

"Bring me to {floorName} please"

These can all be reduced to a single pattern with grouping and optional items. First, it is possible to group "to" and "the" together as optional words like so: "[to | the]", and second we can make the "please" optional as well. Last, we can group the "bring" and "take" as required.

"(Bring | Take) me [to | the] {floorName} [please]"

It's also possible to include optional entities. Imagine there are multiple parking levels and you want to match the word before the {floorName}. You could do so with a pattern like this:

"Take me to [{floorType}] {floorName}"

Optionals are also very useful if you might be using keyword recognition and a push-to-talk function. This means sometimes the keyword will be present, and sometimes it won't. Assuming your keyword was "computer" your pattern would look something like this.

"[Computer] Take me to {floorName}"

ⓘ Note

While it's helpful to use optional items, it increases the chances of pattern collisions. This is where two patterns can match the same-spoken phrase. If this occurs, it can sometimes be solved by separating out the optional items into separate patterns.

Entity roles

Inside the pattern, there may be a scenario where you want to use the same entity multiple times. Consider the scenario of booking a flight from one city to another. In this case the list of cities is the same, but it's necessary to know which city is the user coming from and which city is the destination. To accomplish this, you can use a role assigned to an entity using a ':'.

"Book a flight from {city:from} to {city:destination}"

Given a pattern like this, there will be two entities in the result labeled "city:from" and "city:destination" but they'll both be referencing the "city" entity for matching purposes.

Intent Matching Priority

Sometimes multiple patterns may match the same utterance. In this case, the engine will give priority to patterns as follows.

1. Exact Phrases.
2. Patterns with more Entities.
3. Patterns with Integer Entities.
4. Patterns with List Entities.
5. Patterns with Any Entities.

Next steps

- Start with [simple pattern matching](#).
- Improve your pattern matching by using [custom entities](#).
- Look through our [GitHub samples ↗](#).

Quickstart: Recognize intents with the Speech service and LUIS

Article • 02/20/2022 • 42 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional 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.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Language resource](#) in the Azure portal. You can use the free pricing tier (`F0`) to try the service, and upgrade later to a paid tier for production. You won't need a Speech resource this time.
- ✓ Get the resource key and region. After your Language resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

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 requires a LUIS subscription [in a region where intent recognition is available](#). 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 four things:

- Re-publish with **Speech priming** toggled on
- Your **Luis Primary key**

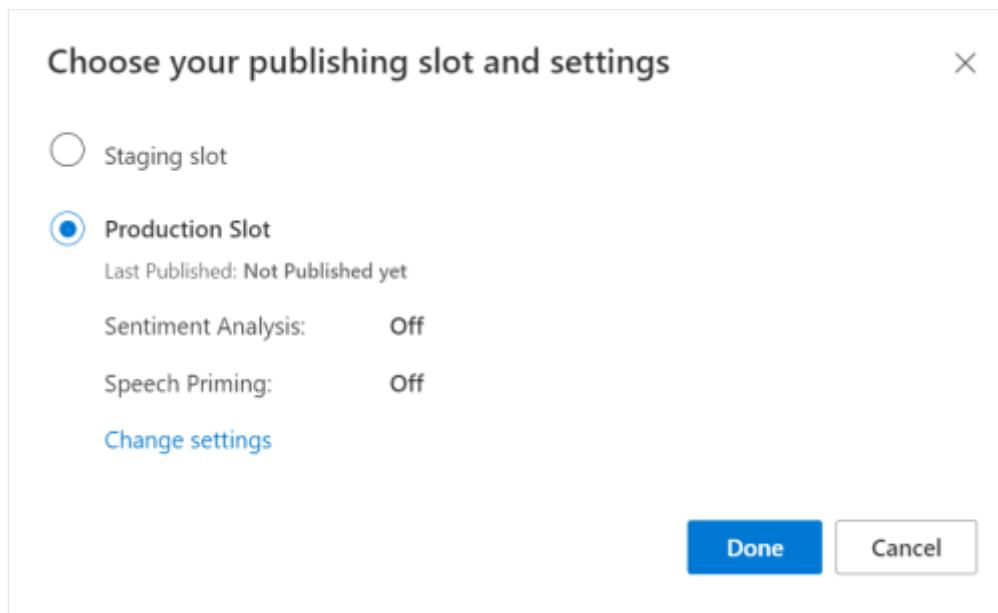
- Your LUIS Location
- Your LUIS App ID

Here's where you can find this information in the [LUIS preview portal](#):

1. From the LUIS preview portal, select your app then select the **Publish** button.
2. Select the **Production** slot, if you're using `en-US` select **change settings**, and toggle the **Speech priming** option to the **On** position. Then select the **Publish** button.

ⓘ Important

Speech priming is highly recommended as it will improve speech recognition accuracy.



3. 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*) for your LUIS prediction resource.

The screenshot shows the Microsoft LUIS Azure Resources page. The left sidebar has options: Settings, Publish Settings, Azure Resources (which is selected and highlighted in blue), and Versions. The main content area is titled "Azure Resources" and describes two types of resources: authoring and prediction. It says an authoring resource is created when you create your Azure cognitive service account. For publishing, it's necessary to create a prediction resource and use its key with endpoint queries. A link to "Learn more" is provided. Below this, there are tabs for "Prediction Resources" (selected) and "Authoring Resource". A large blue button labeled "Add prediction resource" is visible. Under "example-resource", there is a "Un-assign key" link. The "Resource group" is set to "my-resource-group". The "Location" is "eastus", and the "Primary Key" is a long string of characters starting with "xxxxxxxxxxxxxxxxxxxxxx" (partially redacted). The "Secondary Key" and "Endpoint URL" are also listed below.

4. After you've got your key and location, you'll need the app ID. Select **Settings**. Your app ID is available on this page.

The screenshot shows the Microsoft LUIS Application Settings page. The left sidebar has options: Settings (selected and highlighted in blue), Publish Settings, Azure Resources, and Versions. The main content area is titled "Application Settings" and shows the "App name" as "intent-recognition". There is a field for "App description (optional)" with a placeholder "Type text here ...". The "App ID" field contains the value "a39d3646-adf9-4264-94f4-de21bbcfab44", which is highlighted with a red rectangle.

Open your project in Visual Studio

Next, open your project 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 `RecognizeIntentAsync()`.

C#

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

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeIntentAsync()
        {
```

C#

```
    }

    static async Task Main()
    {
        await RecognizeIntentAsync();
        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. Use [Region identifier from region](#).

💡 Tip

If you need help finding these values, see [Create a LUIS app for intent recognition](#).

ⓘ Important

Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like [Azure Key Vault](#). See the Cognitive Services [security](#) article for more information.

C#

```
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 [How to recognize speech](#) 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.

C#

```
// Creates an intent recognizer using microphone as audio input.  
using (var recognizer = new IntentRecognizer(config))  
{
```

C#

```
}
```

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](#).

C#

```
// 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");
```

This example uses the `AddIntent()` function to individually add intents. If you want to add all intents from a model, use `AddAllIntents(model)` and pass the model.

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.

C#

```
// 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();
```

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:

C#

```
// Checks result.
switch (result.Reason)
{
    case ResultReason.RecognizedIntent:
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"      Intent Id: {result.IntentId}.");
        var json =
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceRespons
e_JsonResult);
        Console.WriteLine($"      Language Understanding JSON: {json}.");
        break;
    case ResultReason.RecognizedSpeech:
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"      Intent not recognized.");
        break;
    case ResultReason.NoMatch:
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
        break;
    case 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?");
        }
        break;
}
```

Check your code

At this point, your code should look like this:

ⓘ Note

We've added some comments to this version.

C#

```
//  
// Copyright (c) Microsoft. All rights reserved.  
// Licensed under the MIT license. See LICENSE.md file in the project root  
for full license information.  
  
// <skeleton_1>  
using System;  
using System.Threading.Tasks;  
using Microsoft.CognitiveServices.Speech;  
using Microsoft.CognitiveServices.Speech.Intent;  
  
namespace helloworld  
{  
    class Program  
    {  
        public static async Task RecognizeIntentAsync()  
        {  
            // </skeleton_1>  
            // 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".  
            // <create_speech_configuration>  
            var config = SpeechConfig.FromSubscription(  
                "YourLanguageUnderstandingSubscriptionKey",  
                "YourLanguageUnderstandingServiceRegion");  
            // </create_speech_configuration>
```

```

// <create_intent_recognizer_1>
// Creates an intent recognizer using microphone as audio input.
using (var recognizer = new IntentRecognizer(config))
{
    // </create_intent_recognizer_1>

    // <add_intents>
    // 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");
    // </add_intents>

    // To add all of the possible intents from a LUIS model to
    the recognizer, uncomment the line below:
    // recognizer.AddAllIntents(model);

    // <recognize_intent>
    // 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();
    // </recognize_intent>

    // <print_results>
    // Checks result.
    switch (result.Reason)
    {
        case ResultReason.RecognizedIntent:
            Console.WriteLine($"RECOGNIZED: Text={result.Text}");
            Console.WriteLine($"      Intent Id:{result.IntentId}.");
            var json =
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceRespons
e_JsonResult);
            Console.WriteLine($"      Language Understanding JSON:
{json}.");
    }
}

```

```

        break;
    case ResultReason.RecognizedSpeech:
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        break;
    case ResultReason.NoMatch:
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
        break;
    case 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?");
        }
        break;
    }
    // </print_results>
    // <create_intent_recognizer_2>
}
// </create_intent_recognizer_2>
// <skeleton_2>
}

static async Task Main()
{
    await RecognizeIntentAsync();
    Console.WriteLine("Please press <Return> to continue.");
    Console.ReadLine();
}
}
// </skeleton_2>

```

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 .
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

[See more LUIS samples on GitHub](#)

How to recognize intents with simple language pattern matching

Article • 07/29/2022 • 20 minutes to read

The Cognitive Services [Speech SDK](#) has a built-in feature to provide **intent recognition with simple language pattern matching**. An intent is something the user wants to do: close a window, mark a checkbox, insert some text, etc.

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
- ✓ Add intents and patterns via the Speech SDK API
- ✓ Recognize speech from a microphone
- ✓ Use asynchronous, event-driven continuous recognition

When to use pattern matching

Use this sample code if:

- You're only interested in matching strictly what the user said. These patterns match more aggressively than LUIS.
- You don't have access to a [LUIS](#) app, but still want intents.
- You can't or don't want to create a [LUIS](#) app but you still want some voice-commanding capability.

For more information, see the [pattern matching overview](#).

Prerequisites

Be sure you have the following items before you begin this guide:

- A [Cognitive Services Azure resource](#) or a [Unified Speech resource](#)
- [Visual Studio 2019](#) (any edition).

Speech and simple patterns

The simple patterns are a feature of the Speech SDK and need a Cognitive Services resource or a Unified Speech resource.

A pattern is a phrase that includes an Entity somewhere within it. An Entity is defined by wrapping a word in curly brackets. This example defines an Entity with the ID "floorName", which is case-sensitive:

```
Take me to the {floorName}
```

All other special characters and punctuation will be ignored.

Intents will be added using calls to the IntentRecognizer->AddIntent() API.

Create a project

Create a new C# console application project in Visual Studio 2019 and [install the Speech SDK](#).

Start with some boilerplate code

Let's open `Program.cs` and add some code that works as a skeleton for our project.

C#

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

namespace helloworld
{
    class Program
    {
        static void Main(string[] args)
        {
            IntentPatternMatchingWithMicrophoneAsync().Wait();
        }

        private static async Task IntentPatternMatchingWithMicrophoneAsync()
        {
            var config =
SpeechConfig.FromSubscription("YOUR_SUBSCRIPTION_KEY",
"YOUR_SUBSCRIPTION_REGION");
        }
    }
}
```

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 Cognitive Services prediction resource.

- Replace `"YOUR_SUBSCRIPTION_KEY"` with your Cognitive Services prediction key.
- Replace `"YOUR_SUBSCRIPTION_REGION"` with your Cognitive Services resource region.

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

Initialize an IntentRecognizer

Now create an `IntentRecognizer`. Insert this code right below your Speech configuration.

C#

```
using (var intentRecognizer = new IntentRecognizer(config))  
{  
}
```

Add some intents

You need to associate some patterns with the `IntentRecognizer` by calling `AddIntent()`.

We will add 2 intents with the same ID for changing floors, and another intent with a separate ID for opening and closing doors. Insert this code inside the `using` block:

C#

```
intentRecognizer.AddIntent("Take me to floor {floorName}.", "ChangeFloors");  
intentRecognizer.AddIntent("Go to floor {floorName}.", "ChangeFloors");  
intentRecognizer.AddIntent("{action} the door.", "OpenCloseDoor");
```

Note

There is no limit to the number of entities you can declare, but they will be loosely matched. If you add a phrase like "{action} door" it will match any time there is text before the word "door". Intents are evaluated based on their number of entities. If two patterns would match, the one with more defined entities is returned.

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method asks the Speech service to recognize speech in a single phrase, and stop recognizing speech once the phrase is identified. For simplicity we'll wait on the future returned to complete.

Insert this code below your intents:

```
C#  
  
Console.WriteLine("Say something...");  
  
var result = await intentRecognizer.RecognizeOnceAsync();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, we will print the result.

Insert this code below `var result = await recognizer.RecognizeOnceAsync();`:

```
C#  
  
string floorName;  
switch (result.Reason)  
{  
    case ResultReason.RecognizedSpeech:  
        Console.WriteLine($"RECOGNIZED: Text= {result.Text}");  
        Console.WriteLine($"      Intent not recognized.");  
        break;  
    case ResultReason.RecognizedIntent:  
        Console.WriteLine($"RECOGNIZED: Text= {result.Text}");  
        Console.WriteLine($"      Intent Id= {result.IntentId}.");  
        var entities = result.Entities;  
        if (entities.TryGetValue("floorName", out floorName))  
        {  
            Console.WriteLine($"      FloorName= {floorName}");  
        }  
  
        if (entities.TryGetValue("action", out floorName))  
        {  
            Console.WriteLine($"      Action= {floorName}");  
        }  
  
        break;  
    case ResultReason.NoMatch:  
    {  
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");  
    }
```

```

        var noMatch = NoMatchDetails.FromResult(result);
        switch (noMatch.Reason)
        {
            case NoMatchReason.NotRecognized:
                Console.WriteLine($"NOMATCH: Speech was detected, but not
recognized.");
                break;
            case NoMatchReason.InitialSilenceTimeout:
                Console.WriteLine($"NOMATCH: The start of the audio stream
contains only silence, and the service timed out waiting for speech.");
                break;
            case NoMatchReason.InitialBabbleTimeout:
                Console.WriteLine($"NOMATCH: The start of the audio stream
contains only noise, and the service timed out waiting for speech.");
                break;
            case NoMatchReason.KeywordNotRecognized:
                Console.WriteLine($"NOMATCH: Keyword not recognized");
                break;
        }
        break;
    }
    case 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 set the speech resource
key and region values?");
        }
        break;
    }
    default:
        break;
}

```

Check your code

At this point, your code should look like this:

C#

```

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

```

```
namespace helloworld
{
    class Program
    {
        static void Main(string[] args)
        {
            IntentPatternMatchingWithMicrophoneAsync().Wait();
        }

        private static async Task IntentPatternMatchingWithMicrophoneAsync()
        {
            var config =
SpeechConfig.FromSubscription("YOUR_SUBSCRIPTION_KEY",
"YOUR_SUBSCRIPTION_REGION");
            using (var intentRecognizer = new IntentRecognizer(config))
            {
                intentRecognizer.AddIntent("Take me to floor {floorName}.",
"ChangeFloors");
                intentRecognizer.AddIntent("Go to floor {floorName}.",
"ChangeFloors");
                intentRecognizer.AddIntent("{action} the door.",
"OpenCloseDoor");

                Console.WriteLine("Say something...");

                var result = await intentRecognizer.RecognizeOnceAsync();

                string floorName;
                switch (result.Reason)
                {
                    case ResultReason.RecognizedSpeech:
                        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                        Console.WriteLine($"      Intent not recognized.");
                        break;
                    case ResultReason.RecognizedIntent:
                        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                        Console.WriteLine($"      Intent Id={result.IntentId}");
                        var entities = result.Entities;
                        if (entities.TryGetValue("floorName", out
floorName))
                        {
                            Console.WriteLine($"      FloorName={floorName}");
                        }

                        if (entities.TryGetValue("action", out floorName))
                        {
                            Console.WriteLine($"      Action={floorName}");
                        }
                }
            }
        }
    }
}
```

```
        case ResultReason.NoMatch:
    {
        Console.WriteLine($"NOMATCH: Speech could not be
recognized.");
        var noMatch = NoMatchDetails.FromResult(result);
        switch (noMatch.Reason)
        {
            case NoMatchReason.NotRecognized:
                Console.WriteLine($"NOMATCH: Speech was
detected, but not recognized.");
                break;
            case NoMatchReason.InitialSilenceTimeout:
                Console.WriteLine($"NOMATCH: The start of
the audio stream contains only silence, and the service timed out waiting
for speech.");
                break;
            case NoMatchReason.InitialBabbleTimeout:
                Console.WriteLine($"NOMATCH: The start of
the audio stream contains only noise, and the service timed out waiting for
speech.");
                break;
            case NoMatchReason.KeywordNotRecognized:
                Console.WriteLine($"NOMATCH: Keyword not
recognized");
                break;
        }
        break;
    }
    case ResultReason.Canceled:
    {
        var cancellation =
CancellationDetails.FromResult(result);
        Console.WriteLine($"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 set the
speech resource key and region values?");
        }
        break;
    }
    default:
        break;
}
}
}
}
```

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 will prompt you to say something. The default language is English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

For example if you say "Take me to floor 7", this should be the output:

```
Say something ...
RECOGNIZED: Text= Take me to floor 7.
Intent Id= ChangeFloors
FloorName= 7
```

How to recognize intents with custom entity pattern matching

Article • 07/29/2022 • 25 minutes to read

The Cognitive Services [Speech SDK](#) has a built-in feature to provide **intent recognition with simple language pattern matching**. An intent is something the user wants to do: close a window, mark a checkbox, insert some text, etc.

In this guide, you use the Speech SDK to develop a console application that derives intents from speech utterances spoken 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
- ✓ Add intents and patterns via the Speech SDK API
- ✓ Add custom entities via the Speech SDK API
- ✓ Use asynchronous, event-driven continuous recognition

When to use pattern matching

Use this sample code if:

- You're only interested in matching strictly what the user said. These patterns match more aggressively than LUIS.
- You don't have access to a [LUIS](#) app, but still want intents.
- You can't or don't want to create a [LUIS](#) app but you still want some voice-commanding capability.

For more information, see the [pattern matching overview](#).

Prerequisites

Be sure you have the following items before you begin this guide:

- A [Cognitive Services Azure resource](#) or a [Unified Speech resource](#)
- [Visual Studio 2019](#) (any edition).

Create a project

Create a new C# console application project in Visual Studio 2019 and [install the Speech SDK](#).

Start with some boilerplate code

Let's open `Program.cs` and add some code that works as a skeleton for our project.

```
C#  
  
using System;  
using System.Threading.Tasks;  
using Microsoft.CognitiveServices.Speech;  
using Microsoft.CognitiveServices.Speech.Intent;  
  
namespace helloworld  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            IntentPatternMatchingWithMicrophoneAsync().Wait();  
        }  
  
        private static async Task IntentPatternMatchingWithMicrophoneAsync()  
        {  
            var config =  
SpeechConfig.FromSubscription("YOUR_SUBSCRIPTION_KEY",  
"YOUR_SUBSCRIPTION_REGION");  
        }  
    }  
}
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and Azure region for your Cognitive Services prediction resource.

- Replace `"YOUR_SUBSCRIPTION_KEY"` with your Cognitive Services prediction key.
- Replace `"YOUR_SUBSCRIPTION_REGION"` with your Cognitive Services resource region.

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

Initialize an IntentRecognizer

Now create an `IntentRecognizer`. Insert this code right below your Speech configuration.

```
C#  
  
using (var recognizer = new IntentRecognizer(config))  
{  
  
}
```

Add some intents

You need to associate some patterns with a `PatternMatchingModel` and apply it to the `IntentRecognizer`. We will start by creating a `PatternMatchingModel` and adding a few intents to it.

ⓘ Note

We can add multiple patterns to a `PatternMatchingIntent`.

Insert this code inside the `using` block:

```
C#  
  
// Creates a Pattern Matching model and adds specific intents from your  
model. The  
// Id is used to identify this model from others in the collection.  
var model = new PatternMatchingModel("YourPatternMatchingModelId");  
  
// Creates a pattern that uses groups of optional words. "[Go | Take me]"  
will match either "Go", "Take me", or "".  
var patternWithOptionalWords = "[Go | Take me] to [floor|level]  
{floorName}";  
  
// Creates a pattern that uses an optional entity and group that could be  
used to tie commands together.  
var patternWithOptionalEntity = "Go to parking [{parkingLevel}]";  
  
// You can also have multiple entities of the same name in a single pattern  
by adding appending a unique identifier  
// to distinguish between the instances. For example:  
var patternWithTwoOfTheSameEntity = "Go to floor {floorName:1} [and then go  
to floor {floorName:2}]";  
// NOTE: Both floorName:1 and floorName:2 are tied to the same list of  
entries. The identifier can be a string  
// and is separated from the entity name by a ':'
```

```
// Creates the pattern matching intents and adds them to the model
model.Intents.Add(new PatternMatchingIntent("ChangeFloors",
patternWithOptionalWords, patternWithOptionalEntity,
patternWithTwoOfTheSameEntity));
model.Intents.Add(new PatternMatchingIntent("DoorControl", "{action} the
doors", "{action} doors", "{action} the door", "{action} door"));
```

Add some custom entities

To take full advantage of the pattern matcher you can customize your entities. We will make "floorName" a list of the available floors. We will also make "parkingLevel" an integer entity.

Insert this code below your intents:

C#

```
// Creates the "floorName" entity and set it to type list.
// Adds acceptable values. NOTE the default entity type is Any and so we do
not need
// to declare the "action" entity.
model.Entities.Add(PatternMatchingEntity.CreateListEntity("floorName",
EntityMatchMode.Strict, "ground floor", "lobby", "1st", "first", "one", "1",
"2nd", "second", "two", "2"));

// Creates the "parkingLevel" entity as a pre-built integer
model.Entities.Add(PatternMatchingEntity.CreateIntegerEntity("parkingLevel")
);
```

Apply our model to the Recognizer

Now it is necessary to apply the model to the `IntentRecognizer`. It is possible to use multiple models at once so the API takes a collection of models.

Insert this code below your entities:

C#

```
var modelCollection = new LanguageUnderstandingModelCollection();
modelCollection.Add(model);

recognizer.ApplyLanguageModels(modelCollection);
```

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method asks the Speech service to recognize speech in a single phrase, and stop recognizing speech once the phrase is identified.

Insert this code after applying the language models:

```
C#
```

```
Console.WriteLine("Say something...");  
  
var result = await recognizer.RecognizeOnceAsync();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, we will print the result.

Insert this code below `var result = await recognizer.RecognizeOnceAsync();`:

```
C#
```

```
if (result.Reason == ResultReason.RecognizedIntent)  
{  
    Console.WriteLine($"RECOGNIZED: Text={result.Text}");  
    Console.WriteLine($"          Intent Id={result.IntentId}.");  
  
    var entities = result.Entities;  
    switch (result.IntentId)  
    {  
        case "ChangeFloors":  
            if (entities.TryGetValue("floorName", out string floorName))  
            {  
                Console.WriteLine($"          FloorName={floorName}");  
            }  
  
            if (entities.TryGetValue("floorName:1", out floorName))  
            {  
                Console.WriteLine($"          FloorName:1={floorName}");  
            }  
  
            if (entities.TryGetValue("floorName:2", out floorName))  
            {  
                Console.WriteLine($"          FloorName:2={floorName}");  
            }  
  
            if (entities.TryGetValue("parkingLevel", out string parkingLevel))  
            {  
                Console.WriteLine($"          ParkingLevel={parkingLevel}");  
            }  
    }  
}
```

```

        break;

    case "DoorControl":
        if (entities.TryGetValue("action", out string action))
        {
            Console.WriteLine($"          Action={action}");
        }
        break;
    }

} 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 set the speech resource key
and region values?");
    }
}
}

```

Check your code

At this point, your code should look like this:

C#

```

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

namespace helloworld
{
    class Program
    {
        static void Main(string[] args)
        {
            IntentPatternMatchingWithMicrophoneAsync().Wait();
        }
    }
}

```

```

    }

    private static async Task IntentPatternMatchingWithMicrophoneAsync()
    {
        var config =
SpeechConfig.FromSubscription("YOUR_SUBSCRIPTION_KEY",
"YOUR_SUBSCRIPTION_REGION");

        using (var recognizer = new IntentRecognizer(config))
        {
            // Creates a Pattern Matching model and adds specific
intents from your model. The
            // Id is used to identify this model from others in the
collection.
            var model = new
PatternMatchingModel("YourPatternMatchingModelId");

            // Creates a pattern that uses groups of optional words. "[Go | Take me]" will match either "Go", "Take me", or "".
            var patternWithOptionalWords = "[Go | Take me] to
[floor|level] {floorName}";

            // Creates a pattern that uses an optional entity and group
that could be used to tie commands together.
            var patternWithOptionalEntity = "Go to parking
[{parkingLevel}]";

            // You can also have multiple entities of the same name in a
single pattern by adding appending a unique identifier
            // to distinguish between the instances. For example:
            var patternWithTwoOfTheSameEntity = "Go to floor
{floorName:1} [and then go to floor {floorName:2}]";
            // NOTE: Both floorName:1 and floorName:2 are tied to the
same list of entries. The identifier can be a string
            //       and is separated from the entity name by a ':'

            // Adds some intents to look for specific patterns.
            model.Intents.Add(new PatternMatchingIntent("ChangeFloors",
patternWithOptionalWords, patternWithOptionalEntity,
patternWithTwoOfTheSameEntity));
            model.Intents.Add(new PatternMatchingIntent("DoorControl", "
{action} the doors", "{action} doors", "{action} the door", "{action}
door"));

            // Creates the "floorName" entity and set it to type list.
            // Adds acceptable values. NOTE the default entity type is
Any and so we do not need
            // to declare the "action" entity.

model.Entities.Add(PatternMatchingEntity.CreateListEntity("floorName",
EntityMatchMode.Strict, "ground floor", "lobby", "1st", "first", "one", "1",
"2nd", "second", "two", "2"));

            // Creates the "parkingLevel" entity as a pre-built integer

```

```
model.Entities.Add(PatternMatchingEntity.CreateIntegerEntity("parkingLevel"))
);

    var modelCollection = new
LanguageUnderstandingModelCollection();
    modelCollection.Add(model);

    recognizer.ApplyLanguageModels(modelCollection);

    Console.WriteLine("Say something...");

    var result = await recognizer.RecognizeOnceAsync();

    if (result.Reason == ResultReason.RecognizedIntent)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"          Intent Id={result.IntentId}.");

        var entities = result.Entities;
        switch (result.IntentId)
        {
            case "ChangeFloors":
                if (entities.TryGetValue("floorName", out string
floorName))
                {
                    Console.WriteLine($"          FloorName={floorName}");
                }

                if (entities.TryGetValue("floorName:1", out
floorName))
                {
                    Console.WriteLine($"          FloorName:1={floorName}");
                }

                if (entities.TryGetValue("floorName:2", out
floorName))
                {
                    Console.WriteLine($"          FloorName:2={floorName}");
                }

                if (entities.TryGetValue("parkingLevel", out
string parkingLevel))
                {
                    Console.WriteLine($"          ParkingLevel={parkingLevel}");
                }
        }

        break;

        case "DoorControl":
            if (entities.TryGetValue("action", out string
```

```
action))
    {
        Console.WriteLine($"          Action={action}");
    }
    break;
}
}
else if (result.Reason == ResultReason.RecognizedSpeech)
{
    Console.WriteLine($"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 set the speech
resource key and region values?");
    }
}
}
}
```

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**.

E5

3. Start recognition - It will prompt you to say something. The default language is English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

For example if you say "Take me to floor 2", this should be the output:

```
text

Say something...
RECOGNIZED: Text=Take me to floor 2.
Intent Id=ChangeFloors.
FloorName=2
```

As another example if you say "Take me to floor 7", this should be the output:

```
text

Say something...
RECOGNIZED: Text=Take me to floor 7.
Intent not recognized.
```

No intent was recognized because 7 was not in our list of valid values for floorName.

SDK, REST, and CLI developer resources for Language Understanding (LUIS)

Article • 11/18/2022 • 3 minutes to read

Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

SDKs, REST APIs, CLI, help you develop Language Understanding (LUIS) apps in your programming language. Manage your Azure resources and LUIS predictions.

Azure resource management

Use the Azure Cognitive Services Management layer to create, edit, list, and delete the Language Understanding or Cognitive Service resource.

Find reference documentation based on the tool:

- [Azure CLI](#)
- [Azure RM PowerShell](#)

Language Understanding authoring and prediction requests

The Language Understanding service is accessed from an Azure resource you need to create.

There are two resources:

- Use the **authoring** resource for training to create, edit, train, and publish.
- Use the **prediction** for runtime to send user's text and receive a prediction.

Learn about the [V3 prediction endpoint](#).

Use [Cognitive Services sample code](#) to learn and use the most common tasks.

REST specifications

The [LUIS REST specifications](#), along with all [Azure REST specifications](#), are publicly available on GitHub.

REST APIs

Both authoring and prediction endpoint APIS are available from REST APIs:

Type	Version
Authoring	V2 ↗ preview V3 ↗
Prediction	V2 ↗ V3 ↗

REST Endpoints

LUIS currently has 2 types of endpoints:

- **authoring** on the training endpoint
- query **prediction** on the runtime endpoint.

Purpose	URL
V2 Authoring on training endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/api/v2.0/apps/{{appID}}/
V3 Authoring on training endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/authoring/v3.0-preview/apps/{{appID}}/
V2 Prediction - all predictions on runtime endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/v2.0/apps/{{appId}}?q={{q}}&timezoneOffset[&verbose][&spellCheck][&staging][&bing-spell-check-subscription-key][&log]
V3 Prediction - versions prediction on runtime endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/{{appId}}/versions/{{versionId}}/predict?query={{query}}[&verbose][&log][&show-all-intents]
V3 Prediction - slot prediction on runtime endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/{{appId}}/slots/{{slotName}}/predict?query={{query}}[&verbose][&log][&show-all-intents]

The following table explains the parameters, denoted with curly braces {}, in the previous table.

Parameter	Purpose
<code>your-resource-name</code>	Azure resource name
<code>q</code> or <code>query</code>	utterance text sent from client application such as chat bot
<code>version</code>	10 character version name
<code>slot</code>	<code>production</code> or <code>staging</code>

REST query string parameters

V3 API query string parameters include:

Query parameter	Luis portal name	Type	Version	Default	Purpose
<code>log</code>	Save logs	boolean	V2 & V3	false	Store query in log file. Default value is false.
<code>query</code>	-	string	V3 only	No default - it is required	In V2, the utterance to be predicted is in the <code>q</code> parameter. In V3, the functionality is passed in the <code>query</code> parameter. GET request
<code>show-all-intents</code>	Include scores for all intents	boolean	V3 only	false	Return all intents with the corresponding score in the <code>prediction.intents</code> object. Intents are returned as objects in a parent <code>intents</code> object. This allows programmatic access without needing to find the intent in an array: <code>prediction.intents.give</code> . In V2, these were returned in an array.
<code>verbose</code>	Include more entities details	boolean	V2 & V3	false	In V2, when set to true, all predicted intents were returned. If you need all predicted intents, use the V3 param of <code>show-all-intents</code> . In V3, this parameter only provides entity metadata details of entity prediction.
<code>timezoneOffset</code>	-	string	V2	-	Timezone applied to datetimeV2 entities.
<code>datetimeReference</code>	-	string	V3	-	Timezone applied to datetimeV2 entities. Replaces <code>timezoneOffset</code> from V2.

App schema

The [app schema](#) is imported and exported in a `.json` or `.lu` format.

Language-based SDKs

Language	Reference documentation	Package	Quickstarts
C#	Authoring Prediction	NuGet authoring ↗ NuGet prediction ↗	Authoring Query prediction
Go	Authoring and prediction ↗	SDK ↗	
Java	Authoring and prediction	Maven authoring ↗ Maven prediction ↗	
JavaScript	Authoring Prediction	NPM authoring ↗ NPM prediction ↗	Authoring Prediction
Python	Authoring and prediction	Pip ↗	Authoring Prediction

Containers

Language Understanding (LUIS) provides a [container](#) to provide on-premises and contained versions of your app.

Export and import formats

Language Understanding provides the ability to manage your app and its models in a JSON format, the `.LU` ([LUDown](#) ↗) format, and a compressed package for the Language Understanding container.

Importing and exporting these formats is available from the APIs and from the LUIS portal. The portal provides import and export as part of the Apps list and Versions list.

Workshops

- GitHub: (Workshop) [Conversational-AI : NLU using LUIS](#) ↗

Continuous integration tools

- GitHub: (Preview) [Developing a LUIS app using DevOps practices](#) ↗
- GitHub: [NLU.DevOps](#) ↗ - Tools supporting continuous integration and deployment for NLU services.

Bot Framework tools

The bot framework is available as [an SDK](#) in a variety of languages and as a service using [Azure Bot Service](#).

Bot framework provides [several tools](#) to help with Language Understanding, including:

- [Bot Framework emulator](#) - a desktop application that allows bot developers to test and debug bots built using the Bot Framework SDK
- [Bot Framework Composer](#) - an integrated development tool for developers and multi-disciplinary teams to build bots and conversational experiences with the Microsoft Bot Framework
- [Bot Framework Samples](#) - in #C, JavaScript, TypeScript, and Python

Next steps

- Learn about the common [HTTP error codes](#)
- [Reference documentation](#) for all APIs and SDKs
- [Bot framework](#) and [Azure Bot Service](#)
- [LUDown](#)
- [Cognitive Containers](#)

Language and region support for LUIS

Article • 10/03/2022 • 5 minutes to read

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

LUIS has a variety of features within the service. Not all features are at the same language parity. Make sure the features you are interested in are supported in the language culture you are targeting. A LUIS app is culture-specific and cannot be changed once it is set.

Multilingual LUIS apps

If you need a multilingual LUIS client application such as a chatbot, you have a few options. If LUIS supports all the languages, you develop a LUIS app for each language. Each LUIS app has a unique app ID, and endpoint log. If you need to provide language understanding for a language LUIS does not support, you can use the [Translator service](#) to translate the utterance into a supported language, submit the utterance to the LUIS endpoint, and receive the resulting scores.

ⓘ Note

A newer version of Language Understanding capabilities is now available as part of Azure Cognitive Service for Language. For more information, see [Azure Cognitive Service for Language Documentation](#). For language understanding capabilities that support multiple languages within the Language Service, see [Conversational Language Understanding](#).

Languages supported

LUIS understands utterances in the following languages:

Language	Locale	Prebuilt domain	Prebuilt entity	Phrase list recommendations	**Sentiment analysis and key phrase extraction
Arabic (preview - modern standard Arabic)	ar-AR	-	-	-	-
*Chinese	zh-CN	✓	✓	✓	-
Dutch	nl-NL	✓	-	-	✓
English (United States)	en-US	✓	✓	✓	✓
English (UK)	en-GB	✓	✓	✓	✓
French (Canada)	fr-CA	-	-	-	✓
French (France)	fr-FR	✓	✓	✓	✓
German	de-DE	✓	✓	✓	✓
Gujarati (preview)	gu-IN	-	-	-	-
Hindi (preview)	hi-IN	-	✓	-	-
Italian	it-IT	✓	✓	✓	✓
*Japanese	ja-JP	✓	✓	✓	Key phrase only
Korean	ko-KR	✓	-	-	Key phrase only
Marathi (preview)	mr-IN	-	-	-	-
Portuguese (Brazil)	pt-BR	✓	✓	✓	not all sub-cultures
Spanish (Mexico)	es-MX	-	✓	✓	✓
Spanish (Spain)	es-ES	✓	✓	✓	✓
Tamil (preview)	ta-IN	-	-	-	-
Telugu (preview)	te-IN	-	-	-	-
Turkish	tr-TR	✓	✓	-	Sentiment only

Language support varies for [prebuilt entities](#) and [prebuilt domains](#).

*Chinese support notes

- In the `zh-CN` culture, LUIS expects the simplified Chinese character set instead of the traditional character set.
- The names of intents, entities, features, and regular expressions may be in Chinese or Roman characters.
- See the [prebuilt domains reference](#) for information on which prebuilt domains are supported in the `zh-CN` culture.

*Japanese support notes

- Because LUIS does not provide syntactic analysis and will not understand the difference between Keigo and informal Japanese, you need to incorporate the different levels of formality as training examples for your applications.
 - `でございます` is not the same as `です`.
 - `です` is not the same as `だ`.

**Language service support notes

The Language service includes keyPhrase prebuilt entity and sentiment analysis. Only Portuguese is supported for subcultures: `pt-PT` and `pt-BR`. All other cultures are supported at the primary culture level.

Speech API supported languages

See [Speech Supported languages](#) for Speech dictation mode languages.

Bing Spell Check supported languages

See [Bing Spell Check Supported languages](#) for a list of supported languages and status.

Rare or foreign words in an application

In the `en-us` culture, LUIS learns to distinguish most English words, including slang. In the `zh-cn` culture, LUIS learns to distinguish most Chinese characters. If you use a rare word in `en-us` or character in `zh-cn`, and you see that LUIS seems unable to distinguish that word or character, you can add that word or character to a [phrase-list feature](#). For example, words outside of the culture of the application -- that is, foreign words -- should be added to a phrase-list feature.

Hybrid languages

Hybrid languages combine words from two cultures such as English and Chinese. These languages are not supported in LUIS because an app is based on a single culture.

Tokenization

To perform machine learning, LUIS breaks an utterance into **tokens** based on culture.

Language	every space or special character	character level	compound words
Arabic	✓		
Chinese		✓	
Dutch	✓		✓
English (en-us)	✓		
English (en-GB)	✓		
French (fr-FR)	✓		
French (fr-CA)	✓		
German	✓		✓
Gujarati	✓		
Hindi	✓		
Italian	✓		
Japanese			✓
Korean		✓	
Marathi	✓		
Portuguese (Brazil)	✓		
Spanish (es-ES)	✓		
Spanish (es-MX)	✓		
Tamil	✓		
Telugu	✓		
Turkish	✓		

Custom tokenizer versions

The following cultures have custom tokenizer versions:

Culture	Version	Purpose
German de-de	1.0.0	Tokenizes words by splitting them using a machine learning-based tokenizer that tries to break down composite words into their single components. If a user enters <code>Ich fahre einen krankenwagen</code> as an utterance, it is turned to <code>Ich fahre einen kranken wagen</code> . Allowing the marking of <code>kranken</code> and <code>wagen</code> independently as different entities.
German de-de	1.0.2	Tokenizes words by splitting them on spaces. If a user enters <code>Ich fahre einen krankenwagen</code> as an utterance, it remains a single token. Thus <code>krankenwagen</code> is marked as a single entity.
Dutch nl-nl	1.0.0	Tokenizes words by splitting them using a machine learning-based tokenizer that tries to break down composite words into their single components. If a user enters <code>Ik ga naar de kleuterschool</code> as an utterance, it is turned to <code>Ik ga naar de kleuter school</code> . Allowing the marking of <code>kleuter</code> and <code>school</code> independently as different entities.
Dutch nl-nl	1.0.1	Tokenizes words by splitting them on spaces. If a user enters <code>Ik ga naar de kleuterschool</code> as an utterance, it remains a single token. Thus <code>kleuterschool</code> is marked as a single entity.

Migrating between tokenizer versions

Tokenization happens at the app level. There is no support for version-level tokenization.

[Import the file as a new app](#), instead of a version. This action means the new app has a different app ID but uses the tokenizer version specified in the file.

What is speaker recognition?

Article • 11/30/2022 • 4 minutes to read

Speaker recognition can help determine who is speaking in an audio clip. The service can verify and identify speakers by their unique voice characteristics, by using voice biometry.

You provide audio training data for a single speaker, which creates an enrollment profile based on the unique characteristics of the speaker's voice. You can then cross-check audio voice samples against this profile to verify that the speaker is the same person (speaker verification). You can also cross-check audio voice samples against a *group* of enrolled speaker profiles to see if it matches any profile in the group (speaker identification).

ⓘ Important

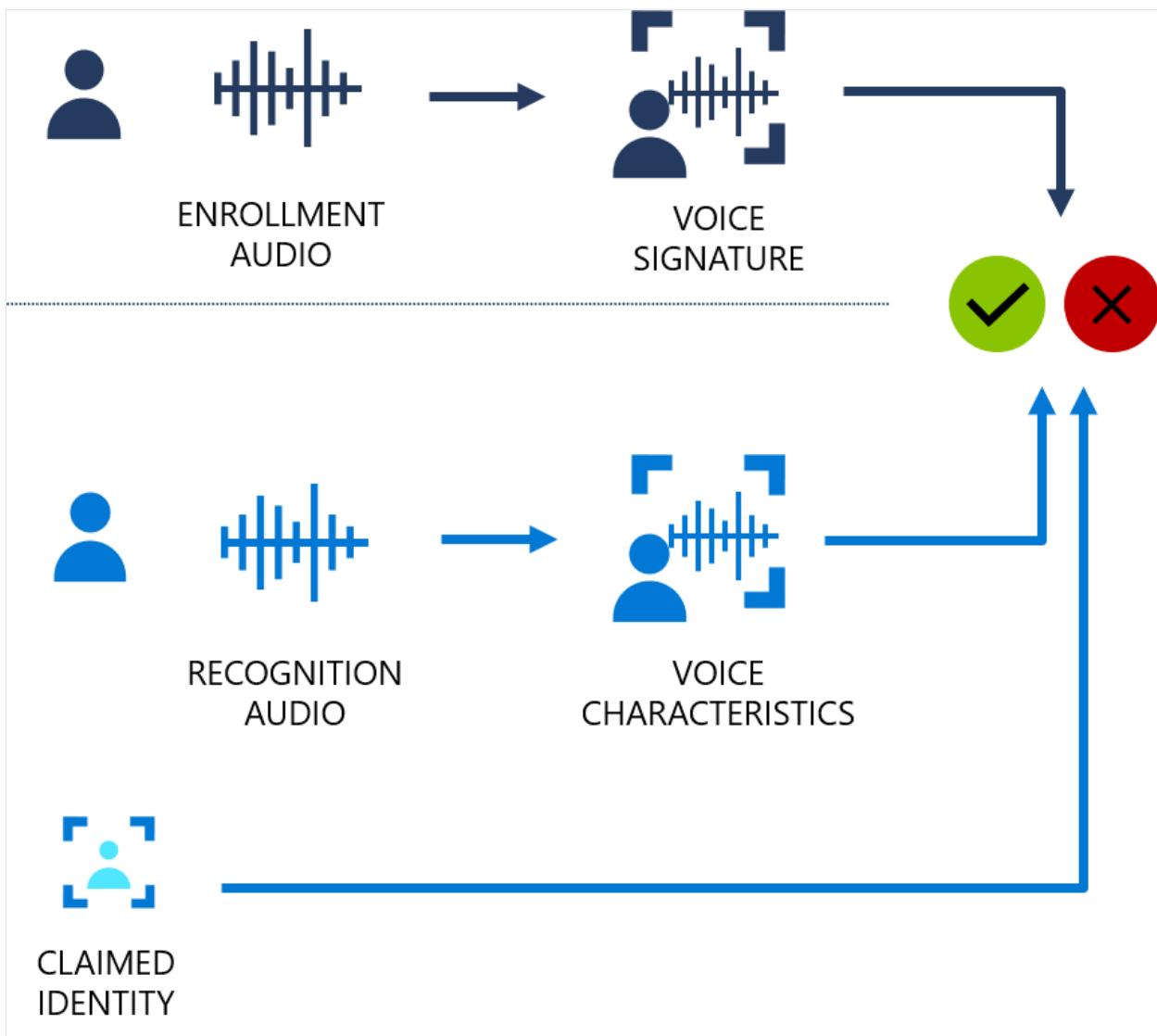
Microsoft limits access to speaker recognition. You can apply for access through the [Azure Cognitive Services speaker recognition limited access review](#). For more information, see [Limited access for speaker recognition](#).

Speaker verification

Speaker verification streamlines the process of verifying an enrolled speaker identity with either passphrases or free-form voice input. For example, you can use it for customer identity verification in call centers or contactless facility access.

How does speaker verification work?

The following flowchart provides a visual of how this works:



Speaker verification can be either text-dependent or text-independent. *Text-dependent* verification means that speakers need to choose the same passphrase to use during both enrollment and verification phases. *Text-independent* verification means that speakers can speak in everyday language in the enrollment and verification phrases.

For text-dependent verification, the speaker's voice is enrolled by saying a passphrase from a set of predefined phrases. Voice features are extracted from the audio recording to form a unique voice signature, and the chosen passphrase is also recognized. Together, the voice signature and the passphrase are used to verify the speaker.

Text-independent verification has no restrictions on what the speaker says during enrollment, besides the initial activation phrase when active enrollment is enabled. It doesn't have any restrictions on the audio sample to be verified, because it only extracts voice features to score similarity.

The APIs aren't intended to determine whether the audio is from a live person, or from an imitation or recording of an enrolled speaker.

Speaker identification

Speaker identification helps you determine an unknown speaker's identity within a group of enrolled speakers. Speaker identification enables you to attribute speech to individual speakers, and unlock value from scenarios with multiple speakers, such as:

- Supporting solutions for remote meeting productivity.
- Building multi-user device personalization.

How does speaker identification work?

Enrollment for speaker identification is text-independent. There are no restrictions on what the speaker says in the audio, besides the initial activation phrase when active enrollment is enabled. Similar to speaker verification, the speaker's voice is recorded in the enrollment phase, and the voice features are extracted to form a unique voice signature. In the identification phase, the input voice sample is compared to a specified list of enrolled voices (up to 50 in each request).

Data security and privacy

Speaker enrollment data is stored in a secured system, including the speech audio for enrollment and the voice signature features. The speech audio for enrollment is only used when the algorithm is upgraded, and the features need to be extracted again. The service doesn't retain the speech recording or the extracted voice features that are sent to the service during the recognition phase.

You control how long data should be retained. You can create, update, and delete enrollment data for individual speakers through API calls. When the subscription is deleted, all the speaker enrollment data associated with the subscription will also be deleted.

As with all of the Cognitive Services resources, developers who use the speaker recognition feature must be aware of Microsoft policies on customer data. You should ensure that you have received the appropriate permissions from the users. You can find more details in [Data and privacy for speaker recognition](#). For more information, see the [Cognitive Services page](#) on the Microsoft Trust Center.

Common questions and solutions

Question	Solution
----------	----------

Question	Solution
What situations am I most likely to use speaker recognition?	Good examples include call center customer verification, voice-based patient check-in, meeting transcription, and multi-user device personalization.
What's the difference between identification and verification?	Identification is the process of detecting which member from a group of speakers is speaking. Verification is the act of confirming that a speaker matches a known, <i>enrolled</i> voice.
What languages are supported?	See Speaker recognition language support .
What Azure regions are supported?	See Speaker recognition region support .
What audio formats are supported?	Mono 16 bit, 16 kHz PCM-encoded WAV.
Can you enroll one speaker multiple times?	Yes, for text-dependent verification, you can enroll a speaker up to 50 times. For text-independent verification or speaker identification, you can enroll with up to 300 seconds of audio.
What data is stored in Azure?	Enrollment audio is stored in the service until the voice profile is deleted . Recognition audio samples aren't retained or stored.

Next steps

[Speaker recognition quickstart](#)

Quickstart: Recognize and verify who is speaking

Article • 02/20/2022 • 39 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this quickstart, you learn basic design patterns for speaker recognition by using the Speech SDK, including:

- Text-dependent and text-independent verification.
- Speaker identification to identify a voice sample among a group of voices.
- Deleting voice profiles.

For a high-level look at speaker recognition concepts, see the [Overview](#) article. See the **Reference** node in the left pane for a list of the supported platforms.

ⓘ Important

Microsoft limits access to speaker recognition. Apply to use it through the [Azure Cognitive Services Speaker Recognition Limited Access Review](#) form. After approval, you can access the Speaker Recognition APIs.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

Install the Speech SDK

Before you start, you must install the Speech SDK. Depending on your platform, use the following instructions:

- [.NET Framework](#)
- [.NET Core](#)
- [Unity](#)
- [UWP](#)

- Xamarin

Import dependencies

To run the examples in this article, include the following `using` statements at the top of your script:

```
C#
```

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
```

Create a speech configuration

To call the Speech service by using the Speech SDK, you need to create a [SpeechConfig](#) instance. In this example, you create a `SpeechConfig` instance by using a subscription key and region. You also create some basic boilerplate code to use for the rest of this article, which you modify for different customizations.

 **Important**

Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like [Azure Key Vault](#). See the Cognitive Services [security](#) article for more information.

```
C#
```

```
public class Program
{
    static async Task Main(string[] args)
    {
        // replace with your own subscription key
        string subscriptionKey = "YourSubscriptionKey";
        // replace with your own subscription region
        string region = "YourSubscriptionRegion";
        var config = SpeechConfig.FromSubscription(subscriptionKey, region);
    }
}
```

Text-dependent verification

Speaker verification is the act of confirming that a speaker matches a known, or *enrolled*, voice. The first step is to enroll a voice profile so that the service has something to compare future voice samples against. In this example, you enroll the profile by using a *text-dependent* strategy, which requires a specific passphrase to use for enrollment and verification. See the [reference docs](#) for a list of supported passphrases.

Start by creating the following function in your `Program` class to enroll a voice profile:

C#

```
public static async Task VerificationEnroll(SpeechConfig config,
Dictionary<string, string> profileMapping)
{
    using (var client = new VoiceProfileClient(config))
    using (var profile = await
client.CreateProfileAsync(VoiceProfileType.TextDependentVerification, "en-
us"))
    {
        var phraseResult = await
client.GetActivationPhrasesAsync(VoiceProfileType.TextDependentVerification,
"en-us");
        using (var audioInput = AudioConfig.FromDefaultMicrophoneInput())
        {
            Console.WriteLine($"Enrolling profile id {profile.Id}.");
            // give the profile a human-readable display name
            profileMapping.Add(profile.Id, "Your Name");

            VoiceProfileEnrollmentResult result = null;
            while (result is null || result.RemainingEnrollmentsCount > 0)
            {
                Console.WriteLine($"Speak the passphrase,
${phraseResult.Phrases[0]}");
                result = await client.EnrollProfileAsync(profile,
audioInput);
                Console.WriteLine($"Remaining enrollments needed:
{result.RemainingEnrollmentsCount}");
                Console.WriteLine("");
            }

            if (result.Reason == ResultReason.EnrolledVoiceProfile)
            {
                await SpeakerVerify(config, profile, profileMapping);
            }
            else if (result.Reason == ResultReason.Canceled)
            {
                var cancellation =
VoiceProfileEnrollmentCancellationDetails.FromResult(result);
                Console.WriteLine($"CANCELED {profile.Id}: ErrorCode=
{cancellation.ErrorCode} ErrorDetails={cancellation.ErrorDetails}");
            }
        }
    }
}
```

```
        }  
    }  
}
```

In this function, `await client.CreateProfileAsync()` is what actually creates the new voice profile. After it's created, you specify how you'll input audio samples by using `AudioConfig.FromDefaultMicrophoneInput()` in this example to capture audio from your default input device. Next, you enroll audio samples in a `while` loop that tracks the number of samples remaining, and that are required, for enrollment. In each iteration, `client.EnrollProfileAsync(profile, audioInput)` prompts you to speak the passphrase into your microphone and adds the sample to the voice profile.

After enrollment is finished, call `await SpeakerVerify(config, profile, profileMapping)` to verify against the profile you just created. Add another function to define `SpeakerVerify`.

C#

```
public static async Task SpeakerVerify(SpeechConfig config, VoiceProfile profile, Dictionary<string, string> profileMapping)  
{  
    var speakerRecognizer = new SpeakerRecognizer(config,  
    AudioConfig.FromDefaultMicrophoneInput());  
    var model = SpeakerVerificationModel.FromProfile(profile);  
  
    Console.WriteLine("Speak the passphrase to verify: \"My voice is my  
    passport, please verify me.\");  
    var result = await speakerRecognizer.RecognizeOnceAsync(model);  
    Console.WriteLine($"Verified voice profile for speaker  
    {profileMapping[result.ProfileId]}, score is {result.Score}");  
}
```

In this function, you pass the `VoiceProfile` object you just created to initialize a model to verify against. Next, `await speakerRecognizer.RecognizeOnceAsync(model)` prompts you to speak the passphrase again. This time it validates it against your voice profile and returns a similarity score that ranges from 0.0 to 1.0. The `result` object also returns `Accept` or `Reject`, based on whether the passphrase matches.

Next, modify your `Main()` function to call the new functions you created. Also, note that you create a `Dictionary<string, string>` to pass by reference through your function calls. The reason for this is that the service doesn't allow storing a human-readable name with a created `VoiceProfile`, and it only stores an ID number for privacy purposes. In the `VerificationEnroll` function, you add to this dictionary an entry with the newly created ID, along with a text name. In application development scenarios

where you need to display a human-readable name, *you must store this mapping somewhere because the service can't store it.*

C#

```
static async Task Main(string[] args)
{
    string subscriptionKey = "YourSubscriptionKey";
    string region = "westus";
    var config = SpeechConfig.FromSubscription(subscriptionKey, region);

    // persist profileMapping if you want to store a record of who the
    // profile is
    var profileMapping = new Dictionary<string, string>();
    await VerificationEnroll(config, profileMapping);

    Console.ReadLine();
}
```

Run the script. You're prompted to speak the phrase "My voice is my passport, verify me" three times for enrollment, and one more time for verification. The result returned is the similarity score, which you can use to create your own custom thresholds for verification.

shell

```
Enrolling profile id 87-2cef-4dff-995b-dcefb64e203f.
Speak the passphrase, "My voice is my passport, verify me."
Remaining enrollments needed: 2

Speak the passphrase, "My voice is my passport, verify me."
Remaining enrollments needed: 1

Speak the passphrase, "My voice is my passport, verify me."
Remaining enrollments needed: 0

Speak the passphrase to verify: "My voice is my passport, verify me."
Verified voice profile for speaker Your Name, score is 0.915581
```

Text-independent verification

In contrast to *text-dependent* verification, *text-independent* verification doesn't require three audio samples but *does* require 20 seconds of total audio.

Make a couple simple changes to your `VerificationEnroll` function to switch to *text-independent* verification. First, you change the verification type to `VoiceProfileType.TextIndependentVerification`. Next, change the `while` loop to track

`result.RemainingEnrollmentsSpeechLength`, which will continue to prompt you to speak until 20 seconds of audio have been captured.

C#

```
public static async Task VerificationEnroll(SpeechConfig config,
Dictionary<string, string> profileMapping)
{
    using (var client = new VoiceProfileClient(config))
    using (var profile = await
client.CreateProfileAsync(VoiceProfileType.TextIndependentVerification, "en-
us"))
    {
        var phraseResult = await
client.GetActivationPhrasesAsync(VoiceProfileType.TextIndependentVerificatio
n, "en-us");
        using (var audioInput = AudioConfig.FromDefaultMicrophoneInput())
        {
            Console.WriteLine($"Enrolling profile id {profile.Id}.");
            // give the profile a human-readable display name
            profileMapping.Add(profile.Id, "Your Name");

            VoiceProfileEnrollmentResult result = null;
            while (result is null || result.RemainingEnrollmentsSpeechLength
> TimeSpan.Zero)
            {
                Console.WriteLine($"Speak the activation phrase,
\"${phraseResult.Phrases[0]}\\"");
                result = await client.EnrollProfileAsync(profile,
audioInput);
                Console.WriteLine($"Remaining enrollment audio time needed:
{result.RemainingEnrollmentsSpeechLength}");
                Console.WriteLine("");
            }

            if (result.Reason == ResultReason.EnrolledVoiceProfile)
            {
                await SpeakerVerify(config, profile, profileMapping);
            }
            else if (result.Reason == ResultReason.Canceled)
            {
                var cancellation =
VoiceProfileEnrollmentCancellationDetails.FromResult(result);
                Console.WriteLine($"CANCELED {profile.Id}: ErrorCode=
{cancellation.ErrorCode} ErrorDetails={cancellation.ErrorDetails}");
            }
        }
    }
}
```

Run the program again, and the similarity score is returned.

shell

```
Enrolling profile id 4tt87d4-f2d3-44ae-b5b4-f1a8d4036ee9.  
Speak the activation phrase, "<FIRST ACTIVATION PHRASE>"  
Remaining enrollment audio time needed: 00:00:15.3200000  
  
Speak the activation phrase, "<FIRST ACTIVATION PHRASE>"  
Remaining enrollment audio time needed: 00:00:09.8100008  
  
Speak the activation phrase, "<FIRST ACTIVATION PHRASE>"  
Remaining enrollment audio time needed: 00:00:05.1900000  
  
Speak the activation phrase, "<FIRST ACTIVATION PHRASE>"  
Remaining enrollment audio time needed: 00:00:00.8700000  
  
Speak the activation phrase, "<FIRST ACTIVATION PHRASE>"  
Remaining enrollment audio time needed: 00:00:00  
  
Speak the passphrase to verify: "My voice is my passport, please verify me."  
Verified voice profile for speaker Your Name, score is 0.849409
```

Speaker identification

Speaker identification is used to determine *who* is speaking from a given group of enrolled voices. The process is similar to *text-independent verification*. The main difference is the capability to verify against multiple voice profiles at once rather than verifying against a single profile.

Create a function `IdentificationEnroll` to enroll multiple voice profiles. The enrollment process for each profile is the same as the enrollment process for *text-independent verification*. The process requires 20 seconds of audio for each profile. This function accepts a list of strings `profileNames` and will create a new voice profile for each name in the list. The function returns a list of `VoiceProfile` objects, which you use in the next function for identifying a speaker.

C#

```
public static async Task<List<VoiceProfile>>  
IdentificationEnroll(SpeechConfig config, List<string> profileNames,  
Dictionary<string, string> profileMapping)  
{  
    List<VoiceProfile> voiceProfiles = new List<VoiceProfile>();  
    using (var client = new VoiceProfileClient(config))  
    {  
        var phraseResult = await  
client.GetActivationPhrasesAsync(VoiceProfileType.TextIndependentVerificatio  
n, "en-us");  
        foreach (string name in profileNames)
```

```

    {
        using (var audioInput =
AudioConfig.FromDefaultMicrophoneInput())
        {
            var profile = await
client.CreateProfileAsync(VoiceProfileType.TextIndependentIdentification,
"en-us");
            Console.WriteLine($"Creating voice profile for {name}.");
            profileMapping.Add(profile.Id, name);

            VoiceProfileEnrollmentResult result = null;
            while (result is null ||
result.RemainingEnrollmentsSpeechLength > TimeSpan.Zero)
            {
                Console.WriteLine($"Speak the activation phrase,
\"${phraseResult.Phrases[0]}\" to add to the profile enrollment sample for
{name}.");
                result = await client.EnrollProfileAsync(profile,
audioInput);
                Console.WriteLine($"Remaining enrollment audio time
needed: {result.RemainingEnrollmentsSpeechLength}");
                Console.WriteLine("");
            }
            voiceProfiles.Add(profile);
        }
    }
    return voiceProfiles;
}

```

Create the following function `SpeakerIdentification` to submit an identification request. The main difference in this function compared to a *speaker verification* request is the use of `SpeakerIdentificationModel.FromProfiles()`, which accepts a list of `VoiceProfile` objects.

C#

```

public static async Task SpeakerIdentification(SpeechConfig config,
List<VoiceProfile> voiceProfiles, Dictionary<string, string> profileMapping)
{
    var speakerRecognizer = new SpeakerRecognizer(config,
AudioConfig.FromDefaultMicrophoneInput());
    var model = SpeakerIdentificationModel.FromProfiles(voiceProfiles);

    Console.WriteLine("Speak some text to identify who it is from your list
of enrolled speakers.");
    var result = await speakerRecognizer.RecognizeOnceAsync(model);
    Console.WriteLine($"The most similar voice profile is
{profileMapping[result.ProfileId]} with similarity score {result.Score}");
}

```

Change your `Main()` function to the following. You create a list of strings `profileNames`, which you pass to your `IdentificationEnroll()` function. You're prompted to create a new voice profile for each name in this list, so you can add more names to create more profiles for friends or colleagues.

C#

```
static async Task Main(string[] args)
{
    // replace with your own subscription key
    string subscriptionKey = "YourSubscriptionKey";
    // replace with your own subscription region
    string region = "YourSubscriptionRegion";
    var config = SpeechConfig.FromSubscription(subscriptionKey, region);

    // persist profileMapping if you want to store a record of who the
    profile is
    var profileMapping = new Dictionary<string, string>();
    var profileNames = new List<string>() { "Your name", "A friend's name"
};

    var enrolledProfiles = await IdentificationEnroll(config, profileNames,
profileMapping);
    await SpeakerIdentification(config, enrolledProfiles, profileMapping);

    foreach (var profile in enrolledProfiles)
    {
        profile.Dispose();
    }
    Console.ReadLine();
}
```

Run the script. You're prompted to speak to enroll voice samples for the first profile. After the enrollment is finished, you're prompted to repeat this process for each name in the `profileNames` list. After each enrollment is finished, you're prompted to have *anyone* speak. The service then attempts to identify this person from among your enrolled voice profiles.

This example returns only the closest match and its similarity score. To get the full response that includes the top five similarity scores, add `string json = result.Properties.GetProperty(PropertyId.SpeechServiceResponse_JsonResult)` to your `SpeakerIdentification` function.

Change audio input type

The examples in this article use the default device microphone as input for audio samples. In scenarios where you need to use audio files instead of microphone input, change any instance of `AudioConfig.FromDefaultMicrophoneInput()` to `AudioConfig.FromWavFileInput(path/to/your/file.wav)` to switch to a file input. You can also have mixed inputs by using a microphone for enrollment and files for verification, for example.

Delete voice profile enrollments

To delete an enrolled profile, use the `DeleteProfileAsync()` function on the `VoiceProfileClient` object. The following example function shows how to delete a voice profile from a known voice profile ID:

C#

```
public static async Task DeleteProfile(SpeechConfig config, string
profileId)
{
    using (var client = new VoiceProfileClient(config))
    {
        var profile = new VoiceProfile(profileId);
        await client.DeleteProfileAsync(profile);
    }
}
```

Next steps

[See the quickstart samples on GitHub](#)

What is keyword recognition?

Article • 11/18/2022 • 6 minutes to read

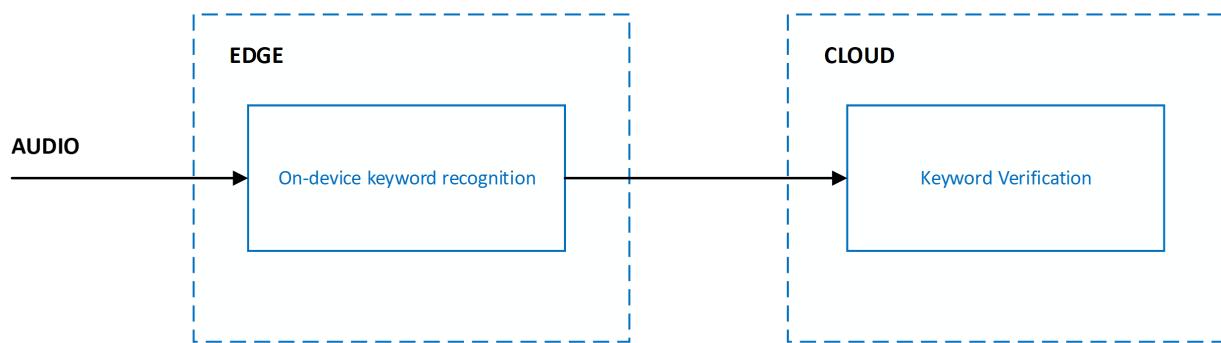
Keyword recognition detects a word or short phrase within a stream of audio. It's also referred to as keyword spotting.

The most common use case of keyword recognition is voice activation of virtual assistants. For example, "Hey Cortana" is the keyword for the Cortana assistant. Upon recognition of the keyword, a scenario-specific action is carried out. For virtual assistant scenarios, a common resulting action is speech recognition of audio that follows the keyword.

Generally, virtual assistants are always listening. Keyword recognition acts as a privacy boundary for the user. A keyword requirement acts as a gate that prevents unrelated user audio from crossing the local device to the cloud.

To balance accuracy, latency, and computational complexity, keyword recognition is implemented as a multistage system. For all stages beyond the first, audio is only processed if the stage prior to it believed to have recognized the keyword of interest.

The current system is designed with multiple stages that span the edge and cloud:



Accuracy of keyword recognition is measured via the following metrics:

- **Correct accept rate:** Measures the system's ability to recognize the keyword when it's spoken by a user. The correct accept rate is also known as the true positive rate.
- **False accept rate:** Measures the system's ability to filter out audio that isn't the keyword spoken by a user. The false accept rate is also known as the false positive rate.

The goal is to maximize the correct accept rate while minimizing the false accept rate. The current system is designed to detect a keyword or phrase preceded by a short amount of silence. Detecting a keyword in the middle of a sentence or utterance isn't supported.

Custom keyword for on-device models

With the [Custom Keyword portal on Speech Studio](#), you can generate keyword recognition models that execute at the edge by specifying any word or short phrase. You can further personalize your keyword model by choosing the right pronunciations.

Pricing

There's no cost to use custom keyword to generate models, including both Basic and Advanced models. There's also no cost to run models on-device with the Speech SDK when used in conjunction with other Speech service features such as speech-to-text.

Types of models

You can use custom keyword to generate two types of on-device models for any keyword.

Model type	Description
Basic	Best suited for demo or rapid prototyping purposes. Models are generated with a common base model and can take up to 15 minutes to be ready. Models might not have optimal accuracy characteristics.
Advanced	Best suited for product integration purposes. Models are generated with adaptation of a common base model by using simulated training data to improve accuracy characteristics. It can take up to 48 hours for models to be ready.

Note

You can view a list of regions that support the **Advanced** model type in the [keyword recognition region support](#) documentation.

Neither model type requires you to upload training data. Custom keyword fully handles data generation and model training.

Pronunciations

When you create a new model, custom keyword automatically generates possible pronunciations of the provided keyword. You can listen to each pronunciation and choose all variations that closely represent the way you expect users to say the keyword. All other pronunciations shouldn't be selected.

It's important to be deliberate about the pronunciations you select to ensure the best accuracy characteristics. For example, if you choose more pronunciations than you need, you might get higher false accept rates. If you choose too few pronunciations, where not all expected variations are covered, you might get lower correct accept rates.

Test models

After on-device models are generated by custom keyword, they can be tested directly on the portal. You can use the portal to speak directly into your browser and get keyword recognition results.

Keyword verification

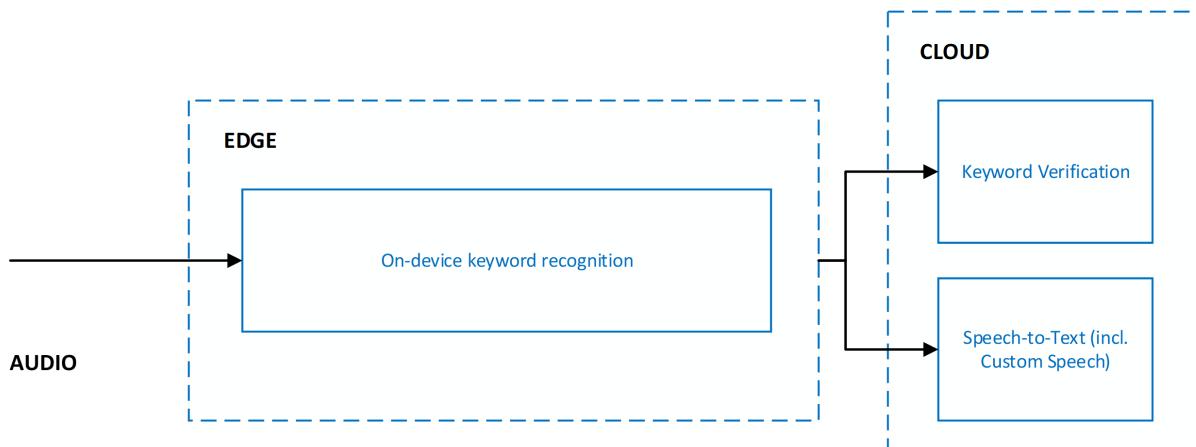
Keyword verification is a cloud service that reduces the impact of false accepts from on-device models with robust models running on Azure. Tuning or training isn't required for keyword verification to work with your keyword. Incremental model updates are continually deployed to the service to improve accuracy and latency and are transparent to client applications.

Pricing

Keyword verification is always used in combination with speech-to-text. There's no cost to use keyword verification beyond the cost of speech-to-text.

Keyword verification and speech-to-text

When keyword verification is used, it's always in combination with speech-to-text. Both services run in parallel, which means audio is sent to both services for simultaneous processing.



Running keyword verification and speech-to-text in parallel yields the following benefits:

- **No other latency on speech-to-text results:** Parallel execution means that keyword verification adds no latency. The client receives speech-to-text results as quickly. If keyword verification determines the keyword wasn't present in the audio, speech-to-text processing is terminated. This action protects against unnecessary speech-to-text processing. Network and cloud model processing increases the user-perceived latency of voice activation. For more information, see [Recommendations and guidelines](#).
- **Forced keyword prefix in speech-to-text results:** Speech-to-text processing ensures that the results sent to the client are prefixed with the keyword. This behavior allows for increased accuracy in the speech-to-text results for speech that follows the keyword.
- **Increased speech-to-text timeout:** Because of the expected presence of the keyword at the beginning of audio, speech-to-text allows for a longer pause of up to five seconds after the keyword before it determines the end of speech and terminates speech-to-text processing. This behavior ensures that the user experience is correctly handled for staged commands (`<keyword> <pause> <command>`) and chained commands (`<keyword> <command>`).

Keyword verification responses and latency considerations

For each request to the service, keyword verification returns one of two responses: accepted or rejected. The processing latency varies depending on the length of the keyword and the length of the audio segment expected to contain the keyword. Processing latency doesn't include network cost between the client and Azure Speech services.

Keyword verification response	Description
Accepted	Indicates the service believed that the keyword was present in the audio stream provided as part of the request.
Rejected	Indicates the service believed that the keyword wasn't present in the audio stream provided as part of the request.

Rejected cases often yield higher latencies as the service processes more audio than accepted cases. By default, keyword verification processes a maximum of two seconds of audio to search for the keyword. If the keyword is determined not to be present in two seconds, the service times out and signals a rejected response to the client.

Use keyword verification with on-device models from custom keyword

The Speech SDK enables seamless use of on-device models generated by using custom keyword with keyword verification and speech-to-text. It transparently handles:

- Audio gating to keyword verification and speech recognition based on the outcome of an on-device model.
- Communicating the keyword to keyword verification.
- Communicating any more metadata to the cloud for orchestrating the end-to-end scenario.

You don't need to explicitly specify any configuration parameters. All necessary information is automatically extracted from the on-device model generated by custom keyword.

The sample and tutorials linked here show how to use the Speech SDK:

- [Voice assistant samples on GitHub ↗](#)
- [Tutorial: Voice enable your assistant built using Azure Bot Service with the C# Speech SDK](#)
- [Tutorial: Create a custom commands application with simple voice commands](#)

Speech SDK integration and scenarios

The Speech SDK enables easy use of personalized on-device keyword recognition models generated with custom keyword and keyword verification. To ensure that your product needs can be met, the SDK supports the following two scenarios:

Scenario	Description	Samples
----------	-------------	---------

Scenario	Description	Samples
End-to-end keyword recognition with speech-to-text	Best suited for products that will use a customized on-device keyword model from custom keyword with Azure Speech keyword verification and speech-to-text. This scenario is the most common.	<ul style="list-style-type: none"> • Voice assistant sample code ↗ • Tutorial: Voice enable your assistant built using Azure Bot Service with the C# Speech SDK • Tutorial: Create a custom commands application with simple voice commands
Offline keyword recognition	Best suited for products without network connectivity that will use a customized on-device keyword model from custom keyword.	<ul style="list-style-type: none"> • C# on Windows UWP sample ↗ • Java on Android sample ↗

Next steps

- [Read the quickstart to generate on-device keyword recognition models using custom keyword](#)
- [Learn more about voice assistants](#)

Quickstart: Create a custom keyword

Article • 02/20/2022 • 13 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this quickstart, you learn the basics of working with custom keywords. A keyword is a word or short phrase, which allows your product to be voice activated. You create keyword models in Speech Studio. Then export a model file that you use with the Speech SDK in your applications.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

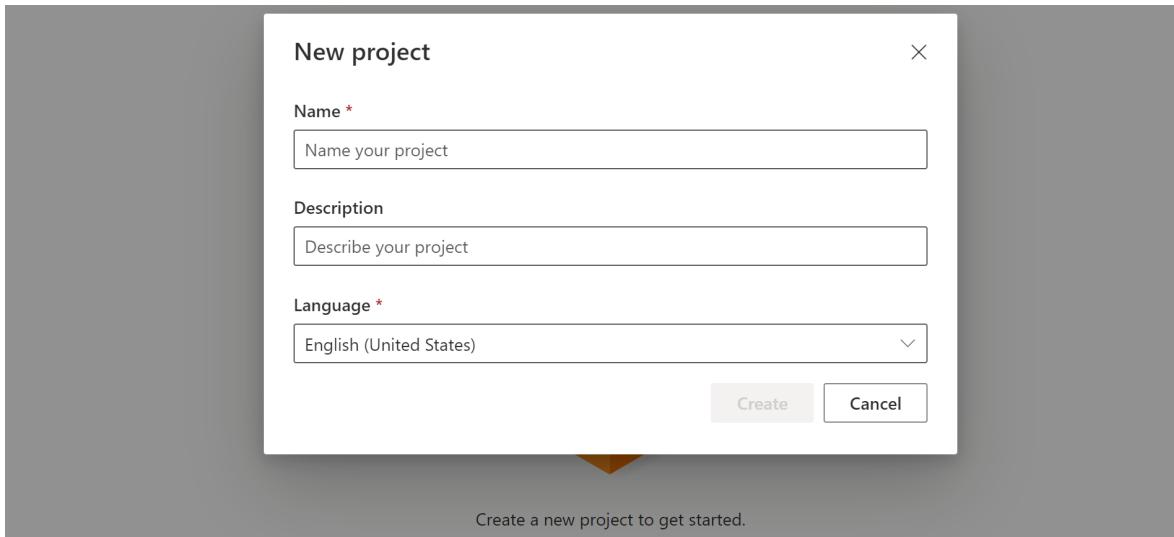
Create a keyword in Speech Studio

Before you can use a custom keyword, you need to create a keyword using the [Custom Keyword](#) page on [Speech Studio](#). After you provide a keyword, it produces a `.table` file that you can use with the Speech SDK.

ⓘ Important

Custom keyword models, and the resulting `.table` files, can **only** be created in Speech Studio. You cannot create custom keywords from the SDK or with REST calls.

1. Go to the [Speech Studio](#) and **Sign in**. If you don't have a speech subscription, go to [Create Speech Services](#).
2. On the [Custom Keyword](#) page, select **Create a new project**.
3. Enter a **Name**, **Description**, and **Language** for your custom keyword project. You can only choose one language per project, and support is currently limited to English (United States) and Chinese (Mandarin, Simplified).



Create a new project to get started.

4. Select your project's name from the list.

Speech Studio > Custom Keyword

Select the project you want to work with

Name	Description	Language	Training	Created ↓
My custom keyword project	This is an example for the documentation	English (United States)	1	11/11/2021 6:46 AM

5. To create a custom keyword for your virtual assistant, select **Create a new model**.

6. Enter a **Name** for the model, **Description**, and **Keyword** of your choice, then select **Next**. See the [guidelines](#) on choosing an effective keyword.

Custom Keyword

My custom keyword project

English (United States)

Models

Test model

Tune model

Create a new model

New model

Name and keyword

Consider the following guidelines when choosing your keyword:
The word should be 4 to 7 syllables. For example, "Hey Computer" is a great choice.
Do not choose a common word. For example, "eat" and "go". Do not use special characters, symbols, or digits.

Name *

My custom keyword model

Description

This is an example for the documentation

Keyword *

Hey Computer

Next Cancel

7. The portal creates candidate pronunciations for your keyword. Listen to each candidate by selecting the play buttons and remove the checks next to any pronunciations that are incorrect. Select all pronunciations that correspond to how you expect your users to say the keyword and then select **Next** to begin generating the keyword model.



8. Select a model type, then select **Create**. You can view a list of regions that support the **Advanced** model type in the [Keyword recognition region support documentation](#).
9. It may take up to 30 minutes for the model to be generated. The keyword list will change from **Processing** to **Succeeded** when the model is complete.

10. From the collapsible menu on the left, select **Tune** for options to tune and download your model. The downloaded file is a `.zip` archive. Extract the archive, and you see a file with the `.table` extension. You use the `.table` file with the SDK, so make sure to note its path.

Use a keyword model with the Speech SDK

First, load your keyword model file using the `FromFile()` static function, which returns a `KeywordRecognitionModel`. Use the path to the `.table` file you downloaded from Speech Studio. Additionally, you create an `AudioConfig` using the default microphone, then instantiate a new `KeywordRecognizer` using the audio configuration.

C#

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
```

```
var keywordModel =  
    KeywordRecognitionModel.FromFile("your/path/to/Activate_device.table");  
using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();  
using var keywordRecognizer = new KeywordRecognizer(audioConfig);
```

Next, running keyword recognition is done with one call to `RecognizeOnceAsync()` by passing your model object. This starts a keyword recognition session that lasts until the keyword is recognized. Thus, you generally use this design pattern in multi-threaded applications, or in use cases where you may be waiting for a wake-word indefinitely.

C#

```
KeywordRecognitionResult result = await  
    keywordRecognizer.RecognizeOnceAsync(keywordModel);
```

ⓘ Note

The example shown here uses local keyword recognition, since it does not require a `SpeechConfig` object for authentication context, and does not contact the back-end. However, you can run both keyword recognition and verification **utilizing a direct back-end connection**.

Continuous recognition

Other classes in the Speech SDK support continuous recognition (for both speech and intent recognition) with keyword recognition. This allows you to use the same code you would normally use for continuous recognition, with the ability to reference a `.table` file for your keyword model.

For speech-to-text, follow the same design pattern shown in the [recognize speech guide](#) to set up continuous recognition. Then, replace the call to

`recognizer.StartContinuousRecognitionAsync()` with

`recognizer.StartKeywordRecognitionAsync(``KeywordRecognitionModel``)`, and pass your `KeywordRecognitionModel` object. To stop continuous recognition with keyword recognition, use `recognizer.StopKeywordRecognitionAsync()` instead of `recognizer.StopContinuousRecognitionAsync()`.

Intent recognition uses an identical pattern with the [StartKeywordRecognitionAsync](#) and [StopKeywordRecognitionAsync](#) functions.

Next steps

[Get the Speech SDK](#)

Recommendations and guidelines for keyword recognition

Article • 08/25/2022 • 2 minutes to read

This article outlines how to choose your keyword optimize its accuracy characteristics and how to design your user experiences with Keyword Verification.

Choosing an effective keyword

Creating an effective keyword is vital to ensuring your product will consistently and accurately respond. Consider the following guidelines when you choose a keyword.

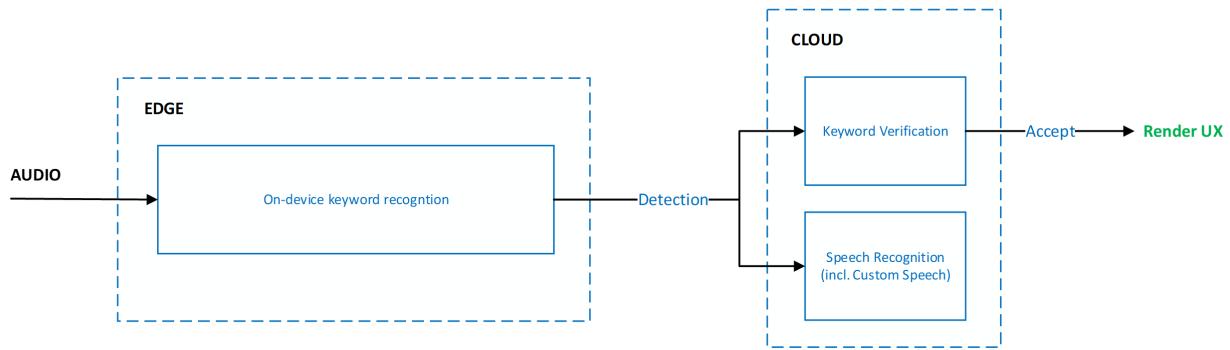
Note

The examples below are in English but the guidelines apply to all languages supported by Custom Keyword. For a list of all supported languages, see [Language support](#).

- 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 pronunciation rules specific to the native language of your end-users.
- A unique or even a made-up word that follows common 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 lead to higher than desired false accept rates for your product.
- Avoid using a keyword that might have alternative pronunciations. Users would have to know the "right" pronunciation to get their product to voice activate. 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 "/līv/" 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.

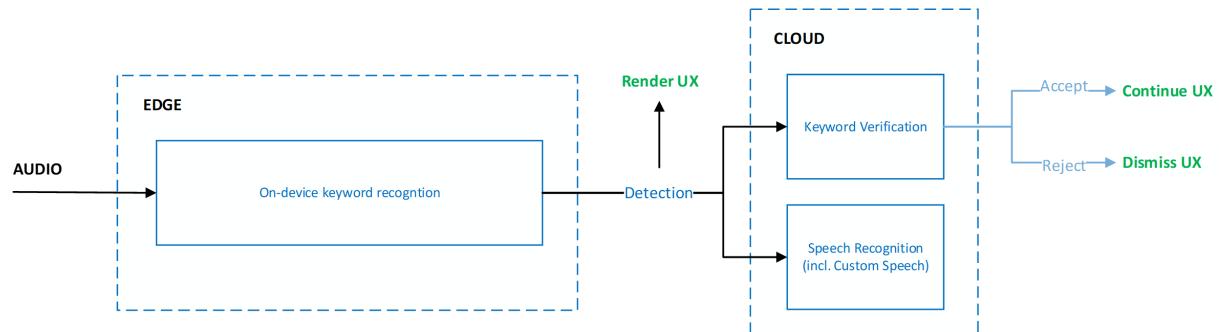
User experience recommendations with Keyword Verification

With a multi-stage keyword recognition scenario where [Keyword Verification](#) is used, applications can choose when the end-user is notified of a keyword detection. The recommendation for rendering any visual or audible indicator is to rely upon on responses from the Keyword Verification service:



This ensures the optimal experience in terms of accuracy to minimize the user-perceived impact of false accepts but incurs additional latency.

For applications that require latency optimization, applications can provide light and unobtrusive indicators to the end-user based on the on-device keyword recognition. For example, lighting an LED pattern or pulsing an icon. The indicators can continue to exist if Keyword Verification responds with a keyword accept, or can be dismissed if the response is a keyword reject:



Next steps

- [Get the Speech SDK.](#)
- [Learn more about Voice Assistants.](#)

Captioning with speech to text

Article • 10/17/2022 • 6 minutes to read

In this guide, you learn how to create captions with speech to text. Captioning is the process of converting the audio content of a television broadcast, webcast, film, video, live event, or other production into text, and then displaying the text on a screen, monitor, or other visual display system.

Concepts include how to synchronize captions with your input audio, apply profanity filters, get partial results, apply customizations, and identify spoken languages for multilingual scenarios. This guide covers captioning for speech, but doesn't include speaker ID or sound effects such as bells ringing.

Here are some common captioning scenarios:

- Online courses and instructional videos
- Sporting events
- Voice and video calls

The following are aspects to consider when using captioning:

- Let your audience know that captions are generated by an automated service.
- Center captions horizontally on the screen, in a large and prominent font.
- Consider whether to use partial results, when to start displaying captions, and how many words to show at a time.
- Learn about captioning protocols such as [SMPTE-TT](#).
- Consider output formats such as SRT (SubRip Text) and WebVTT (Web Video Text Tracks). These can be loaded onto most video players such as VLC, automatically adding the captions on to your video.

💡 Tip

Try the [Speech Studio](#) and choose a sample video clip to see real-time or offline processed captioning results.

Try the [Azure Video Indexer](#) as a demonstration of how you can get captions for videos that you upload.

Captioning can accompany real time or pre-recorded speech. Whether you're showing captions in real time or with a recording, you can use the [Speech SDK](#) or [Speech CLI](#) to recognize speech and get transcriptions. You can also use the [Batch transcription API](#) for pre-recorded video.

Caption output format

The Speech service supports output formats such as SRT (SubRip Text) and WebVTT (Web Video Text Tracks). These can be loaded onto most video players such as VLC, automatically adding the captions on to your video.

💡 Tip

The Speech service provides **profanity filter** options. You can specify whether to mask, remove, or show profanity.

The [SRT](#) (SubRip Text) timespan output format is `hh:mm:ss,fff`.

```
srt  
  
1  
00:00:00,180 --> 00:00:03,230  
Welcome to applied Mathematics course 201.
```

The [WebVTT](#) (Web Video Text Tracks) timespan output format is `hh:mm:ss,fff`.

```
WEBVTT  
  
00:00:00.180 --> 00:00:03.230  
Welcome to applied Mathematics course 201.  
{  
    "ResultId": "8e89437b4b9349088a933f8db4ccc263",  
    "Duration": "00:00:03.0500000"  
}
```

Input audio to the Speech service

For real time captioning, use a microphone or audio input stream instead of file input. For examples of how to recognize speech from a microphone, see the [Speech to text quickstart](#) and [How to recognize speech](#) documentation. For more information about streaming, see [How to use the audio input stream](#).

For captioning of a prerecording, send file input to the Speech service. For more information, see [How to use compressed input audio](#).

Caption and speech synchronization

You'll want to synchronize captions with the audio track, whether it's done in real time or with a prerecording.

The Speech service returns the offset and duration of the recognized speech.

- **Offset:** The offset into the audio stream being recognized, expressed as duration. Offset is measured in ticks, starting from 0 (zero) tick, associated with the first audio byte processed by the SDK. For example, the offset begins when you start recognition, since that's when the SDK starts processing the audio stream. One tick represents one hundred nanoseconds or one ten-millionth of a second.
- **Duration:** Duration of the utterance that is being recognized. The duration in ticks doesn't include trailing or leading silence.

For more information, see [Get speech recognition results](#).

Get partial results

Consider when to start displaying captions, and how many words to show at a time. Speech recognition results are subject to change while an utterance is still being recognized. Partial results are returned with each `Recognizing` event. As each word is processed, the Speech service re-evaluates an utterance in the new context and again returns the best result. The new result isn't guaranteed to be the same as the previous result. The complete and final transcription of an utterance is returned with the `Recognized` event.

Note

Punctuation of partial results is not available.

For captioning of prerecorded speech or wherever latency isn't a concern, you could wait for the complete transcription of each utterance before displaying any words. Given the final offset and duration of each word in an utterance, you know when to show subsequent words at pace with the soundtrack.

Real time captioning presents tradeoffs with respect to latency versus accuracy. You could show the text from each `Recognizing` event as soon as possible. However, if you can accept some latency, you can improve the accuracy of the caption by displaying the text from the `Recognized` event. There's also some middle ground, which is referred to as "stable partial results".

You can request that the Speech service return fewer `Recognizing` events that are more accurate. This is done by setting the

`SpeechServiceResponse_StablePartialResultThreshold` property to a value between `0` and `2147483647`. The value that you set is the number of times a word has to be recognized before the Speech service returns a `Recognizing` event. For example, if you set the `SpeechServiceResponse_StablePartialResultThreshold` property value to `5`, the Speech service will affirm recognition of a word at least five times before returning the partial results to you with a `Recognizing` event.

C#

```
speechConfig SetProperty(PropertyId.SpeechServiceResponse_StablePartialResultThreshold, 5);
```

Requesting more stable partial results will reduce the "flickering" or changing text, but it can increase latency as you wait for higher confidence results.

Stable partial threshold example

In the following recognition sequence without setting a stable partial threshold, "math" is recognized as a word, but the final text is "mathematics". At another point, "course 2" is recognized, but the final text is "course 201".

Console

```
RECOGNIZING: Text=welcome to
RECOGNIZING: Text=welcome to applied math
RECOGNIZING: Text=welcome to applied mathematics
RECOGNIZING: Text=welcome to applied mathematics course 2
RECOGNIZING: Text=welcome to applied mathematics course 201
RECOGNIZED: Text=Welcome to applied Mathematics course 201.
```

In the previous example, the transcriptions were additive and no text was retracted. But at other times you might find that the partial results were inaccurate. In either case, the unstable partial results can be perceived as "flickering" when displayed.

For this example, if the stable partial result threshold is set to `5`, no words are altered or backtracked.

Console

```
RECOGNIZING: Text=welcome to
RECOGNIZING: Text=welcome to applied
```

RECOGNIZING: Text=welcome to applied mathematics
RECOGNIZED: Text=Welcome to applied Mathematics course 201.

Language identification

If the language in the audio could change, use continuous [language identification](#). Language identification is used to identify languages spoken in audio when compared against a list of [supported languages](#). You provide up to 10 candidate languages, at least one of which is expected to be in the audio. The Speech service returns the most likely language in the audio.

Customizations to improve accuracy

A [phrase list](#) is a list of words or phrases that you provide right before starting speech recognition. Adding a phrase to a phrase list increases its importance, thus making it more likely to be recognized.

Examples of phrases include:

- Names
- Geographical locations
- Homonyms
- Words or acronyms unique to your industry or organization

There are some situations where [training a custom model](#) is likely the best option to improve accuracy. For example, if you're captioning orthodontics lectures, you might want to train a custom model with the corresponding domain data.

Next steps

- [Captioning quickstart](#)
- [Get speech recognition results](#)

Quickstart: Create captions with speech to text

Article • 05/08/2022 • 37 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) | [Additional Samples on GitHub](#)

In this quickstart, you run a console app to create [captions](#) with speech to text.

💡 Tip

Try the [Speech Studio](#) and choose a sample video clip to see real-time or offline processed captioning results.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

Set up the environment

The Speech SDK is available as a [NuGet package](#) and implements .NET Standard 2.0. You install the Speech SDK later in this guide, but first check the [SDK installation guide](#) for any more requirements.

You must also install [GStreamer](#) for compressed input audio.

Set environment variables

Your application must be authenticated to access Cognitive Services resources. For production, use a secure way of storing and accessing your credentials. For example, after you [get a key for your Speech resource](#), write it to a new environment variable on the local machine running the application.

💡 Tip

Don't include the key directly in your code, and never post it publicly. See the Cognitive Services [security](#) article for more authentication options like [Azure Key Vault](#).

To set the environment variable for your Speech resource key, open a console window, and follow the instructions for your operating system and development environment. To set the `SPEECH_KEY` environment variable, replace `your-key` with one of the keys for your resource.

Windows

Console

```
setx SPEECH_KEY your-key
```

 **Note**

If you only need to access the environment variable in the current running console, you can set the environment variable with `set` instead of `setx`.

After you add the environment variable, you may need to restart any running programs that will need to read the environment variable, including the console window. For example, if you are using Visual Studio as your editor, restart Visual Studio before running the example.

To set the environment variable for your Speech resource region, follow the same steps. Set `SPEECH_REGION` to the region of your resource. For example, `westus`.

Create captions from speech

Follow these steps to build and run the captioning quickstart code example.

1. Copy the [scenarios/csharp/dotnetcore/captioning/](#) sample files from GitHub. If you have [Git installed](#), open a command prompt and run the `git clone` command to download the Speech SDK samples repository.

.NET CLI

```
git clone https://github.com/Azure-Samples/cognitive-services-speech-sdk.git
```

2. Open a command prompt and change to the project directory.

```
.NET CLI
```

```
cd <your-local-path>/scenarios/csharp/dotnetcore/captioning/captioning/
```

3. Build the project with the .NET CLI.

```
.NET CLI
```

```
dotnet build
```

4. Run the application with your preferred command line arguments. See [usage and arguments](#) for the available options. Here is an example:

```
.NET CLI
```

```
dotnet run --input caption.this.mp4 --format any --output
caption.output.txt --srt --realTime --threshold 5 --delay 0 --profanity
mask --phrases "Contoso;Jessie;Rehaan"
```

Important

Make sure that the paths specified by `--input` and `--output` are valid. Otherwise you must change the paths.

Make sure that you set the `SPEECH_KEY` and `SPEECH_REGION` environment variables as described [above](#). Otherwise use the `--key` and `--region` arguments.

Check results

When you use the `realTime` option in the example above, the partial results from `Recognizing` events are included in the output. In this example, only the final `Recognized` event includes the commas. Commas aren't the only differences between `Recognizing` and `Recognized` events. For more information, see [Get partial results](#).

```
srt
```

```
1  
00:00:00,170 --> 00:00:00,380
```

```
The
```

```
2
```

```
00:00:00,380 --> 00:00:01,770
The rainbow

3
00:00:01,770 --> 00:00:02,560
The rainbow has seven

4
00:00:02,560 --> 00:00:03,820
The rainbow has seven colors

5
00:00:03,820 --> 00:00:05,050
The rainbow has seven colors red

6
00:00:05,050 --> 00:00:05,850
The rainbow has seven colors red
orange

7
00:00:05,850 --> 00:00:06,440
The rainbow has seven colors red
orange yellow

8
00:00:06,440 --> 00:00:06,730
The rainbow has seven colors red
orange yellow green

9
00:00:06,730 --> 00:00:07,160
orange, yellow, green, blue,
indigo and Violet.
```

When you use the `--offline` option, the results are stable from the final `Recognized` event. Partial results aren't included in the output:

```
srt

1
00:00:00,170 --> 00:00:05,540
The rainbow has seven colors, red,
orange, yellow, green, blue,

2
00:00:05,540 --> 00:00:07,160
indigo and Violet.
```

The [SRT](#) (SubRip Text) timespan output format is `hh:mm:ss,fff`. For more information, see [Caption output format](#).

Usage and arguments

Usage: `captioning --input <input file>`

Connection options include:

- `--key`: Your Speech resource key. Overrides the SPEECH_KEY environment variable. You must set the environment variable (recommended) or use the `--key` option.
- `--region REGION`: Your Speech resource region. Overrides the SPEECH_REGION environment variable. You must set the environment variable (recommended) or use the `--region` option. Examples: `westus`, `northeurope`

Input options include:

- `--input FILE`: Input audio from file. The default input is the microphone.
- `--format FORMAT`: Use compressed audio format. Valid only with `--file`. Valid values are `alaw`, `any`, `flac`, `mp3`, `mulaw`, and `ogg_opus`. The default value is `any`. To use a `wav` file, don't specify the format. This option is not available with the JavaScript captioning sample. For compressed audio files such as MP4, install GStreamer and see [How to use compressed input audio](#).

Language options include:

- `--language LANG`: Specify a language using one of the corresponding [supported locales](#). This is used when breaking captions into lines. Default value is `en-US`.

Recognition options include:

- `--offline`: Output offline results. Overrides `--realTime`. Default output mode is offline.
- `--realTime`: Output real-time results.

Real-time output includes `Recognizing` event results. The default offline output is `Recognized` event results only. These are always written to the console, never to an output file. The `--quiet` option overrides this. For more information, see [Get speech recognition results](#).

Accuracy options include:

- `--phrases PHRASE1;PHRASE2`: You can specify a list of phrases to be recognized, such as `Contoso;Jessie;Rehaan`. For more information, see [Improve recognition with phrase list](#).

Output options include:

- `--help`: Show this help and stop
- `--output FILE`: Output captions to the specified `file`. This flag is required.
- `--srt`: Output captions in SRT (SubRip Text) format. The default format is WebVTT (Web Video Text Tracks). For more information about SRT and WebVTT caption file formats, see [Caption output format](#).
- `--maxLength LENGTH`: Set the maximum number of characters per line for a caption to `LENGTH`. Minimum is 20. Default is 37 (30 for Chinese).
- `--lines LINES`: Set the number of lines for a caption to `LINES`. Minimum is 1. Default is 2.
- `--delay MILLISECONDS`: How many MILLISECONDS to delay the display of each caption, to mimic a real-time experience. This option is only applicable when you use the `realTime` flag. Minimum is 0.0. Default is 1000.
- `--remainTime MILLISECONDS`: How many MILLISECONDS a caption should remain on screen if it is not replaced by another. Minimum is 0.0. Default is 1000.
- `--quiet`: Suppress console output, except errors.
- `--profanity OPTION`: Valid values: raw, remove, mask. For more information, see [Profanity filter concepts](#).
- `--threshold NUMBER`: Set stable partial result threshold. The default value is 3. This option is only applicable when you use the `realTime` flag. For more information, see [Get partial results concepts](#).

Clean up resources

You can use the [Azure portal](#) or [Azure Command Line Interface \(CLI\)](#) to remove the Speech resource you created.

Next steps

[Learn more about speech recognition](#)

Call Center Overview

Article • 10/18/2022 • 3 minutes to read

Azure Cognitive Services for Language and Speech can help you realize partial or full automation of telephony-based customer interactions, and provide accessibility across multiple channels. With the Language and Speech services, you can further analyze call center transcriptions, extract and redact conversation personally identifiable information (PII), summarize the transcription, and detect the sentiment.

Some example scenarios for the implementation of Azure Cognitive Services in call and contact centers are:

- Virtual agents: Conversational AI-based telephony-integrated voicebots and voice-enabled chatbots
- Agent-assist: Real-time transcription and analysis of a call to improve the customer experience by providing insights and suggest actions to agents
- Post-call analytics: Post-call analysis to create insights into customer conversations to improve understanding and support continuous improvement of call handling, optimization of quality assurance and compliance control as well as other insight driven optimizations.

💡 Tip

Try the [Language Studio](#) or [Speech Studio](#) for a demonstration on how to use the Language and Speech services to analyze call center conversations.

To deploy a call center transcription solution to Azure with a no-code approach, try the [Ingestion Client](#).

Cognitive Services features for call centers

A holistic call center implementation typically incorporates technologies from the Language and Speech services.

Audio data typically used in call centers generated through landlines, mobile phones, and radios is often narrowband, in the range of 8 KHz, which can create challenges when you're converting speech to text. The Speech service recognition models are trained to ensure that you can get high-quality transcriptions, however you choose to capture the audio.

Once you've transcribed your audio with the Speech service, you can use the Language service to perform analytics on your call center data such as: sentiment analysis, summarizing the reason for customer calls, how they were resolved, extracting and redacting conversation PII, and more.

Speech service

The Speech service offers the following features that can be used for call center use cases:

- **Real-time speech-to-text:** Recognize and transcribe audio in real-time from multiple inputs. For example, with virtual agents or agent-assist, you can continuously recognize audio input and control how to process results based on multiple events.
- **Batch speech-to-text:** Transcribe large amounts of audio files asynchronously including speaker diarization and is typically used in post-call analytics scenarios. Diarization is the process of recognizing and separating speakers in mono channel audio data.
- **Text-to-speech:** Text-to-speech enables your applications, tools, or devices to convert text into humanlike synthesized speech.
- **Speaker identification:** Helps you determine an unknown speaker's identity within a group of enrolled speakers and is typically used for call center customer verification scenarios or fraud detection.
- **Language Identification:** Identify languages spoken in audio and can be used in real-time and post-call analysis for insights or to control the environment (such as output language of a virtual agent).

The Speech service works well with prebuilt models. However, you might want to further customize and tune the experience for your product or environment. Typical examples for Speech customization include:

Speech customization	Description
Custom Speech	A speech-to-text feature used evaluate and improve the speech recognition accuracy of use-case specific entities (such as alpha-numeric customer, case, and contract IDs, license plates, and names). You can also train a custom model with your own product names and industry terminology.
Custom Neural Voice	A text-to-speech feature that lets you create a one-of-a-kind, customized, synthetic voice for your applications.

Language service

The Language service offers the following features that can be used for call center use cases:

- **Personally Identifiable Information (PII) extraction and redaction:** Identify, categorize, and redact sensitive information in conversation transcription.
- **Conversation summarization:** Summarize in abstract text what each conversation participant said about the issues and resolutions. For example, a call center can group product issues that have a high volume.
- **Sentiment analysis and opinion mining:** Analyze transcriptions and associate positive, neutral, or negative sentiment at the utterance and conversation-level.

While the Language service works well with prebuilt models, you might want to further customize and tune models to extract more information from your data. Typical examples for Language customization include:

Language customization	Description
Custom NER (named entity recognition)	Improve the detection and extraction of entities in transcriptions.
Custom text classification	Classify and label transcribed utterances with either single or multiple classifications.

You can find an overview of all Language service features and customization options [here](#).

Next steps

- [Post-call transcription and analytics quickstart](#)
- [Try out the Language Studio ↗](#)
- [Try out the Speech Studio ↗](#)

Quickstart: Post-call transcription and analytics

Article • 09/23/2022 • 8 minutes to read

[Language service documentation](#) | [Language Studio](#) | [Speech service documentation](#) | [Speech Studio](#)

In this C# quickstart, you perform sentiment analysis and conversation summarization of [call center](#) transcriptions. The sample will automatically identify, categorize, and redact sensitive information. The quickstart implements a cross-service scenario that uses features of the [Azure Cognitive Speech](#) and [Azure Cognitive Language](#) services.

Tip

Try the [Language Studio](#) or [Speech Studio](#) for a demonstration on how to use the Language and Speech services to analyze call center conversations.

To deploy a call center transcription solution to Azure with a no-code approach, try the [Ingestion Client](#).

The following Azure Cognitive Services for Speech features are used in the quickstart:

- [Batch transcription](#): Submit a batch of audio files for transcription.
- [Speaker separation](#): Separate multiple speakers through diarization of mono 16khz 16 bit PCM wav files.

The Language service offers the following features that are used:

- [Personally Identifiable Information \(PII\) extraction and redaction](#): Identify, categorize, and redact sensitive information in conversation transcription.
- [Conversation summarization](#): Summarize in abstract text what each conversation participant said about the issues and resolutions. For example, a call center can group product issues that have a high volume.
- [Sentiment analysis and opinion mining](#): Analyze transcriptions and associate positive, neutral, or negative sentiment at the utterance and conversation-level.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)

- ✓ Create a Cognitive Services multi-service resource [↗](#) in the Azure portal. This quickstart only requires one Cognitive Services multi-service resource. The sample code allows you to specify separate [Language ↗](#) and [Speech ↗](#) resource keys.
- ✓ Get the resource key and region. After your Cognitive Services resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

 **Important**

This quickstart requires access to [conversation summarization](#). To get access, you must submit an [online request ↗](#) and have it approved.

The `--languageKey` and `--languageEndpoint` values in this quickstart must correspond to a resource that's in one of the regions supported by the [conversation summarization API ↗](#): `eastus`, `northeurope`, and `uksouth`.

Run post-call transcription analysis with C#

Follow these steps to build and run the post-call transcription analysis quickstart code example.

1. Copy the [scenarios/csharp/dotnetcore/call-center/↗](#) sample files from GitHub. If you have [Git installed ↗](#), open a command prompt and run the `git clone` command to download the Speech SDK samples repository.

.NET CLI

```
git clone https://github.com/Azure-Samples/cognitive-services-speech-
sdk.git
```

2. Open a command prompt and change to the project directory.

.NET CLI

```
cd <your-local-path>/scenarios/csharp/dotnetcore/call-center/call-
center/
```

3. Build the project with the .NET CLI.

.NET CLI

```
dotnet build
```

4. Run the application with your preferred command line arguments. See [usage and arguments](#) for the available options.

Here's an example that transcribes from an example audio file at [GitHub](#):

.NET CLI

```
dotnet run --languageKey YourResourceKey --languageEndpoint  
YourResourceEndpoint --speechKey YourResourceKey --speechRegion  
YourResourceRegion --input "https://github.com/Azure-Samples/cognitive-  
services-speech-sdk/raw/master/scenarios/call-  
center/sampleddata/Call1_separated_16k_health_insurance.wav" --stereo -  
-output summary.json
```

If you already have a transcription for input, here's an example that only requires a Language resource:

.NET CLI

```
dotnet run --languageKey YourResourceKey --languageEndpoint  
YourResourceEndpoint --jsonInput "YourTranscriptionFile.json" --stereo  
--output summary.json
```

Replace `YourResourceKey` with your Cognitive Services resource key, replace `YourResourceRegion` with your Cognitive Services resource [region](#) (such as `eastus`), and replace `YourResourceEndpoint` with your Cognitive Services endpoint. Make sure that the paths specified by `--input` and `--output` are valid. Otherwise you must change the paths.

Important

Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like [Azure Key Vault](#). See the Cognitive Services [security](#) article for more information.

Check results

The console output shows the full conversation and summary. Here's an example of the overall summary, with redactions for brevity:

Output

Conversation summary:

```
issue: Customer wants to sign up for insurance.  
resolution: Customer was advised that customer would be contacted by the  
insurance company.
```

If you specify the `--output FILE` optional argument, a JSON version of the results are written to the file. The file output is a combination of the JSON responses from the [batch transcription](#) (Speech), [sentiment](#) (Language), and [conversation summarization](#) (Language) APIs.

The `transcription` property contains a JSON object with the results of sentiment analysis merged with batch transcription. Here's an example, with redactions for brevity:

JSON

```
{  
    "source": "https://github.com/Azure-Samples/cognitive-services-speech-  
sdk/raw/master/scenarios/call-  
center/sampleddata/Call1_separated_16k_health_insurance.wav",  
    // Example results redacted for brevity  
    "nBest": [  
        {  
            "confidence": 0.77464247,  
            "lexical": "hello thank you for calling contoso who am i  
speaking with today",  
            "itn": "hello thank you for calling contoso who am i speaking  
with today",  
            "maskedITN": "hello thank you for calling contoso who am i  
speaking with today",  
            "display": "Hello, thank you for calling Contoso. Who am I  
speaking with today?",  
            "sentiment": {  
                "positive": 0.78,  
                "neutral": 0.21,  
                "negative": 0.01  
            }  
        },  
    ]  
    // Example results redacted for brevity  
}
```

The `conversationAnalyticsResults` property contains a JSON object with the results of the conversation PII and conversation summarization analysis. Here's an example, with redactions for brevity:

JSON

```
{
  "conversationAnalyticsResults": {
    "conversationSummaryResults": {
      "conversations": [
        {
          "id": "conversation1",
          "summaries": [
            {
              "aspect": "issue",
              "text": "Customer wants to sign up for insurance"
            },
            {
              "aspect": "resolution",
              "text": "Customer was advised that customer would be contacted by the insurance company"
            }
          ],
          "warnings": []
        }
      ],
      "errors": [],
      "modelVersion": "2022-05-15-preview"
    },
    "conversationPiiResults": {
      "combinedRedactedContent": [
        {
          "channel": "0",
          "display": "Hello, thank you for calling Contoso. Who am I speaking with today? Hi, ****. Uh, are you calling because you need health insurance?", // Example results redacted for brevity
          "itn": "hello thank you for calling contoso who am i speaking with today hi **** uh are you calling because you need health insurance", // Example results redacted for brevity
          "lexical": "hello thank you for calling contoso who am i speaking with today hi **** uh are you calling because you need health insurance" // Example results redacted for brevity
        },
        {
          "channel": "1",
          "display": "Hi, my name is *****. I'm trying to enroll myself with Contoso. Yes. Yeah, I'm calling to sign up for insurance.", // Example results redacted for brevity
          "itn": "hi my name is ***** i'm trying to enroll myself with contoso yes yeah i'm calling to sign up for insurance", // Example results redacted for brevity
          "lexical": "hi my name is ***** i'm trying to enroll myself with contoso yes yeah i'm calling to sign up for insurance" // Example results redacted for brevity
        }
      ],
      "conversations": [
        {
          "id": "conversation1",
          "conversationItems": [
            {
              "id": "item1",
              "text": "Customer wants to sign up for insurance"
            },
            {
              "id": "item2",
              "text": "Customer was advised that customer would be contacted by the insurance company"
            }
          ]
        }
      ]
    }
  }
}
```

```
{
    "id": "0",
    "redactedContent": {
        "itn": "hello thank you for calling contoso who am i speaking with today",
        "lexical": "hello thank you for calling contoso who am i speaking with today",
        "text": "Hello, thank you for calling Contoso. Who am I speaking with today?"
    },
    "entities": [],
    "channel": "0",
    "offset": "PT0.77S"
},
{
    "id": "1",
    "redactedContent": {
        "itn": "hi my name is ***** i'm trying to enroll myself with contoso",
        "lexical": "hi my name is ***** i'm trying to enroll myself with contoso",
        "text": "Hi, my name is *****. I'm trying to enroll myself with Contoso."
    },
    "entities": [
        {
            "text": "Mary Rondo",
            "category": "Name",
            "offset": 15,
            "length": 10,
            "confidenceScore": 0.97
        }
    ],
    "channel": "1",
    "offset": "PT4.55S"
},
{
    "id": "2",
    "redactedContent": {
        "itn": "hi *** uh are you calling because you need health insurance",
        "lexical": "hi *** uh are you calling because you need health insurance",
        "text": "Hi, ***. Uh, are you calling because you need health insurance?"
    },
    "entities": [
        {
            "text": "Mary",
            "category": "Name",
            "offset": 4,
            "length": 4,
            "confidenceScore": 0.93
        }
    ],
}
```

```
        "channel": "0",
        "offset": "PT9.55S"
    },
    {
        "id": "3",
        "redactedContent": {
            "itn": "yes yeah i'm calling to sign up for insurance",
            "lexical": "yes yeah i'm calling to sign up for insurance",
            "text": "Yes. Yeah, I'm calling to sign up for insurance."
        },
        "entities": [],
        "channel": "1",
        "offset": "PT13.09S"
    },
    // Example results redacted for brevity
],
    "warnings": []
}
]
}
}
```

Usage and arguments

Usage: `call-center -- [...]`

ⓘ Important

You can use a [Cognitive Services multi-service](#) resource or separate [Language](#) and [Speech](#) resources. In either case, the `--languageKey` and `--languageEndpoint` values must correspond to a resource that's in one of the regions supported by the [conversation summarization API](#): `eastus`, `northeurope`, and `uksouth`.

Connection options include:

- `--speechKey KEY`: Your [Cognitive Services](#) or [Speech](#) resource key. Required for audio transcriptions with the `--input` from URL option.
- `--speechRegion REGION`: Your [Cognitive Services](#) or [Speech](#) resource region. Required for audio transcriptions with the `--input` from URL option. Examples:
`eastus`, `northeurope`
- `--languageKey KEY`: Your [Cognitive Services](#) or [Language](#) resource key. Required.

- `--languageEndpoint` `ENDPOINT`: Your [Cognitive Services](#) or [Language](#) resource endpoint. Required. Example:
`https://YourResourceName.cognitiveservices.azure.com`

Input options include:

- `--input` `URL`: Input audio from URL. You must set either the `--input` or `--jsonInput` option.
- `--jsonInput` `FILE`: Input an existing batch transcription JSON result from FILE. With this option, you only need a Language resource to process a transcription that you already have. With this option, you don't need an audio file or a Speech resource. Overrides `--input`. You must set either the `--input` or `--jsonInput` option.
- `--stereo`: Indicates that the audio via ``input URL`` should be in stereo format. If stereo isn't specified, then mono 16khz 16 bit PCM wav files are assumed. Diarization of mono files is used to separate multiple speakers. Diarization of stereo files isn't supported, since 2-channel stereo files should already have one speaker per channel.
- `--certificate`: The PEM certificate file. Required for C++.

Language options include:

- `--language` `LANGUAGE`: The language to use for sentiment analysis and conversation analysis. This value should be a two-letter ISO 639-1 code. The default value is `en`.
- `--locale` `LOCALE`: The locale to use for batch transcription of audio. The default value is `en-US`.

Output options include:

- `--help`: Show the usage help and stop
- `--output` `FILE`: Output the transcription, sentiment, conversation PII, and conversation summaries in JSON format to a text file. For more information, see [output examples](#).

Clean up resources

You can use the [Azure portal](#) or [Azure Command Line Interface \(CLI\)](#) to remove the Cognitive Services resource you created.

Next steps

[Try the Ingestion Client](#)

Ingestion Client with Azure Cognitive Services

Article • 01/11/2023 • 2 minutes to read

The Ingestion Client is a tool released by Microsoft on GitHub that helps you quickly deploy a call center transcription solution to Azure with a no-code approach.

💡 Tip

You can use the tool and resulting solution in production to process a high volume of audio.

Ingestion Client uses the [Azure Cognitive Service for Language](#), [Azure Cognitive Service for Speech](#), [Azure storage ↗](#), and [Azure Functions ↗](#).

Get started with the Ingestion Client

An Azure Account and an Azure Cognitive Services resource are needed to run the Ingestion Client.

- Azure subscription - [Create one for free ↗](#)
- [Create a Cognitive Services resource ↗](#) in the Azure portal.
- Get the resource key and region. After your Cognitive Services resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

See the [Getting Started Guide for the Ingestion Client ↗](#) on GitHub to learn how to setup and use the tool.

Ingestion Client Features

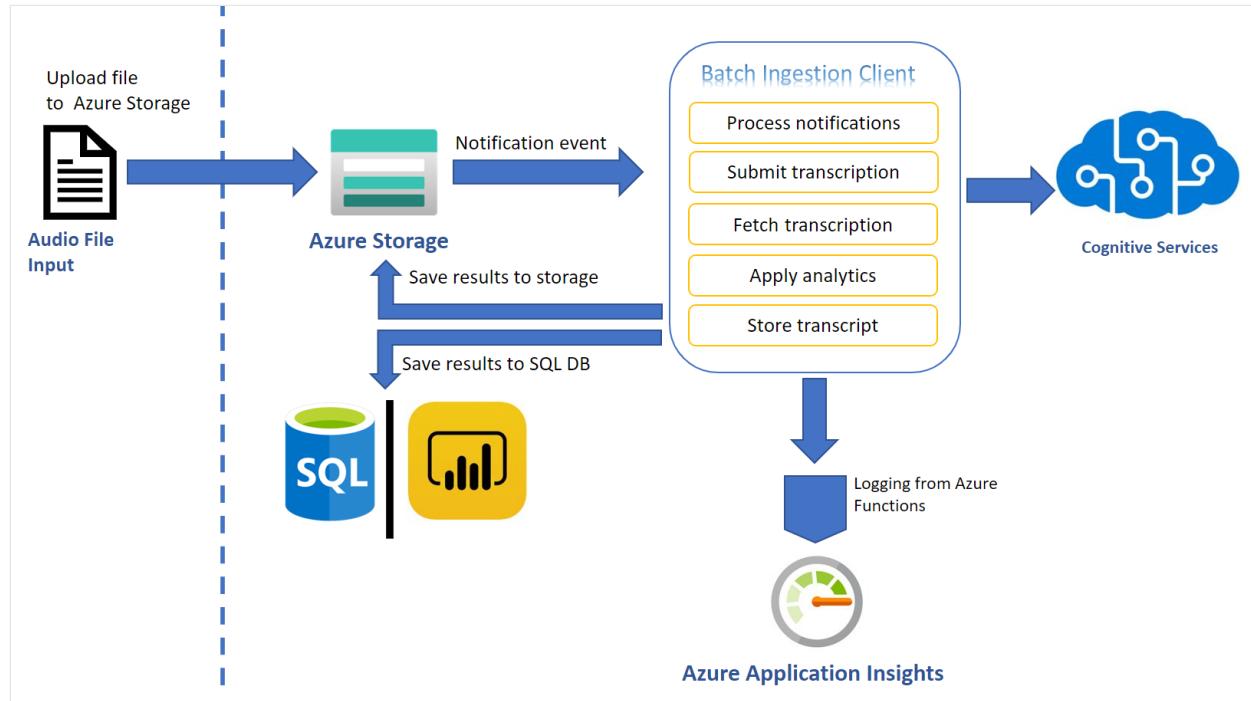
The Ingestion Client works by connecting a dedicated [Azure storage ↗](#) account to custom [Azure Functions ↗](#) in a serverless fashion to pass transcription requests to the service. The transcribed audio files land in the dedicated [Azure Storage container ↗](#).

ⓘ Important

Pricing varies depending on the mode of operation (batch vs real time) as well as the Azure Function SKU selected. By default the tool will create a Premium Azure

Function SKU to handle large volume. Visit the [Pricing](#) page for more information.

Internally, the tool uses Speech and Language services, and follows best practices to handle scale-up, retries and failover. The following schematic describes the resources and connections.



The following Speech service feature is used by the Ingestion Client:

- **Batch speech-to-text:** Transcribe large amounts of audio files asynchronously including speaker diarization and is typically used in post-call analytics scenarios. Diarization is the process of recognizing and separating speakers in mono channel audio data.

Here are some Language service features that are used by the Ingestion Client:

- **Personally Identifiable Information (PII) extraction and redaction:** Identify, categorize, and redact sensitive information in conversation transcription.
- **Sentiment analysis and opinion mining:** Analyze transcriptions and associate positive, neutral, or negative sentiment at the utterance and conversation-level.

Besides Cognitive Services, these Azure products are used to complete the solution:

- **Azure storage:** For storing telephony data and the transcripts that are 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](#): For creating the shared access signature (SAS) URI for each recording, and triggering the HTTP POST request to start a transcription. Additionally, you use Azure Functions to create requests to retrieve and delete transcriptions by using the Batch Transcription API.

Tool customization

The tool is built to show customers results quickly. You can customize the tool to your preferred SKUs and setup. The SKUs can be edited from the [Azure portal](#) and [the code itself is available on GitHub](#).

Note

We suggest creating the resources in the same dedicated resource group to understand and track costs more easily.

Next steps

- [Learn more about Cognitive Services features for call center](#)
- [Explore the Language service features](#)
- [Explore the Speech service features](#)

Additional resources

Documentation

[Post-call transcription and analytics quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you perform sentiment analysis and conversation summarization of call center transcriptions.

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Intent recognition overview - Speech service - Azure Cognitive Services](#)

Intent recognition allows you to recognize user objectives you have pre-defined. This article is an overview of the benefits and capabilities of the intent recognition service.

[Guidance for integration and responsible use with Azure Cognitive Service for](#)

[language - Azure Cognitive Services](#)

Guidance for how to deploy Azure Cognitive Service for language features responsibly, based on the knowledge and understanding from the team that created this product.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[Intent recognition quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you recognize intents from audio data with the Speech service and LUIS.

[Text Analysis Runtime - REST API \(Azure Cognitive Services - Language\)](#)

Learn more about [Cognitive Services - Language Text Analysis Runtime Operations]. How to [Analyze Text,Cancel Job,Job Status,Submit Job].

[Model lifecycle of Custom Speech - Speech service - Azure Cognitive Services](#)

Custom Speech provides base models for training and lets you create custom models from your data. This article describes the timelines for models and for endpoints that use these models.

[Show 5 more](#)

Telephony Integration

Article • 10/18/2022 • 2 minutes to read

To support real-time scenarios, like Virtual Agent and Agent Assist in Call Centers, an integration with the Call Centers telephony system is required.

Typically, the integration with Microsoft Speech Services is handled by a telephony client connected to the customers SIP/RTP processor, for example, to a Session Border Controller (SBC).

Usually the telephony client handles the incoming audio stream from the SIP/RTP processor, the conversion to PCM and connects the streams using continuous recognition. It also triages the processing of the results, for example, analysis of speech transcripts for Agent Assist or connect with a dialog processing engine (for example, Azure Botframework or Power Virtual Agent) for Virtual Agent.

For easier integration the Speech Service also supports "ALAW in WAV container" and "MULAW in WAV container" for audio streaming.

To build this integration we recommend using the [Speech SDK](#).

💡 Tip

For guidance on reducing Text to Speech latency check out the [How to lower speech synthesis latency](#) guide.

In addition, consider implementing a Text to Speech cache to store all synthesized audio and playback from the cache in case a string has previously been synthesized.

Next steps

- [Learn about Speech SDK](#)
- [How to lower speech synthesis latency](#)

Game development with Azure Cognitive Service for Speech

Article • 01/25/2023 • 3 minutes to read

Azure Cognitive Services for Speech can be used to improve various gaming scenarios, both in- and out-of-game.

Here are a few Speech features to consider for flexible and interactive game experiences:

- Bring everyone into the conversation by synthesizing audio from text. Or by displaying text from audio.
- Make the game more accessible for players who are unable to read text in a particular language, including young players who haven't learned to read and write. Players can listen to storylines and instructions in their preferred language.
- Create game avatars and non-playable characters (NPC) that can initiate or participate in a conversation in-game.
- Prebuilt neural voice can provide highly natural out-of-box voices with leading voice variety in terms of a large portfolio of languages and voices.
- Custom neural voice for creating a voice that stays on-brand with consistent quality and speaking style. You can add emotions, accents, nuances, laughter, and other para linguistic sounds and expressions.
- Use game dialogue prototyping to shorten the amount of time and money spent in product to get the game to market sooner. You can rapidly swap lines of dialog and listen to variations in real-time to iterate the game content.

You can use the [Speech SDK](#) or [Speech CLI](#) for real-time low latency speech-to-text, text-to-speech, language identification, and speech translation. You can also use the [Batch transcription API](#) to transcribe pre-recorded speech to text. To synthesize a large volume of text input (long and short) to speech, use the [Batch synthesis API](#).

For information about locale and regional availability, see [Language and voice support](#) and [Region support](#).

Text-to-speech

Help bring everyone into the conversation by converting text messages to audio using [Text-to-Speech](#) for scenarios, such as game dialogue prototyping, greater accessibility, or non-playable character (NPC) voices. Text-to-Speech includes [prebuilt neural voice](#) and [custom neural voice](#) features. Prebuilt neural voice can provide highly natural out-

of-box voices with leading voice variety in terms of a large portfolio of languages and voices. Custom neural voice is an easy-to-use self-service for creating a highly natural custom voice.

When enabling this functionality in your game, keep in mind the following benefits:

- Voices and languages supported - A large portfolio of [locales and voices](#) are supported. You can also [specify multiple languages](#) for Text-to-Speech output. For [custom neural voice](#), you can [choose to create](#) different languages from single language training data.
- Emotional styles supported - [Emotional tones](#), such as cheerful, angry, sad, excited, hopeful, friendly, unfriendly, terrified, shouting, and whispering. You can [adjust the speaking style](#), style degree, and role at the sentence level.
- Visemes supported - You can use visemes during real-time synthesizing to control the movement of 2D and 3D avatar models, so that the mouth movements are perfectly matched to synthetic speech. For more information, see [Get facial position with viseme](#).
- Fine-tuning Text-to-Speech output with Speech Synthesis Markup Language (SSML) - With SSML, you can customize Text-to-Speech outputs, with richer voice tuning supports. For more information, see [Speech Synthesis Markup Language \(SSML\) overview](#).
- Audio outputs - Each prebuilt neural voice model is available at 24 kHz and high-fidelity 48 kHz. If you select 48-kHz output format, the high-fidelity voice model with 48 kHz will be invoked accordingly. The sample rates other than 24 kHz and 48 kHz can be obtained through upsampling or downsampling when synthesizing. For example, 44.1 kHz is downsampled from 48 kHz. Each audio format incorporates a bitrate and encoding type. For more information, see the [supported audio formats](#). For more information on 48-kHz high-quality voices, see [this introduction blog ↗](#).

For an example, see the [Text-to-speech quickstart](#).

Speech-to-text

You can use [speech-to-text](#) to display text from the spoken audio in your game. For an example, see the [Speech-to-text quickstart](#).

Language identification

With [language identification](#), you can detect the language of the chat string submitted by the player.

Speech translation

It's not unusual that players in the same game session natively speak different languages and may appreciate receiving both the original message and its translation. You can use [speech translation](#) to translate text between languages so players across the world can communicate with each other in their native language.

For an example, see the [Speech translation quickstart](#).

ⓘ Note

Besides the Speech service, you can also use the [Translator service](#). To execute text translation between supported source and target languages in real time see [Text translation](#).

Next steps

- [Azure gaming documentation](#)
- [Text-to-speech quickstart](#)
- [Speech-to-text quickstart](#)
- [Speech translation quickstart](#)

Additional resources

Pronunciation assessment in Speech Studio

Article • 02/06/2023 • 6 minutes to read

Pronunciation assessment uses the Speech-to-Text capability to provide subjective and objective feedback for language learners. Practicing pronunciation and getting timely feedback are essential for improving language skills. Assessments driven by experienced teachers can take a lot of time and effort and makes a high-quality assessment expensive for learners. Pronunciation assessment can help make the language assessment more engaging and accessible to learners of all backgrounds.

Pronunciation assessment provides various assessment results in different granularities, from individual phonemes to the entire text input.

- At the full-text level, pronunciation assessment offers additional Fluency and Completeness scores: Fluency indicates how closely the speech matches a native speaker's use of silent breaks between words, and Completeness indicates how many words are pronounced in the speech to the reference text input. An overall score aggregated from Accuracy, Fluency and Completeness is then given to indicate the overall pronunciation quality of the given speech.
- At the word-level, pronunciation assessment can automatically detect miscues and provide accuracy score simultaneously, which provides more detailed information on omission, repetition, insertions, and mispronunciation in the given speech.
- Syllable-level accuracy scores are currently only available via the [JSON file](#) or [Speech SDK](#).
- At the phoneme level, pronunciation assessment provides accuracy scores of each phoneme, helping learners to better understand the pronunciation details of their speech.

This article describes how to use the pronunciation assessment tool through the [Speech Studio](#). You can get immediate feedback on the accuracy and fluency of your speech without writing any code. For information about how to integrate pronunciation assessment in your speech applications, see [How to use pronunciation assessment](#).

ⓘ Note

Usage of pronunciation assessment costs the same as standard Speech to Text pay-as-you-go pricing. Pronunciation assessment doesn't yet support commitment tier pricing.

For information about availability of pronunciation assessment, see [supported languages and available regions](#).

Try out pronunciation assessment

You can explore and try out pronunciation assessment even without signing in.

💡 Tip

To assess more than 5 seconds of speech with your own script, sign in with an [Azure account](#) and use your [Speech resource](#).

Follow these steps to assess your pronunciation of the reference text:

1. Go to **Pronunciation Assessment** in the [Speech Studio](#).

The screenshot shows the 'Speech-to-text' section of the Speech Studio interface. It features three main cards: 'Real-time Speech-to-text', 'Custom Speech', and 'Pronunciation Assessment'. Each card has an icon, a brief description, and a 'Try out' button.

Category	Icon	Description	Action
Real-time Speech-to-text	Microphone and document icon	Quickly test live transcription capabilities on your own audio without writing any code.	Try out Real-time Speech-to-text
Custom Speech	Microphone, document, and gear icon	Add your own data and adapt to specific speaking styles, vocabulary, and more with a customized speech-to-text model.	Start a Custom Speech project
Pronunciation Assessment	Microphone and document icon	Get instant feedback on pronunciation accuracy and fluency by reading a script aloud.	Try out Pronunciation Assessment

2. Choose a supported language that you want to evaluate the pronunciation.

Choose a Pronunciation Assessment language

English (United States)



Provide audio of the script being read aloud, either by recording here or by adding a prerecorded audio file.

[Sample 1](#) [Sample 2](#) [Sample 3](#) [Sample 4](#) [Enter your own script](#)

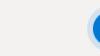
Today was a beautiful day. We had a great time taking a long walk outside in the morning. The countryside was in full bloom, yet the air was crisp and cold. Towards the end of the day, clouds came in, forecasting much needed rain.

3. Choose from the provisioned text samples, or under the **Enter your own script** label, enter your own reference text.

When reading the text, you should be close to microphone to make sure the recorded voice isn't too low.

[Sample 1](#) [Sample 2](#) [Sample 3](#) [Sample 4](#) [Enter your own script](#)

Today was a beautiful day. We had a great time taking a long walk outside in the morning. The countryside was in full bloom, yet the air was crisp and cold. Towards the end of the day, clouds came in, forecasting much needed rain.



Record audio with a microphone



Drag and drop .WAV audio file(s) (16kHz or 8kHz, 16-bit, and mono PCM) here or
[Browse files...](#)

Otherwise you can upload recorded audio for pronunciation assessment. Once successfully uploaded, the audio will be automatically evaluated by the system, as shown in the following screenshot.

[Sample 1](#) [Sample 2](#) [Sample 3](#) [Sample 4](#) [Enter your own script](#)

Today was a beautiful day. We had a great time taking a long walk outside in the morning. The countryside was in full bloom, yet the air was crisp and cold. Towards the end of the day, clouds came in, forecasting much needed rain.



Record audio with a microphone



Drag and drop .WAV audio file(s) (16kHz or 8kHz, 16-bit, and mono PCM) here or
[Browse files...](#)

e8cf9490-803a-11ed-b02e-07c7a6df2413.wav

Pronunciation assessment results

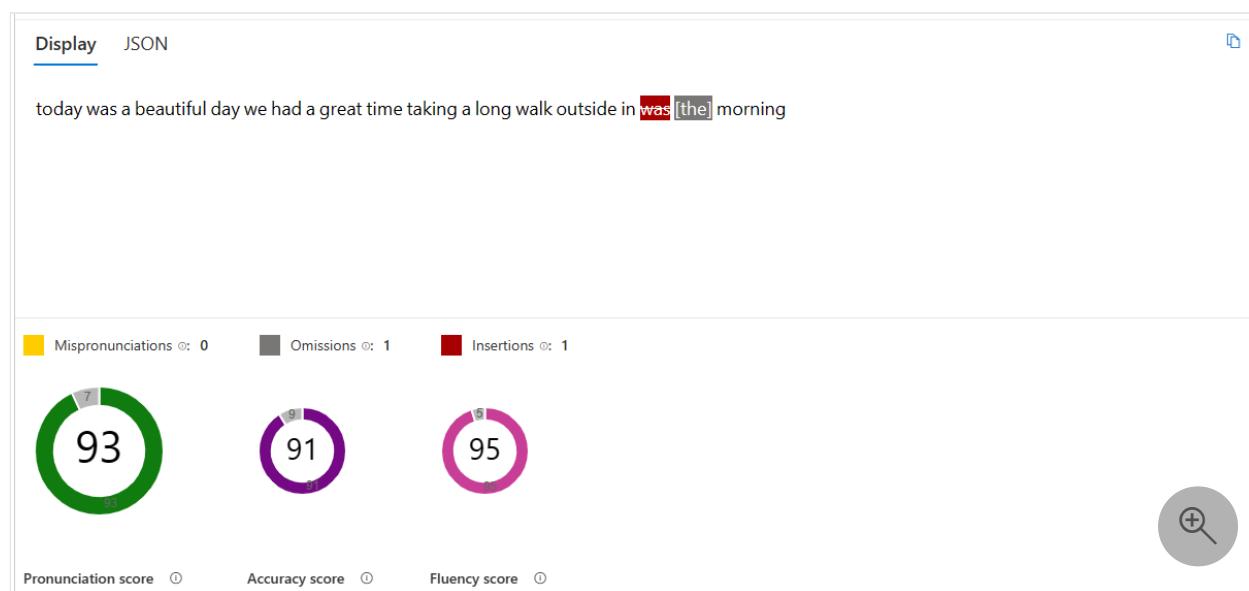
Once you've recorded the reference text or uploaded the recorded audio, the **Assessment result** will be output. The result includes your spoken audio and the feedback on the accuracy and fluency of spoken audio, by comparing a machine generated transcript of the input audio with the reference text. You can listen to your spoken audio, and download it if necessary.

You can also check the pronunciation assessment result in JSON. The word-level, syllable-level, and phoneme-level accuracy scores are included in the JSON file.

Overall scores

Pronunciation Assessment evaluates three aspects of pronunciation: accuracy, fluency, and completeness. At the bottom of **Assessment result**, you can see **Pronunciation score**, **Accuracy score**, **Fluency score**, and **Completeness score**. The **Accuracy score** and the **Fluency score** will vary over time throughout the recording process. The **Completeness score** is only calculated at the end of the evaluation. The **Pronunciation score** is overall score indicating the pronunciation quality of the given speech. During recording, the **Pronunciation score** is aggregated from **Accuracy score** and **Fluency score** with weight. Once completing recording, this overall score is aggregated from **Accuracy score**, **Fluency score**, and **Completeness score** with weight.

During recording



Completing recording

Display JSON

today was a beautiful day we had a great time taking a long walk outside in **was** [the] morning the countryside was in full bloom yet the air was crisp and **cold** towards the end of the day clouds came in forecasting much needed rain

Yellow Mispronunciations 0 | Grey Omissions 1 | Red Insertions 2



Pronunciation score ⓘ Accuracy score ⓘ Fluency score ⓘ Completeness score ⓘ



Scores within words

Display

The complete transcription is shown in the **Display** window. If a word is omitted, inserted, or mispronounced compared to the reference text, the word will be highlighted according to the error type. While hovering over each word, you can see accuracy scores for the whole word or specific phonemes.

countryside : 30
k ah n t r iy s ay d
0 0 11 16 27 15 26 56 100

a **long** long walk **[outside]** in the morning the **countryside** was in full bloom
needed rain



Next steps

- Use [pronunciation assessment with the Speech SDK](#)
- Read the blog about [use cases ↗](#)

Additional resources

Documentation

[azure.cognitiveservices.speech.SpeechConfig class](#)

Class that defines configurations for speech / intent recognition and speech synthesis. The configuration can be initialized in different ways: from subscription: pass a subscription key and a region from endpoint: pass an endpoint. Subscription key or authorization token are optional. from...

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[azure.cognitiveservices.speech.Recognizer class](#)

Base class for different recognizers

[Custom Speech overview - Speech service - Azure Cognitive Services](#)

Custom Speech is a set of online tools that allows you to evaluate and improve the Microsoft speech-to-text accuracy for your applications, tools, and products.

[Speech-to-text REST API - Speech service - Azure Cognitive Services](#)

Get reference documentation for Speech-to-text REST API.

[azure.cognitiveservices.speech.SpeechSynthesizer class](#)

A speech synthesizer.

[Show 5 more](#)

Use pronunciation assessment

Article • 02/06/2023 • 8 minutes to read

In this article, you'll learn how to evaluate pronunciation with the Speech-to-Text capability through the Speech SDK. To [get pronunciation assessment results](#), you'll apply the `PronunciationAssessmentConfig` settings to a `SpeechRecognizer` object.

You can get pronunciation assessment scores for:

- Full text
- Words
- Syllable groups
- Phonemes in [SAPI](#) or [IPA ↗](#) format

ⓘ Note

The syllable group, phoneme name, and spoken phoneme of pronunciation assessment are currently only available for the en-US locale.

Usage of pronunciation assessment costs the same as standard Speech to Text pay-as-you-go [pricing ↗](#). Pronunciation assessment doesn't yet support commitment tier pricing.

For information about availability of pronunciation assessment, see [supported languages](#) and [available regions](#).

Configuration parameters

This table lists some of the key configuration parameters for pronunciation assessment.

Parameter	Description
<code>ReferenceText</code>	The text that the pronunciation will be evaluated against.
<code>GradingSystem</code>	The point system for score calibration. The <code>FivePoint</code> system gives a 0-5 floating point score, and <code>HundredMark</code> gives a 0-100 floating point score. Default: <code>FivePoint</code> .

Parameter	Description
Granularity	Determines the lowest level of evaluation granularity. Scores for levels above or equal to the minimal value are returned. Accepted values are <code>Phoneme</code> , which shows the score on the full text, word, syllable, and phoneme level, <code>Syllable</code> , which shows the score on the full text, word, and syllable level, <code>Word</code> , which shows the score on the full text and word level, or <code>FullText</code> , which shows the score on the full text level only. The provided full reference text can be a word, sentence, or paragraph, and it depends on your input reference text. Default: <code>Phoneme</code> .
EnableMiscue	Enables miscue calculation when the pronounced words are compared to the reference text. If this value is <code>True</code> , the <code>ErrorType</code> result value can be set to <code>Omission</code> or <code>Insertion</code> based on the comparison. Accepted values are <code>False</code> and <code>True</code> . Default: <code>False</code> .
ScenarioId	A GUID indicating a customized point system.

You must create a `PronunciationAssessmentConfig` object with the reference text, grading system, and granularity. Enabling miscue and other configuration settings are optional.

C#

```
var pronunciationAssessmentConfig = new PronunciationAssessmentConfig(
    referenceText: "good morning",
    gradingSystem: GradingSystem.HundredMark,
    granularity: Granularity.Phoneme,
    enableMiscue: true);
```

Syllable groups

Pronunciation assessment can provide syllable-level assessment results. Grouping in syllables is more legible and aligned with speaking habits, as a word is typically pronounced syllable by syllable rather than phoneme by phoneme.

The following table compares example phonemes with the corresponding syllables.

Sample word	Phonemes	Syllables
technological	teknələdʒɪkl	tek-nə-la-dʒɪkl
hello	həloʊ	hə-loʊ
luck	lʌk	lʌk

Sample word	Phonemes	Syllables
photosynthesis	fəʊtəsɪnθəsɪs	fou-tə-sin-thə-sis

To request syllable-level results along with phonemes, set the granularity [configuration parameter](#) to `Phoneme`.

Phoneme alphabet format

For `en-us` locale, the phoneme name is provided together with the score, to help identify which phonemes were pronounced accurately or inaccurately. For other locales, you can only get the phoneme score.

The following table compares example SAPI phonemes with the corresponding IPA phonemes.

Sample word	SAPI Phonemes	IPA phonemes
hello	h eh l ow	h ε l oʊ
luck	l ah k	l ʌ k
photosynthesis	f ow t ax s ih n th ax s ih s	f ou təsɪnθəsɪs

To request IPA phonemes, set the phoneme alphabet to `"IPA"`. If you don't specify the alphabet, the phonemes will be in SAPI format by default.

```
C#
pronunciationAssessmentConfig.PhonemeAlphabet = "IPA";
```

Spoken phoneme

With spoken phonemes, you can get confidence scores indicating how likely the spoken phonemes matched the expected phonemes.

For example, when you speak the word "hello", the expected IPA phonemes are "h ε l oʊ". The actual spoken phonemes could be "h ə l oʊ". In the following assessment result, the most likely spoken phoneme was `"ə"` instead of the expected phoneme `"ε"`. The expected phoneme `"ε"` only received a confidence score of 47. Other potential matches received confidence scores of 52, 17, and 2.

JSON

```
{  
    "Phoneme": "\u0259",  
    "PronunciationAssessment": {  
        "AccuracyScore": 47.0,  
        "NBestPhonemes": [  
            {  
                "Phoneme": "\u028a",  
                "Score": 100.0  
            },  
            {  
                "Phoneme": "\u0281",  
                "Score": 52.0  
            },  
            {  
                "Phoneme": "\u0259",  
                "Score": 47.0  
            },  
            {  
                "Phoneme": "h",  
                "Score": 17.0  
            },  
            {  
                "Phoneme": "\u0282",  
                "Score": 2.0  
            }  
        ]  
    },  
    "Offset": 11100000,  
    "Duration": 500000  
},
```

To indicate whether, and how many potential spoken phonemes to get confidence scores for, set the `NBestPhonemeCount` parameter to an integer value such as 5.

C#

```
pronunciationAssessmentConfig.NBestPhonemeCount = 5;
```

Get pronunciation assessment results

When speech is recognized, you can request the pronunciation assessment results as SDK objects or a JSON string.

C#

```
using (var speechRecognizer = new SpeechRecognizer(  
    speechConfig,  
    audioConfig))  
{
```

```

pronunciationAssessmentConfig.ApplyTo(speechRecognizer);
var speechRecognitionResult = await
speechRecognizer.RecognizeOnceAsync();

// The pronunciation assessment result as a Speech SDK object
var pronunciationAssessmentResult =
    PronunciationAssessmentResult.FromResult(speechRecognitionResult);

// The pronunciation assessment result as a JSON string
var pronunciationAssessmentResultJson =
speechRecognitionResult.Properties.GetProperty(PropertyId.SpeechServiceRespo
nse_JsonResult);
}

```

Result parameters

This table lists some of the key pronunciation assessment results.

Parameter	Description
AccuracyScore	Pronunciation accuracy of the speech. Accuracy indicates how closely the phonemes match a native speaker's pronunciation. Syllable, word, and full text accuracy scores are aggregated from phoneme-level accuracy score, and refined with assessment objectives.
FluencyScore	Fluency of the given speech. Fluency indicates how closely the speech matches a native speaker's use of silent breaks between words.
CompletenessScore	Completeness of the speech, calculated by the ratio of pronounced words to the input reference text.
PronScore	Overall score indicating the pronunciation quality of the given speech. PronScore is aggregated from AccuracyScore, FluencyScore, and CompletenessScore with weight.
ErrorType	This value indicates whether a word is omitted, inserted, or mispronounced, compared to the ReferenceText. Possible values are None, Omission, Insertion, and Mispronunciation. The error type can be Mispronunciation when the pronunciation AccuracyScore for a word is below 60.

JSON result example

Pronunciation assessment results for the spoken word "hello" are shown as a JSON string in the following example. Here's what you should know:

- The phoneme [alphabet](#) is IPA.
- The [syllables](#) are returned alongside phonemes for the same word.

- You can use the `Offset` and `Duration` values to align syllables with their corresponding phonemes. For example, the starting offset (11700000) of the second syllable ("lou") aligns with the third phoneme ("l").
- There are five `NBestPhonemes` corresponding to the number of spoken phonemes requested.
- Within `Phonemes`, the most likely `spoken phonemes` was "ə" instead of the expected phoneme "ɛ". The expected phoneme "ɛ" only received a confidence score of 47. Other potential matches received confidence scores of 52, 17, and 2.

JSON

```
{
  "Id": "bbb42ea51bdb46d19a1d685e635fe173",
  "RecognitionStatus": 0,
  "Offset": 7500000,
  "Duration": 13800000,
  "DisplayText": "Hello.",
  "NBest": [
    {
      "Confidence": 0.975003,
      "Lexical": "hello",
      "ITN": "hello",
      "MaskedITN": "hello",
      "Display": "Hello.",
      "PronunciationAssessment": {
        "AccuracyScore": 100,
        "FluencyScore": 100,
        "CompletenessScore": 100,
        "PronScore": 100
      },
      "Words": [
        {
          "Word": "hello",
          "Offset": 7500000,
          "Duration": 13800000,
          "PronunciationAssessment": {
            "AccuracyScore": 99.0,
            "ErrorType": "None"
          },
          "Syllables": [
            {
              "Syllable": "hɛ",
              "PronunciationAssessment": {
                "AccuracyScore": 91.0
              },
              "Offset": 7500000,
              "Duration": 4100000
            },
            {
              "Syllable": "lou",
              "PronunciationAssessment": {
                "AccuracyScore": 47.0
              }
            }
          ]
        }
      ]
    }
  ]
}
```

```
        "AccuracyScore": 100.0
    },
    "Offset": 11700000,
    "Duration": 9600000
}
],
"Phonemes": [
{
    "Phoneme": "h",
    "PronunciationAssessment": {
        "AccuracyScore": 98.0,
        "NBestPhonemes": [
            {
                "Phoneme": "h",
                "Score": 100.0
            },
            {
                "Phoneme": "oo",
                "Score": 52.0
            },
            {
                "Phoneme": "ə",
                "Score": 35.0
            },
            {
                "Phoneme": "k",
                "Score": 23.0
            },
            {
                "Phoneme": "æ",
                "Score": 20.0
            }
        ]
    },
    "Offset": 7500000,
    "Duration": 3500000
},
{
    "Phoneme": "ε",
    "PronunciationAssessment": {
        "AccuracyScore": 47.0,
        "NBestPhonemes": [
            {
                "Phoneme": "ə",
                "Score": 100.0
            },
            {
                "Phoneme": "l",
                "Score": 52.0
            },
            {
                "Phoneme": "ε",
                "Score": 47.0
            }
        ]
    }
}
```

```
        "Phoneme": "h",
        "Score": 17.0
    },
{
    "Phoneme": "æ",
    "Score": 2.0
}
]
},
{
    "Offset": 11100000,
    "Duration": 500000
},
{
    "Phoneme": "l",
    "PronunciationAssessment": {
        "AccuracyScore": 100.0,
        "NBestPhonemes": [
            {
                "Phoneme": "l",
                "Score": 100.0
            },
            {
                "Phoneme": "œ",
                "Score": 46.0
            },
            {
                "Phoneme": "ə",
                "Score": 5.0
            },
            {
                "Phoneme": "ɛ",
                "Score": 3.0
            },
            {
                "Phoneme": "u",
                "Score": 1.0
            }
        ]
    },
    "Offset": 11700000,
    "Duration": 1100000
},
{
    "Phoneme": "œ",
    "PronunciationAssessment": {
        "AccuracyScore": 100.0,
        "NBestPhonemes": [
            {
                "Phoneme": "œ",
                "Score": 100.0
            },
            {
                "Phoneme": "d",
                "Score": 29.0
            }
        ]
    }
}
```

```
        {
            "Phoneme": "t",
            "Score": 24.0
        },
        {
            "Phoneme": "n",
            "Score": 22.0
        },
        {
            "Phoneme": "l",
            "Score": 18.0
        }
    ]
},
{
    "Offset": 12900000,
    "Duration": 8400000
}
]
}
]
}
]
```

Next steps

- Try out [pronunciation assessment in Speech Studio](#)
- Try out the [pronunciation assessment demo ↗](#) and watch the [video tutorial ↗](#) of pronunciation assessment.

Additional resources

Documentation

[How to use pronunciation assessment in Speech Studio - Azure Cognitive Services](#)

The pronunciation assessment tool in Speech Studio gives you feedback on the accuracy and fluency of your speech, no coding required.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[azure.cognitiveservices.speech.Recognizer class](#)

Base class for different recognizers

[azure.cognitiveservices.speech.SpeechConfig class](#)

Class that defines configurations for speech / intent recognition and speech synthesis. The configuration can be initialized in different ways: from subscription: pass a subscription key and a region from endpoint: pass an endpoint. Subscription key or authorization token are optional. from...

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Speech-to-text REST API - Speech service - Azure Cognitive Services](#)

Get reference documentation for Speech-to-text REST API.

[Show 5 more](#)

What is conversation transcription?

Article • 01/11/2023 • 3 minutes to read

Conversation transcription is a [speech-to-text](#) solution that provides real-time or asynchronous transcription of any conversation. This feature, which is currently in preview, combines speech recognition, speaker identification, and sentence attribution to determine who said what, and when, in a conversation.

ⓘ Note

Multi-device conversation access is a preview feature.

Key features

You might find the following features of conversation transcription useful:

- **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 by 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 doesn't put a limit on the number of speakers in the room, it's optimized for 2-10 speakers per session.

Get started

See the real-time conversation transcription [quickstart](#) to get started.

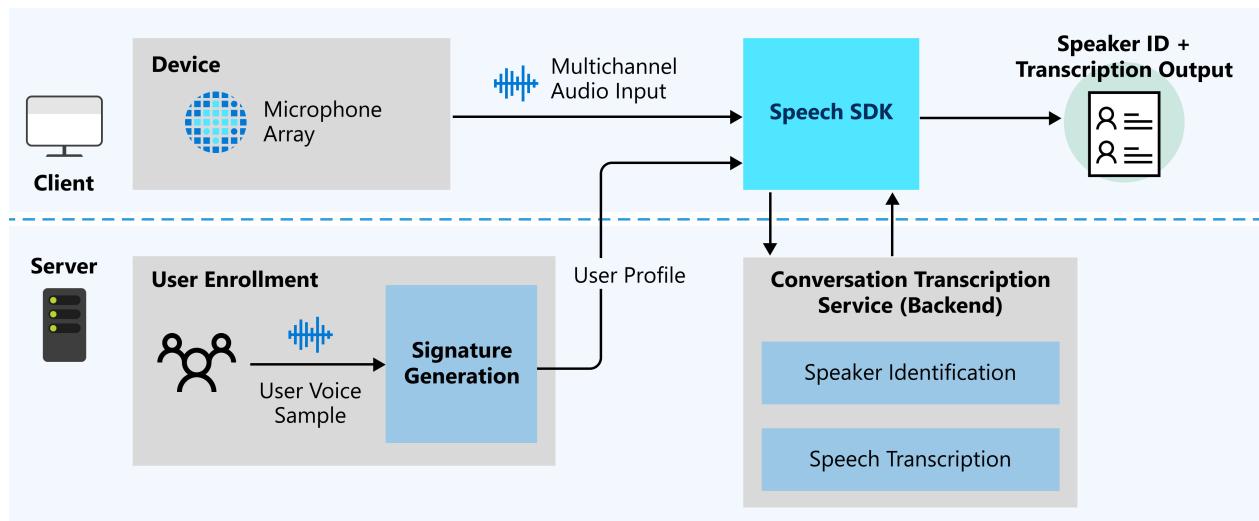
Use cases

To make meetings inclusive for everyone, such as participants who are deaf and hard of hearing, it's 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.

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

The following diagram shows a high-level overview of how the feature works.



Expected inputs

Conversation transcription uses two types of inputs:

- **Multi-channel audio stream:** For specification and design details, see [Microphone array recommendations](#).
- **User voice samples:** Conversation transcription needs user profiles in advance of the conversation for speaker identification. Collect audio recordings from each user, and then send the recordings to the [signature generation service](#) to validate the audio and generate user profiles.

User voice samples for voice signatures are required for speaker identification. Speakers who don't have voice samples are recognized as *unidentified*. Unidentified speakers can still be differentiated when the `DifferentiateGuestSpeakers` property is enabled (see the following example). The transcription output then shows speakers as, for example, *Guest_0* and *Guest_1*, instead of recognizing them as pre-enrolled specific speaker names.

```
C#
```

```
config SetProperty("DifferentiateGuestSpeakers", "true");
```

Real-time vs. asynchronous

The following sections provide more detail about transcription modes you can choose.

Real-time

Audio data is processed live to return the speaker identifier and 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 to participants with hearing loss or deafness is an ideal use case for real-time transcription.

Asynchronous

Audio data is batch processed to return the speaker identifier and transcript. Select this mode if your transcription solution requirement is to provide higher accuracy, without the 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 the speaker identifier and transcript, and, in addition, requests a high-accuracy transcript through asynchronous processing. Select this mode if your application has a need for real-time transcription, and also requires a higher accuracy transcript for use after the conversation or meeting occurred.

Language support

Currently, conversation transcription supports all speech-to-text languages in the following regions: `centralus`, `eastasia`, `eastus`, `westeurope`.

Next steps

[Transcribe conversations in real time](#)

Additional resources

Documentation

[Real-time Conversation Transcription quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, learn how to transcribe meetings and other conversations. You can add, remove, and identify multiple participants by streaming audio to the Speech service.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[Asynchronous Conversation Transcription - Speech service - Azure Cognitive Services](#)

Learn how to use asynchronous Conversation Transcription using the Speech service. Available for Java and C# only.

[Speech-to-text FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions about the speech-to-text service.

[Speaker recognition overview - Speech service - Azure Cognitive Services](#)

Speaker recognition provides algorithms that verify and identify speakers by their unique voice characteristics, by using voice biometry. Speaker recognition is used to answer the question "who is speaking?". This article is an overview of the benefits and capabilities of the speaker recognition...

[azure.cognitiveservices.speech package](#)

Microsoft Speech SDK for Python

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[Show 5 more](#)

Quickstart: Real-time Conversation Transcription

Article • 11/17/2022 • 12 minutes to read

You can transcribe meetings and other conversations with the ability to add, remove, and identify multiple participants by streaming audio to the Speech service. You first create voice signatures for each participant using the REST API, and then use the voice signatures with the Speech SDK to transcribe conversations. See the Conversation Transcription [overview](#) for more information.

Limitations

- Only available in the following subscription regions: `centralus`, `eastasia`, `eastus`, `westeurope`
- Requires a 7-mic circular multi-microphone array. The microphone array should meet [our specification](#).

ⓘ Note

The Speech SDK for C++, Java, Objective-C, and Swift support Conversation Transcription, but we haven't yet included a guide here.

Prerequisites

- ✓ Azure subscription - [Create one for free](#)
- ✓ [Create a Speech resource](#) in the Azure portal.
- ✓ Get the resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about Cognitive Services resources, see [Get the keys for your resource](#).

Set up the environment

The Speech SDK is available as a [NuGet package](#) and implements .NET Standard 2.0. You install the Speech SDK later in this guide, but first check the [platform-specific installation instructions](#) for any more requirements.

Create voice signatures

If you want to enroll user profiles, the first step is to create voice signatures for the conversation participants so that they can be identified as unique speakers. This isn't required if you don't want to use pre-enrolled user profiles to identify specific participants.

The input `.wav` audio file for creating voice signatures must be 16-bit, 16 kHz sample rate, in single channel (mono) format. The recommended length for each audio sample is between 30 seconds and two minutes. An audio sample that is too short will result in reduced accuracy when recognizing the speaker. The `.wav` file should be a sample of one person's voice so that a unique voice profile is created.

The following example shows how to create a voice signature by [using the REST API](#) in C#. You must insert your `subscriptionKey`, `region`, and the path to a sample `.wav` file.

C#

```
using System;
using System.IO;
using System.Net.Http;
using System.Runtime.Serialization;
using System.Threading.Tasks;
using Newtonsoft.Json;

[DataContract]
internal class VoiceSignature
{
    [DataMember]
    public string Status { get; private set; }

    [DataMember]
    public VoiceSignatureData Signature { get; private set; }

    [DataMember]
    public string Transcription { get; private set; }
}

[DataContract]
internal class VoiceSignatureData
{
    internal VoiceSignatureData()
    { }

    internal VoiceSignatureData(int version, string tag, string data)
    {
        this.Version = version;
        this.Tag = tag;
        this.Data = data;
    }
}
```

```

[DataMember]
public int Version { get; private set; }

[DataMember]
public string Tag { get; private set; }

[DataMember]
public string Data { get; private set; }
}

private static async Task<string> GetVoiceSignatureString()
{
    var subscriptionKey = "your-subscription-key";
    var region = "your-region";

    byte[] fileBytes = File.ReadAllBytes("path-to-voice-sample.wav");
    var content = new ByteArrayContent(fileBytes);
    var client = new HttpClient();
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key",
subscriptionKey);
    var response = await client.PostAsync($"https://signature.
{region}.cts.speech.microsoft.com/api/v1/Signature/GenerateVoiceSignatureFro
mByteArray", content);

    var jsonData = await response.Content.ReadAsStringAsync();
    var result = JsonConvert.DeserializeObject<VoiceSignature>(jsonData);
    return JsonConvert.SerializeObject(result.Signature);
}

```

Running the function `GetVoiceSignatureString()` returns a voice signature string in the correct format. Run the function twice so you have two strings to use as input to the variables `voiceSignatureStringUser1` and `voiceSignatureStringUser2` below.

 **Note**

Voice signatures can **only** be created using the REST API.

Transcribe conversations

The following sample code demonstrates how to transcribe conversations in real time for two speakers. It assumes you've already created voice signature strings for each speaker as shown above. Substitute real information for `subscriptionKey`, `region`, and the path `filepath` for the audio you want to transcribe.

If you don't use pre-enrolled user profiles, it will take a few more seconds to complete the first recognition of unknown users as speaker1, speaker2, etc.

ⓘ Note

Make sure the same `subscriptionKey` is used across your application for signature creation, or you will encounter errors.

This sample code does the following:

- Creates an `AudioConfig` from the sample `.wav` file to transcribe.
- Creates a `Conversation` using `CreateConversationAsync()`.
- Creates a `ConversationTranscriber` using the constructor, and subscribes to the necessary events.
- Adds participants to the conversation. The strings `voiceSignatureStringUser1` and `voiceSignatureStringUser2` should come as output from the steps above from the function `GetVoiceSignatureString()`.
- Joins the conversation and begins transcription.
- If you want to differentiate speakers without providing voice samples, enable the `DifferentiateGuestSpeakers` feature as in [Conversation Transcription Overview](#).

ⓘ Note

`AudioStreamReader` is a helper class you can get on [GitHub](#).

Call the function `TranscribeConversationsAsync()` to start conversation transcription.

C#

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Transcription;

class transcribe_conversation
{
    // all your other code

    public static async Task TranscribeConversationsAsync(string
        voiceSignatureStringUser1, string voiceSignatureStringUser2)
    {
        var subscriptionKey = "your-subscription-key";
        var region = "your-region";
        var filepath = "audio-file-to-transcribe.wav";

        var config = SpeechConfig.FromSubscription(subscriptionKey, region);
    }
}
```

```

config SetProperty("ConversationTranscriptionInRoomAndOnline", "true");

// en-us by default. Adding this code to specify other languages, like zh-cn.
// config.SpeechRecognitionLanguage = "zh-cn";
var stopRecognition = new TaskCompletionSource<int>();

using (var audioInput = AudioConfig.FromWavFileInput(filepath))
{
    var meetingID = Guid.NewGuid().ToString();
    using (var conversation = await
Conversation.CreateConversationAsync(config, meetingID))
    {
        // create a conversation transcriber using audio stream input
        using (var conversationTranscriber = new
ConversationTranscriber(audioInput))
        {
            conversationTranscriber.Transcribing += (s, e) =>
            {
                Console.WriteLine($"TRANSCRIBING: Text={e.Result.Text}
SpeakerId={e.Result.UserId}");
            };

            conversationTranscriber.Transcribed += (s, e) =>
            {
                if (e.Result.Reason == ResultReason.RecognizedSpeech)
                {
                    Console.WriteLine($"TRANSCRIBED: Text=
{e.Result.Text} SpeakerId={e.Result.UserId}");
                }
                else if (e.Result.Reason == ResultReason.NoMatch)
                {
                    Console.WriteLine($"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 set the speech
resource key and region values?");
                stopRecognition.TrySetResult(0);
            }
        };

        conversationTranscriber.SessionStarted += (s, e) =>
        {

```

```
        Console.WriteLine($"\\nSession started event. SessionId={e.SessionId}");  
    };  
  
    conversationTranscriber.SessionStopped += (s, e) =>  
    {  
        Console.WriteLine($"\\nSession stopped event. SessionId={e.SessionId}");  
        Console.WriteLine("\\nStop recognition.");  
        stopRecognition.TrySetResult(0);  
    };  
  
    // Add participants to the conversation.  
    var speaker1 = Participant.From("User1", "en-US",  
voiceSignatureStringUser1);  
    var speaker2 = Participant.From("User2", "en-US",  
voiceSignatureStringUser2);  
    await conversation.AddParticipantAsync(speaker1);  
    await conversation.AddParticipantAsync(speaker2);  
  
    // Join to the conversation and start transcribing  
    await  
conversationTranscriber.JoinConversationAsync(conversation);  
    await  
conversationTranscriber.StartTranscribingAsync().ConfigureAwait(false);  
  
    // waits for completion, then stop transcription  
    Task.WaitAny(new[] { stopRecognition.Task });  
    await  
conversationTranscriber.StopTranscribingAsync().ConfigureAwait(false);  
    }  
}
```

Next steps

Asynchronous Conversation Transcription

Asynchronous Conversation Transcription

Article • 04/22/2022 • 6 minutes to read

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

The first step for asynchronous transcription is to send the audio to the Conversation Transcription Service using the Speech SDK.

This example code shows how to create a `ConversationTranscriber` for asynchronous-only mode. In order to stream audio to the transcriber, you add audio streaming code derived from [Transcribe conversations in real time with the Speech SDK](#).

C#

```
async Task CompleteContinuousRecognition(ConversationTranscriber recognizer,
    string conversationId)
{
    var finishedTaskCompletionSource = new TaskCompletionSource<int>();
```

```

recognizer.SessionStopped += (s, e) =>
{
    finishedTaskCompletionSource.TrySetResult(0);
};

string canceled = string.Empty;

recognizer.Canceled += (s, e) => {
    canceled = e.ErrorDetails;
    if (e.Reason == CancellationReason.Error)
    {
        finishedTaskCompletionSource.TrySetResult(0);
    }
};

await recognizer.StartTranscribingAsync().ConfigureAwait(false);

// Waits for completion.
// Use Task.WaitAny to keep the task rooted.
Task.WaitAny(new[] { finishedTaskCompletionSource.Task });

await recognizer.StopTranscribingAsync().ConfigureAwait(false);
}

async Task<List<string>> GetRecognizerResult(ConversationTranscriber
recognizer, string conversationId)
{
    List<string> recognizedText = new List<string>();
    recognizer.Transcribed += (s, e) =>
    {
        if (e.Result.Text.Length > 0)
        {
            recognizedText.Add(e.Result.Text);
        }
    };
}

await CompleteContinuousRecognition(recognizer, conversationId);

recognizer.Dispose();
return recognizedText;
}

async Task UploadAudio()
{
    // 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 SetProperty("transcriptionMode", "async",
ServicePropertyChannel.UriQueryParameter);

    // Alternatively: set the property for real-time plus asynchronous
}

```

```

transcription
    // speechConfig.setServiceProperty("transcriptionMode",
    "RealTimeAndAsync", ServicePropertyChannel.UriQueryParameter);

    // 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(16000, 16, 8);
    // Create an input stream
    AudioInputStream audioStream =
AudioInputStream.CreatePullStream(wavfilePullStreamCallback,
audioStreamFormat);

    // Ensure the conversationId for a new conversation is a truly unique
GUID
    String conversationId = Guid.NewGuid().ToString();

    // Create a Conversation
    using (var conversation = await
Conversation.CreateConversationAsync(speechConfig, conversationId))
{
    using (var conversationTranscriber = new
ConversationTranscriber(AudioConfig.FromStreamInput(audioStream)))
    {
        await
conversationTranscriber.JoinConversationAsync(conversation);
        // Helper function to get the real time transcription results
        var result = await GetRecognizerResult(conversationTranscriber,
conversationId);
    }
}
}

```

If you want real-time *plus* asynchronous, comment and uncomment the appropriate lines of code as follows:

C#

```

// Set the property for asynchronous transcription
// speechConfig.SetServiceProperty("transcriptionMode", "async",
ServicePropertyChannel.UriQueryParameter);

// Alternatively: set the property for real-time plus asynchronous
transcription
speechConfig.SetServiceProperty("transcriptionMode", "RealTimeAndAsync",
ServicePropertyChannel.UriQueryParameter);

```

Get transcription results

Install `Microsoft.CognitiveServices.Speech.Remoteconversation` version 1.13.0 or above via NuGet.

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. Create an object of `RemoteConversationTranscriptionOperation` to get a long running [Operation](#) object. You can check the status of the operation or wait for it to complete.

C#

```
// Create the speech config
SpeechConfig config = SpeechConfig.FromSubscription("YourSpeechKey",
    "YourSpeechRegion");
// Create the speech client
RemoteConversationTranscriptionClient client = new
    RemoteConversationTranscriptionClient(config);
// Create the remote operation
RemoteConversationTranscriptionOperation operation =
    new
        RemoteConversationTranscriptionOperation(conversationId, client);

// Wait for operation to finish
await operation.WaitForCompletionAsync(TimeSpan.FromSeconds(10),
    CancellationToken.None);
// Get the result of the long running operation
var val = operation.Value.ConversationTranscriptionResults;
// Print the fields from the results
foreach (var item in val)
{
    Console.WriteLine($"{item.Text}, {item.ResultId}, {item.Reason},
    {item.UserId}, {item.OffsetInTicks}, {item.Duration}");
    Console.WriteLine(${item.Properties.GetProperty(PropertyId.SpeechServiceResponse_JsonResult)})")
;
}
Console.WriteLine("Operation completed");
```

Next steps

[Explore our samples on GitHub](#)

What is Multi-device Conversation?

Article • 08/25/2022 • 4 minutes to read

Multi-device conversation makes it easy to create a speech or text conversation between multiple clients and coordinate the messages sent between them.

ⓘ Note

Multi-device conversation access is a preview feature.

With multi-device conversation, you can:

- Connect multiple clients into the same conversation and manage the sending and receiving of messages between them.
- Easily transcribe audio from each client and send the transcription to the others, with optional translation.
- Easily send text messages between clients, with optional translation.

You can build a feature or solution that works across an array of devices. Each device can independently send messages (either transcriptions of audio or instant messages) to all other devices.

Whereas [Conversation Transcription](#) works on a single device with a multichannel microphone array, Multi-device Conversation is suited for scenarios with multiple devices, each with a single microphone.

ⓘ Important

Multi-device conversation does not support sending audio files between clients: only the transcription and/or translation.

Key features

- **Real-time transcription:** Everyone will receive a transcript of the conversation, so they can follow along the text in real-time or save it for later.
- **Real-time translation:** With more than 70 [supported languages](#) for text translation, users can translate the conversation to their preferred languages.
- **Readable transcripts:** The transcription and translation are easy to follow, with punctuation and sentence breaks.

- **Voice or text input:** Each user can speak or type on their own device, depending on the language support capabilities enabled for the participant's chosen language. Please refer to [Language support](#).
- **Message relay:** The multi-device conversation service will distribute messages sent by one client to all the others, in the languages of their choice.
- **Message identification:** Every message that users receive in the conversation will be tagged with the nickname of the user who sent it.

Use cases

Lightweight conversations

Creating and joining a conversation is easy. One user will act as the 'host' and create a conversation, which generates a random five letter conversation code and a QR code. All other users can join the conversation by typing in the conversation code or scanning the QR code.

Since users join via the conversation code and aren't required to share contact information, it is easy to create quick, on-the-spot conversations.

Inclusive meetings

Real-time transcription and translation can help make conversations accessible for people who speak different languages and/or are deaf or hard of hearing. Each person can also actively participate in the conversation, by speaking their preferred language or sending instant messages.

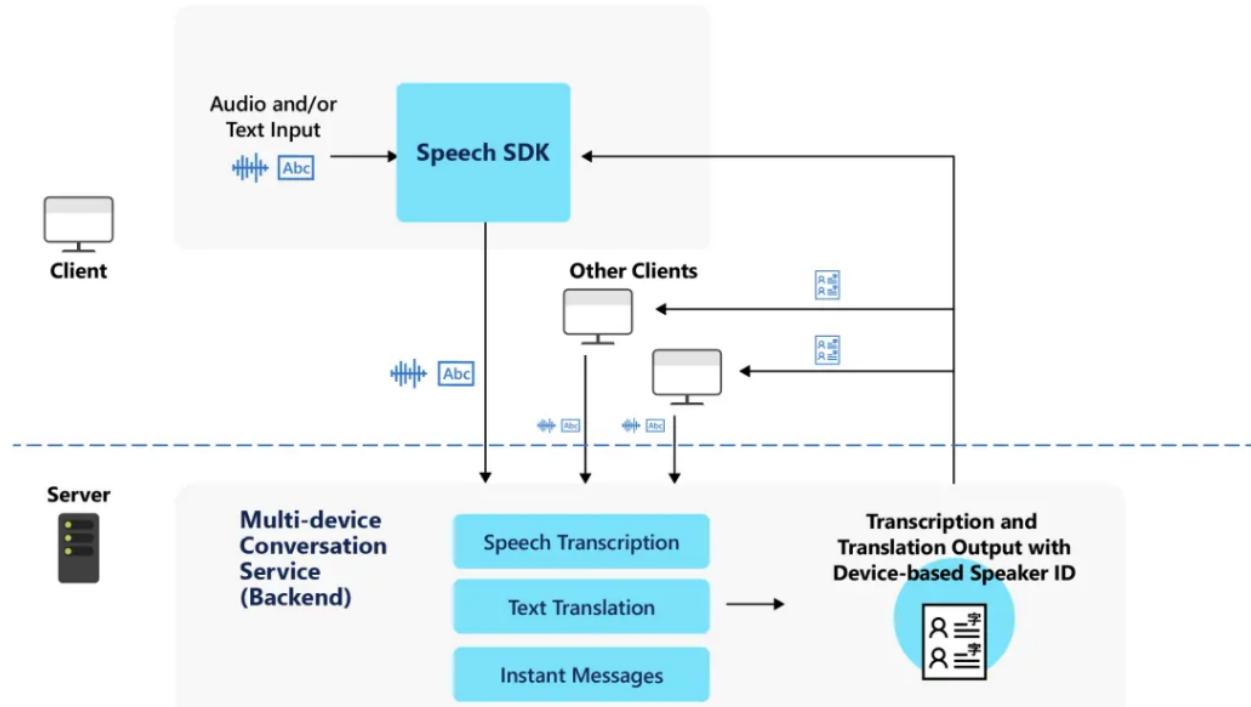
Presentations

You can also provide captions for presentations and lectures both on-screen and on the audience members' own devices. After the audience joins with the conversation code, they can see the transcript in their preferred language, on their own device.

How it works

All clients will use the Speech SDK to create or join a conversation. The Speech SDK interacts with the multi-device conversation service, which manages the lifetime of a conversation, including the list of participants, each client's chosen language(s), and messages sent.

Each client can send audio or instant messages. The service will use speech recognition to convert audio into text, and send instant messages as-is. If clients choose different languages, then the service will translate all messages to the specified language(s) of each client.



Overview of Conversation, Host, and Participant

A conversation is a session that one user starts for the other participating users to join. All clients connect to the conversation using the five-letter conversation code.

Each conversation creates metadata that includes:

- Timestamps of when the conversation started and ended
- List of all participants in the conversation, which includes each user's chosen nickname and primary language for speech or text input.

There are two types of users in a conversation: host and participant.

The host is the user who starts a conversation, and who acts as the administrator of that conversation.

- Each conversation can only have one host
- The host must be connected to the conversation for the duration of the conversation. If the host leaves the conversation, the conversation will end for all other participants.

- The host has a few extra controls to manage the conversation:
 - Lock the conversation - prevent additional participants from joining
 - Mute all participants - prevent other participants from sending any messages to the conversation, whether transcribed from speech or instant messages
 - Mute individual participants
 - Unmute all participants
 - Unmute individual participants

A participant is a user who joins a conversation.

- A participant can leave and rejoin the same conversation at any time, without ending the conversation for other participants.
- Participants cannot lock the conversation or mute/unmute others

Note

Each conversation can have up to 100 participants, of which 10 can be simultaneously speaking at any given time.

Language support

When creating or joining a conversation, each user must choose a primary language: the language that they will speak and send instant messages in, and also the language they will see other users' messages.

There are two kinds of languages: speech-to-text and text-only:

- If the user chooses a speech-to-text language as their primary language, then they will be able to use both speech and text input in the conversation.
- If the user chooses a text-only language, then they will only be able to use text input and send instant messages in the conversation. Text-only languages are the languages that are supported for text translation, but not speech-to-text. You can see available languages on the [language support](#) page.

Apart from their primary language, each participant can also specify additional languages for translating the conversation.

Below is a summary of what the user will be able to do in a multi-device conversation, based to their chosen primary language.

What the user can do in the conversation	Speech-to-text	Text-only
--	----------------	-----------

What the user can do in the conversation	Speech-to-text	Text-only
Use speech input	✓	✗
Send instant messages	✓	✓
Translate the conversation	✓	✓

ⓘ Note

For lists of available speech-to-text and text translation languages, see [supported languages](#).

Next steps

- [Translate conversations in real-time](#)

Quickstart: Multi-device Conversation

Article • 04/22/2022 • 13 minutes to read

In this quickstart, you'll learn how to use the [Speech SDK](#) to create a new multi-device conversation with translation support, as well as join an existing conversation.

ⓘ Note

The Speech SDK for Java, JavaScript, Objective-C, and Swift support Multi-device Conversation, but we haven't yet included a guide here.

You can view or download all [Speech SDK C# Samples](#) on GitHub.

Prerequisites

Before you get started, make sure to:

- ✓ [Create a Speech resource](#)
- ✓ [Setup your development environment and create an empty project](#)

Add sample code

1. Open `Program.cs`, and replace all code in it with the following code:

```
C#  
  
using Microsoft.CognitiveServices.Speech;  
using Microsoft.CognitiveServices.Speech.Audio;  
using Microsoft.CognitiveServices.Speech.Transcription;  
using System;  
using System.Threading.Tasks;  
  
namespace HelloWorld  
{  
    class Program  
    {  
        static async Task Main(string[] args)  
        {  
            await CreateConversationAsync();  
        }  
  
        static async Task CreateConversationAsync()  
        {  
            // Replace these values with the details of your Cognitive  
        }  
    }  
}
```

```
Speech subscription
    string subscriptionKey = "YourSubscriptionKey";

        // Replace below with your region identifier from here:
https://aka.ms/speech/sdkregion
    string region = "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";

        // Set this to the display name you want for the
conversation host
    string displayName = "The host";

        // Create the task completion source that will be used to
wait until the user presses Ctrl + C
    var completionSource = new TaskCompletionSource<bool>();

        // Register to listen for Ctrl + C
    Console.CancelKeyPress += (s, e) =>
    {
        completionSource.TrySetResult(true);
        e.Cancel = true; // don't terminate the current process
    };

        // Create an instance of the speech translation config
    var config =
SpeechTranslationConfig.FromSubscription(subscriptionKey, region);
    config.SpeechRecognitionLanguage = fromLanguage;
    config.AddTargetLanguage(toLanguage);

        // Create the conversation
    using (var conversation = await
Conversation.CreateConversationAsync(config).ConfigureAwait(false))
    {
        // Start the conversation so the host user and others
can join
        await
conversation.StartConversationAsync().ConfigureAwait(false);

        // Get the conversation ID. It will be up to your
scenario to determine how this is shared with other participants.
        string conversationId = conversation.ConversationId;
        Console.WriteLine($"Created '{conversationId}'"
conversation");

        // At this point, you can use the conversation object
to manage the conversation.
        // For example, to mute everyone else in the room you
can call this method:
        await
conversation.MuteAllParticipantsAsync().ConfigureAwait(false);
```

```
// Configure which audio source you want to use. If you
are using a text only language, you
    // can use the other overload of the constructor that
takes no arguments
        var audioConfig =
AudioConfig.FromDefaultMicrophoneInput();
        using (var conversationTranslator = new
ConversationTranslator(audioConfig))
{
    // You should connect all the event handlers you
need at this point
    conversationTranslator.SessionStarted += (s, e) =>
{
    Console.WriteLine($"Session started:
{e.SessionId}");
};

    conversationTranslator.SessionStopped += (s, e) =>
{
    Console.WriteLine($"Session stopped:
{e.SessionId}");
};

    conversationTranslator.Canceled += (s, e) =>
{
    switch (e.Reason)
{
        case CancellationReason.EndOfStream:
            Console.WriteLine($"End of audio
reached");
            break;

        case CancellationReason.Error:
            Console.WriteLine($"Canceled due to
error. {e.ErrorCode}: {e.ErrorDetails}");
            break;
    }
};

    conversationTranslator.ConversationExpiration +=

(s, e) =>
{
    Console.WriteLine($"Conversation will expire in
{e.ExpirationTime.TotalMinutes} minutes");
};

    conversationTranslator.ParticipantsChanged += (s,
e) =>
{
    Console.Write("The following participant(s)
have ");
    switch (e.Reason)
{
        case
ParticipantChangedReason.JoinedConversation:
            Console.Write("joined");
            break;
    }
};
```

```

        case
ParticipantChangedReason.LeftConversation:
            Console.WriteLine("left");
            break;

        case ParticipantChangedReason.Updated:
            Console.WriteLine("been updated");
            break;
    }

    Console.WriteLine(":");

    foreach (var participant in e.Participants)
    {

Console.WriteLine($"\\t{participant.DisplayName}");
    }
};

conversationTranslator.TextMessageReceived += (s,
e) =>
{
    Console.WriteLine($"Received an instant message
from '{e.Result.ParticipantId}': '{e.Result.Text}'");
    foreach (var entry in e.Result.Translations)
    {
        Console.WriteLine($"\\tTranslated into
'{entry.Key}': '{entry.Value}'");
    }
};

conversationTranslator.Transcribed += (s, e) =>
{
    Console.WriteLine($"Received a transcription
from '{e.Result.ParticipantId}': '{e.Result.Text}'");
    foreach (var entry in e.Result.Translations)
    {
        Console.WriteLine($"\\tTranslated into
'{entry.Key}': '{entry.Value}'");
    }
};

conversationTranslator.Transcribing += (s, e) =>
{
    Console.WriteLine($"Received a partial
transcription from '{e.Result.ParticipantId}': '{e.Result.Text}'");
    foreach (var entry in e.Result.Translations)
    {
        Console.WriteLine($"\\tTranslated into
'{entry.Key}': '{entry.Value}'");
    }
};

// Enter the conversation to start receiving events
await
conversationTranslator.JoinConversationAsync(conversation,
displayName).ConfigureAwait(false);

```

```
// You can now send an instant message to all other
participants in the room
    await
conversationTranslator.SendTextMessageAsync("The instant message to
send").ConfigureAwait(false);

        // If specified a speech-to-text language, you can
start capturing audio
        await
conversationTranslator.StartTranscribingAsync().ConfigureAwait(false);
            Console.WriteLine("Started transcribing. Press Ctrl
+ c to stop");

        // At this point, you should start receiving
transcriptions for what you are saying using the default microphone.
Press Ctrl+c to stop audio capture
        await completionSource.Task.ConfigureAwait(false);

        // Stop audio capture
        await
conversationTranslator.StopTranscribingAsync().ConfigureAwait(false);

        // Leave the conversation. After this you will no
longer receive events
        await
conversationTranslator.LeaveConversationAsync().ConfigureAwait(false);
    }

        // End the conversation
        await
conversation.EndConversationAsync().ConfigureAwait(false);

        // Delete the conversation. Any other participants that
are still in the conversation will be removed
        await
conversation.DeleteConversationAsync().ConfigureAwait(false);
    }
}
```

2. In the same file, replace the string `YourSubscriptionKey` with your Cognitive Speech subscription key.
 3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription.
 4. From the menu bar, choose **File > Save All**.

Build and run the application to create a new conversation

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. Once you see the `Started transcribing` message appear, you can start speaking. You'll see the transcriptions appear as you speak.
 - If you share the conversation code with the others and they join the conversation, you'll see their transcriptions as well.
4. Once you're done speaking, press `Ctrl+C` to stop audio capture, and end the conversation.

Build and run the application to join an existing conversation

1. Copy and paste the following function into your **Program.cs**:

```
C#  
  
static async Task JoinConversationAsync(string conversationId)  
{  
    // Set this to the display name you want for the participant  
    string displayName = "participant";  
  
    // Set the speech to text, or text language you want to use  
    string language = "en-US";  
  
    // Create the task completion source that will be used to wait  
    // until the user presses Ctrl + c  
    var completionSource = new TaskCompletionSource<bool>();  
  
    // Register to listen for Ctrl+C  
    Console.CancelKeyPress += (s, e) =>  
    {  
        completionSource.TrySetResult(true);  
        e.Cancel = true; // don't terminate the current process  
    };  
  
    // As a participant, you don't need to specify any subscription  
    // key, or region. You can directly create  
    // the conversation translator object
```

```

    var audioConfig = AudioConfig.FromDefaultMicrophoneInput();
    using (var conversationTranslator = new
ConversationTranslator(audioConfig))
    {
        // Register for any events you are interested here. For now
let's just register for
        // transcription, and instant message events
        conversationTranslator.TextMessageReceived += (s, e) =>
        {
            Console.WriteLine($"Received an instant message from
'{e.Result.ParticipantId}': '{e.Result.Text}'");
            foreach (var entry in e.Result.Translations)
            {
                Console.WriteLine($"{entry.Key}: '{entry.Value}'");
            }
        };
        conversationTranslator.Transcribed += (s, e) =>
        {
            Console.WriteLine($"Received a transcription from
'{e.Result.ParticipantId}': '{e.Result.Text}'");
            foreach (var entry in e.Result.Translations)
            {
                Console.WriteLine($"{entry.Key}: '{entry.Value}'");
            }
        };
        conversationTranslator.Transcribing += (s, e) =>
        {
            Console.WriteLine($"Received a partial transcription from
'{e.Result.ParticipantId}': '{e.Result.Text}'");
            foreach (var entry in e.Result.Translations)
            {
                Console.WriteLine($"{entry.Key}: '{entry.Value}'");
            }
        };
    }

        // To start receiving events, you will need to join the
conversation
        await
conversationTranslator.JoinConversationAsync(conversationId,
displayName, language).ConfigureAwait(false);

        // You can now send an instant message
        await conversationTranslator.SendTextMessageAsync("Message from
participant").ConfigureAwait(false);

        // Start capturing audio if you specified a speech-to-text
language
        await
conversationTranslator.StartTranscribingAsync().ConfigureAwait(false);
        Console.WriteLine("Started transcribing. Press Ctrl-C to
stop");

```

```

        // At this point, you should start receiving transcriptions for
        what you are saying using
        // the default microphone. Press Ctrl+C to stop audio capture
        await completionSource.Task.ConfigureAwait(false);

        // Stop audio capture
        await
    conversationTranslator.StopTranscribingAsync().ConfigureAwait(false);

        // Leave the conversation. You will stop receiving events after
        this
        await
    conversationTranslator.LeaveConversationAsync().ConfigureAwait(false);
    }
}

```

2. Replace `CreateConversationAsync();` in your `public static async Task`

`Main(string[] args)` function with:

C#

```

// Set this to the conversation you want to join
JoinConversationAsync("YourConversationId");

```

3. You'll need to create a conversation that you can join:

- a. Launch your browser and navigate to: <https://translator.microsoft.com>.
- b. Click on "Start conversation".
- c. Sign in using any of the available options.
- d. Type in a name (e.g. The host).
- e. Select a language (e.g. English).
- f. Select the **Enter** button.
- g. Note the conversation code at the top of the page.

💡 Tip

If you use the copy button on the web page, make sure you remove the `translate.it/` at the start. You only need the 5 character conversation ID (example: ABCDE).

4. Go back to Visual Studio and replace the string `YourConversationId` with the conversation ID you created in the previous step.

5. From the menu bar, select **Build > Build Solution** to build the application. The code should compile without errors now.

6. Choose Debug > Start Debugging (or press F5) to start the helloworld application.
7. Once you see the Started transcribing message appear, you can start speaking. You'll see the transcriptions appear as you speak.
 - If you go back to your browser, you should see your transcriptions appear there as you speak as well.
8. Once you're done speaking, press `ctrl+c` to stop audio capture, and end the conversation.
9. Go back to your browser and exit the conversation using the exit button in the upper right corner.

Next Steps

[Explore C# samples on GitHub](#)

Voice assistants documentation

Voice assistants using the Speech service empowers developers to create natural, human-like conversational interfaces for their applications and experiences.

About voice assistants

OVERVIEW

[What is a voice assistant?](#)

[What is a Custom Commands app?](#)

[What is Direct Line Speech?](#)

REFERENCE

[Voice assistants FAQ](#)

[Voice assistants pricing ↗](#)

GET STARTED

[Build and deploy a sample Voice Assistant to your Azure subscription \(GitHub\) ↗](#)

Develop with Custom Commands

QUICKSTART

[Create a voice assistant using Custom Commands](#)

HOW-TO GUIDE

[Create application with simple commands](#)

[Add parameters to your commands](#)

[Add validations to parameters and prompts for correction](#)

[Integrate using the Speech SDK](#)

[Setup Continuous Deployment for your Custom Commands app using Azure DevOps](#)

Develop with Direct Line Speech



QUICKSTART

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

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

[Java \(Android\)](#)



Voice enable your bot with Speech SDK

Develop with Custom Keyword



HOW-TO GUIDE

[Create a Custom Keyword](#)



REFERENCE

[Choosing an effective keyword](#)

Help and feedback



REFERENCE

[Support and help options](#)

What is a voice assistant?

Article • 05/27/2022 • 3 minutes to read

By using voice assistants with the Speech service, developers can 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 [Direct Line Speech](#) (via Azure Bot Service) for adding voice capabilities to your bots or [Custom Commands](#) for voice-command scenarios.

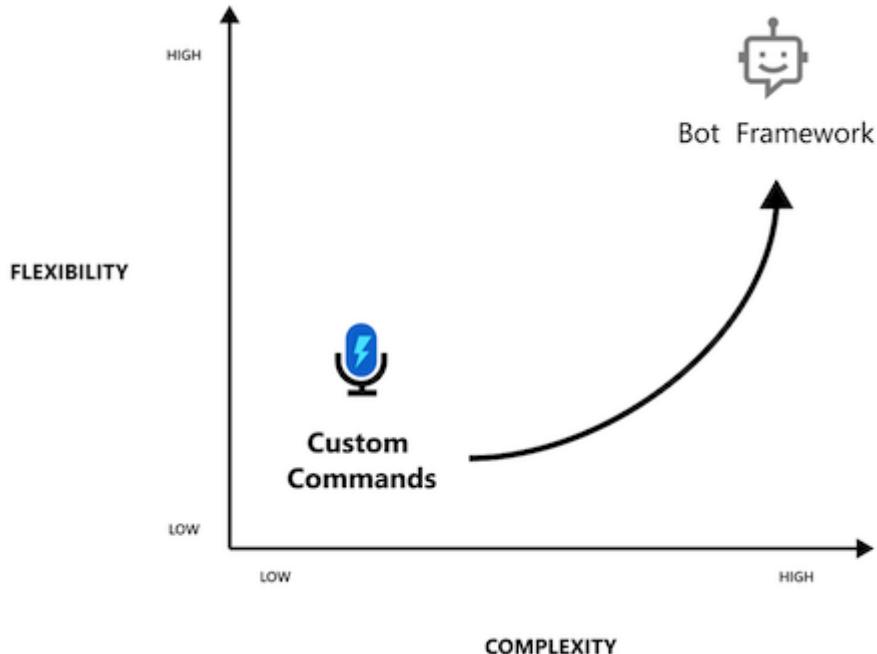
Choose an assistant solution

The first step in creating a voice assistant is to decide what you want it to do. Speech service provides multiple, complementary solutions for crafting assistant interactions. For flexibility and versatility, you can add voice in and voice out capabilities to a bot by using Azure Bot Service with the [Direct Line Speech](#) channel, or you can simply author a [Custom Commands](#) app for more straightforward voice-command scenarios.

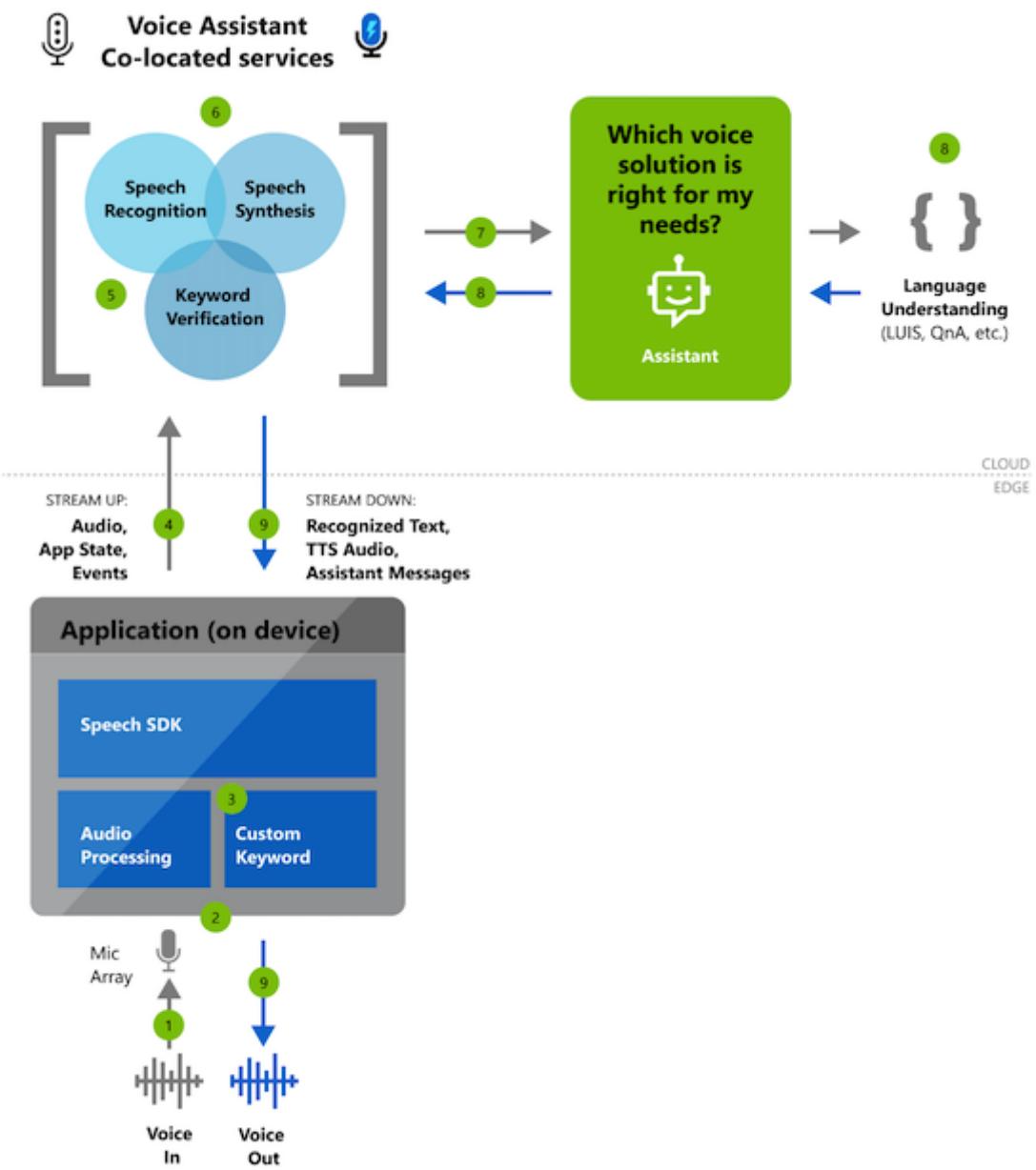
If you want...	Consider using..	Examples
Open-ended conversation with robust skills integration and full deployment control	Azure Bot Service bot with Direct Line Speech channel	<ul style="list-style-type: none">"I need to go to Seattle""What kind of pizza can I order?"
Voice-command or simple task-oriented conversations with simplified authoring and hosting	Custom Commands	<ul style="list-style-type: none">"Turn on the overhead light""Make it 5 degrees warmer"More examples at Speech Studio ↗

If you aren't yet sure what you want your assistant to do, we recommend [Direct Line Speech](#) as the best option. 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.

If you want to keep it simpler for now, [Custom Commands](#) makes it easy to build rich, voice-command apps that are optimized for voice-first interaction. Custom Commands provides a unified authoring experience, an automatic hosting model, and relatively lower complexity, all of which can help you focus on building the best solution for your voice-command scenario.



Reference architecture for building a voice assistant by using the Speech SDK



Core features

Whether you choose [Direct Line Speech](#) or [Custom Commands](#) 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 by using a custom keyword such as "Hey Contoso." An app does this with a custom keyword engine in the Speech SDK, which you can configure by going to Get started with custom keywords . Voice assistants can use service-side keyword verification to improve the accuracy of the keyword activation (versus using the device alone).

Category	Features
Speech-to-text	Voice assistants convert real-time audio into recognized text by using speech-to-text 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 through text-to-speech 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 Text to Speech (Neural TTS) voice that gives a voice to your brand. To learn more, contact us .

Get started with voice assistants

We offer the following quickstart articles, organized by programming language, that are designed to have you running code in less than 10 minutes:

- [Quickstart: Create a custom voice assistant by using Direct Line Speech](#)
- [Quickstart: Build a voice-command app by using Custom Commands](#)

Sample code and tutorials

Sample code for creating a voice assistant is available on GitHub. The samples cover the client application for connecting to your assistant in several popular programming languages.

- [Voice assistant samples on GitHub ↗](#)
- [Tutorial: Voice-enable an assistant that's built by using Azure Bot Service with the C# Speech SDK](#)
- [Tutorial: Create a Custom Commands application with simple voice commands](#)

Customization

Voice assistants that you build by using Speech service can use a full range of customization options.

- [Custom Speech](#)
- [Custom Voice](#)
- [Custom Keyword](#)

 Note

Customization options vary by language and locale. To learn more, see [Supported languages](#).

Next steps

- [Learn more about Custom Commands](#)
- [Learn more about Direct Line Speech](#)
- [Get the Speech SDK](#)

Voice assistants frequently asked questions

FAQ

If you can't find answers to your questions in this document, check out [other support options](#).

General

What is a voice assistant?

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](#) 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.

Should I use Custom Commands or Direct Line Speech? What's the difference?

[Custom Commands](#) 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.

How do I get started?

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

Where's my channel secret?

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.

I get a 401 error when connecting and nothing works. I know my Speech resource key is valid. What's going on?

When managing your Speech resource 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+ /) Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Unavailable setting Delete

Resource group (change)	: MyResourceGroup
Status	: Active
Location	West US 2
Subscription (change)	: My Subscription
Subscription ID	: My-Subscription-Id
Taqs (change)	: Click here to add taqs

API type : Speech
Pricing tier : Standard
Endpoint : <https://westus2.api.cognitive.microsoft.com/sts/v1.0/issuetoken>
Manage keys : Show access keys ...

I get recognition text back from my `DialogServiceConnector`, but I see a '1011' error and nothing from my bot. Why?

This error indicates a communication problem between your assistant and the voice assistant service.

- For Custom Commands, ensure that your Custom Commands 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.

This code still doesn't work and/or I'm getting a different error when using a `DialogServiceConnector`. What should I do?

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 Commands?

Article • 04/22/2022 • 2 minutes to read

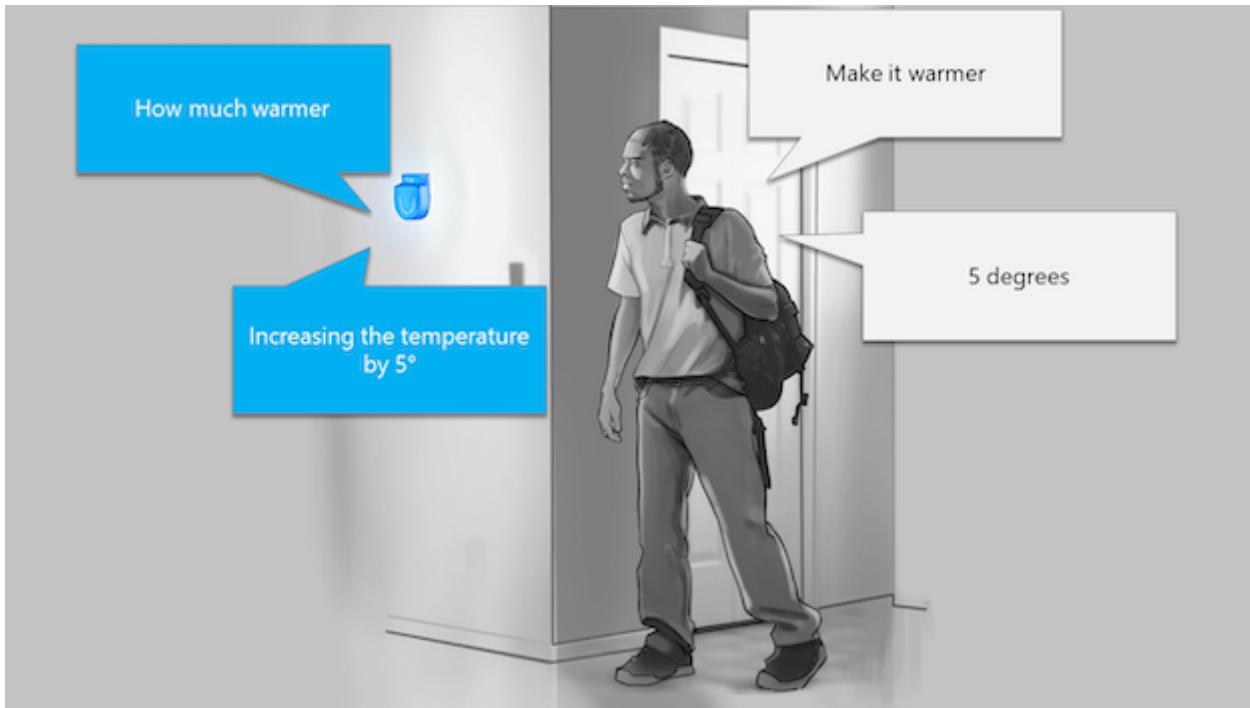
Applications such as [Voice assistants](#) listen to users and take an action in response, often speaking back. They use [speech-to-text](#) to transcribe the user's speech, then take action on the natural language understanding of the text. This action frequently includes spoken output from the assistant generated with [text-to-speech](#). Devices connect to assistants with the Speech SDK's `DialogServiceConnector` object.

Custom Commands makes it easy to build rich voice commanding apps optimized for voice-first interaction experiences. It provides a unified authoring experience, an automatic hosting model, and relatively lower complexity, helping you focus on building the best solution for your voice commanding scenarios.

Custom Commands is best suited for task completion or command-and-control scenarios, and well matched for Internet of Things (IoT) devices, ambient and headless devices. Examples include solutions for Hospitality, Retail and Automotive industries, where you want voice-controlled experiences for your guests, in-store inventory management or in-car functionality.

If you're interested in building complex conversational apps, you're encouraged to try the Bot Framework using the [Virtual Assistant Solution](#). You can add voice to any bot framework bot using Direct Line Speech.

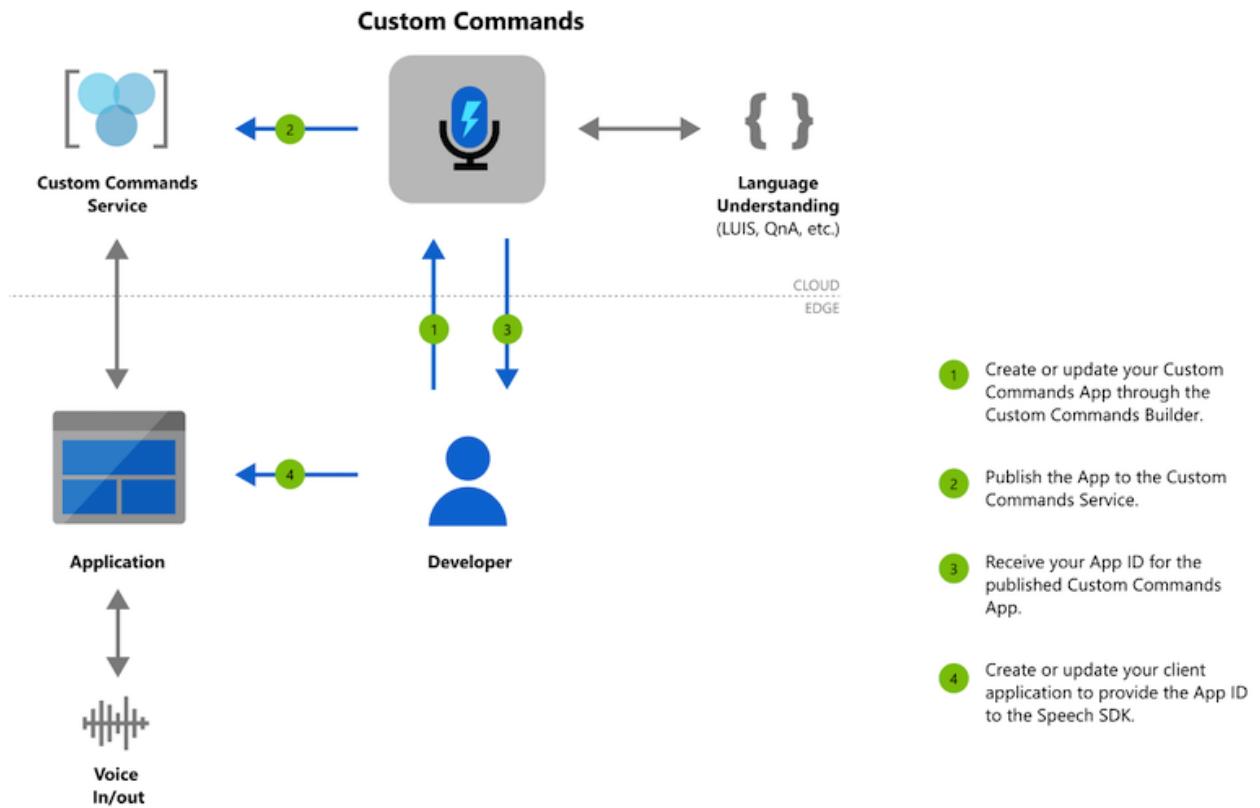
Good candidates for Custom Commands have a fixed vocabulary with well-defined sets of variables. For example, home automation tasks, like controlling a thermostat, are ideal.



Getting started with Custom Commands

Our goal with Custom Commands is to reduce your cognitive load to learn all the different technologies and focus building your voice commanding app. First step for using Custom Commands to [create an Azure Speech resource](#). You can author your Custom Commands app on the Speech Studio and publish it, after which an on-device application can communicate with it using the Speech SDK.

Authoring flow for Custom Commands



Follow our quickstart to have your first Custom Commands app running code in less than 10 minutes.

- Create a voice assistant using [Custom Commands](#)

Once you're done with the quickstart, explore our how-to guides for detailed steps for designing, developing, debugging, deploying and integrating a Custom Commands application.

Building Voice Assistants with Custom Commands

[https://www.youtube-nocookie.com/embed/1zr0umHGFyc ↗](https://www.youtube-nocookie.com/embed/1zr0umHGFyc)

Next steps

- View our Voice Assistants repo on GitHub for samples ↗
- Go to the Speech Studio to try out Custom Commands ↗
- Get the Speech SDK

Quickstart: Create a voice assistant with Custom Commands

Article • 09/20/2022 • 3 minutes to read

In this quickstart, you create and test a basic Custom Commands application using Speech Studio. You will also be able to access this application from a Windows client app.

Region Availability

At this time, Custom Commands supports speech resources created in regions that have [voice assistant capabilities](#).

Prerequisites

- ✓ [Create an Azure Speech resource in a region that supports Custom Commands](#). ↗
Refer to the [Region Availability](#) section above for list of supported regions.
- ✓ Download the sample [Smart Room Lite](#) json file.
- ✓ Download the latest version of [Windows Voice Assistant Client](#) ↗.

Go to the Speech Studio for Custom Commands

1. In a web browser, go to [Speech Studio](#) ↗.
2. Enter your credentials to sign in to the portal.

The default view is your list of Speech resources.

ⓘ Note

If you don't see the select resource page, you can navigate there by choosing "Resource" from the settings menu on the top bar.

3. Select your Speech resource, and then select **Go to Studio**.
4. Select **Custom Commands**.

The default view is a list of the Custom Commands applications you have under your selected resource.

Import an existing application as a new Custom Commands project

1. Select **New project** to create a project.
2. In the **Name** box, enter project name as `Smart-Room-Lite` (or something else of your choice).
3. In the **Language** list, select **English (United States)**.
4. Select **Browse files** and in the browse window, select the `SmartRoomLite.json` file.

New project X

Name *

Description

Language *

Import an existing application

No file selected

LUIS authoring resource *

[Create new LUIS authoring resource](#)

5. In the **LUIS authoring resource** list, select an authoring resource. If there are no valid authoring resources, create one by selecting **Create new LUIS authoring resource**.
 - a. In the **Resource Name** box, enter the name of the resource.
 - b. In the **Resource Group** list, select a resource group.
 - c. In the **Location** list, select a location.
 - d. In the **Pricing Tier** list, select a 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.

6. Next, select **Create** to create your project.
7. After the project is created, select your project. You should now see overview of your new Custom Commands application.

Try out some voice commands

1. Select **Train** at the top of the right pane.
2. Once training is completed, select **Test** and try out the following utterances:
 - Turn on the tv
 - Set the temperature to 80 degrees
 - Turn it off
 - The tv
 - Set an alarm for 5 PM

Integrate Custom Commands application in an assistant

Before you can access this application from outside Speech Studio, you need to publish the application. For publishing an application, you will need to configure prediction LUIS resource.

Update prediction LUIS resource

1. Select **Settings** in the left pane and select **Luis resources** in the middle pane.

2. Select a prediction resource, or create one by selecting **Create new resource**.

3. Select **Save**.

The screenshot shows the Microsoft Speech Studio interface with the path 'Speech Studio > Custom Commands'. The project is 'Smart-Room-Lite' and the region is 'English (United States)'. On the left, there's a sidebar with 'Commands' (containing 'FallbackCommand', 'TurnOnOff', 'SetTemperature', 'SetAlarm'), 'Web endpoints', and 'Settings' (selected). The main pane has tabs 'General' and 'LUIS resources' (selected). Under 'General', it says 'The authoring resource is used for updating, training and publishing your application. You can also use the authoring resource for 1,000 prediction requests.' Under 'LUIS resources', it says 'The prediction resource is used to recognize inputs during runtime for your application. After 1,000 predictions done with the authoring resource, you'll need to add a prediction resource.' There are dropdown menus for 'Authoring resource' (set to 'EO: SmartRoomLiteDemo-Authoring - (westus)') and 'Prediction resource' (set to 'Select a resource'). At the bottom, there's a link 'Create new resource'.

ⓘ Note

Because the authoring resource supports only 1,000 prediction endpoint requests per month, you will mandatorily need to set a LUIS prediction resource before publishing your Custom Commands application.

Publish the application

Select **Publish** on top of the right pane. Once publish completes, a new window will appear. Note down the **Application ID** and **Speech resource key** value from it. You will need these two values to be able to access the application from outside Speech Studio.

Alternatively, you can also get these values by selecting **Settings > General** section.

Access application from client

In the scope of this article, we will be using the Windows Voice Assistant client you downloaded as part of the pre-requisites. Unzip the folder.

1. Launch **VoiceAssistantClient.exe**.
2. Create a new publish profile and enter value for **Connection Profile**. In the **General Settings** section, enter values **Subscription Key** (this is same as the **Speech resource key** value you saved when publishing the application), **Subscription key region** and **Custom commands app ID**.

Settings

Connection Profile:

General settings:

Subscription key:

Subscription key region:

Custom commands app Id:

Bot Id:

User From Id:

Language:

Log file path:

Custom SR settings:

Endpoint Id:

Enabled: Click to enable

Custom TTS settings:

Voice deployment Ids:

Enabled: Click to enable

Wake word settings:

Model file path: ...

Enabled: Invalid wake word model file or location

Advanced settings:

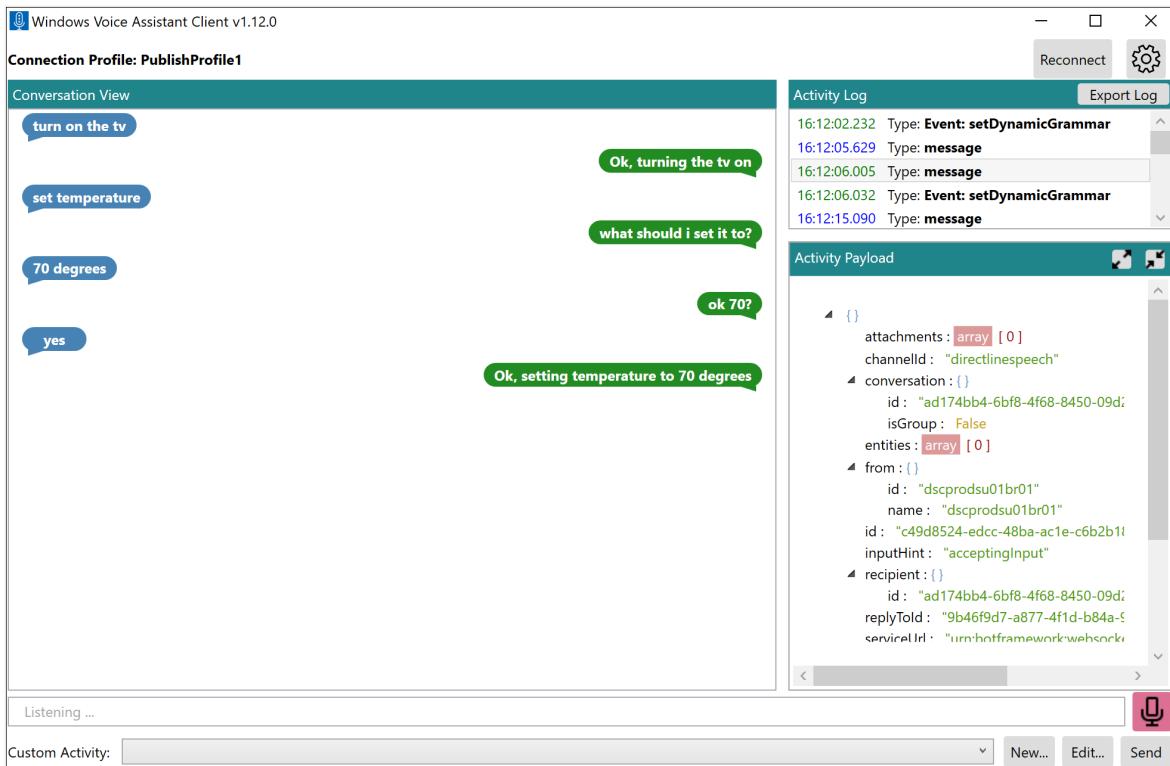
Url override:

Proxy host:

Proxy port:

You must provide a profile name Save and Apply Profile Delete Profile Discard Profile Changes

3. Select **Save and Apply Profile**.
4. Now try out the following inputs via speech/text



💡 Tip

You can click on entries in **Activity Log** to inspect the raw responses being sent from the Custom Commands service.

Next steps

In this article, you used an existing application. Next, in the [how-to sections](#), you learn how to design, develop, debug, test and integrate a Custom Commands application from scratch.

Develop Custom Commands applications

Article • 08/05/2022 • 19 minutes to read

In this how-to article, you learn how to develop and configure Custom Commands applications. The Custom Commands feature helps you build rich voice-command apps that are optimized for voice-first interaction experiences. The feature is best suited to task completion or command-and-control scenarios. It's particularly well suited for Internet of Things (IoT) devices and for ambient and headless devices.

In this article, you create an application that can turn a TV on and off, set the temperature, and set an alarm. After you create these basic commands, you'll learn about the following options for customizing commands:

- Adding parameters to commands
- Adding configurations to command parameters
- Building interaction rules
- Creating language-generation templates for speech responses
- Using Custom Voice tools

Create an application by using simple commands

Start by creating an empty Custom Commands application. For details, refer to the [quickstart](#). In this application, instead of importing a project, you create a blank project.

1. In the **Name** box, enter the project name *Smart-Room-Lite* (or another name of your choice).
2. In the **Language** list, select **English (United States)**.
3. Select or create a LUIS resource.

New project X

Name *

Description

Language *

Import an existing application

Browse files...

No file selected

Luis authoring resource *

E0: SmartRoomLiteDemo-Authoring - (westus)

Create new LUIS authoring resource

Cancel Create

Update LUIS resources (optional)

You can update the authoring resource that you selected in the **New project** window. You can also set a prediction resource.

A prediction resource is used for recognition when your Custom Commands application is published. You don't need a prediction resource during the development and testing phases.

Add a TurnOn command

In the empty Smart-Room-Lite Custom Commands application you created, add a command. The command will process an utterance, `Turn on the tv`. It will respond with the message `Ok, turning the tv on`.

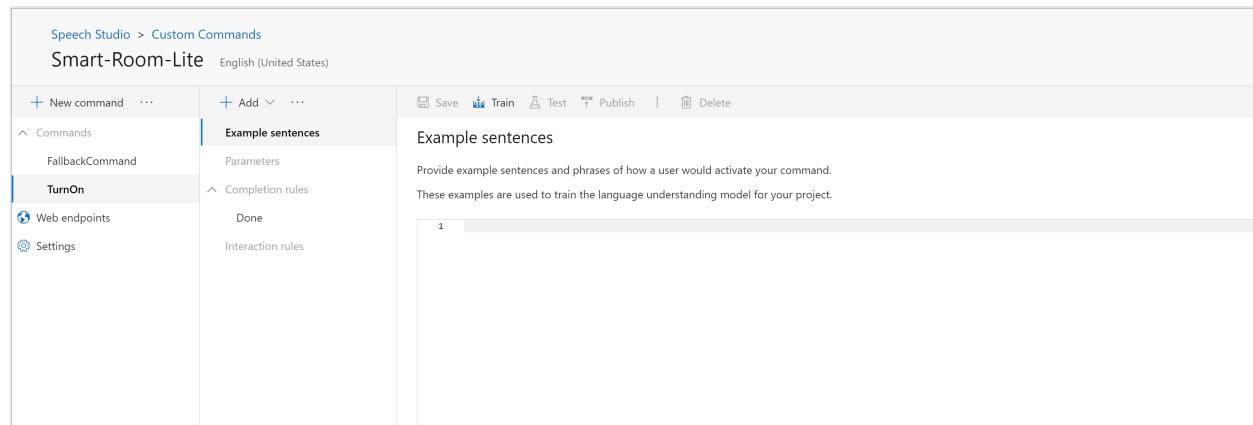
1. Create a new command by selecting **New command** at the top of the left pane.
The **New command** window opens.
2. For the **Name** field, provide the value `TurnOn`.

3. Select **Create**.

The middle pane lists the properties of the command.

The following table explains the command's configuration properties. For more information, see [Custom Commands concepts and definitions](#).

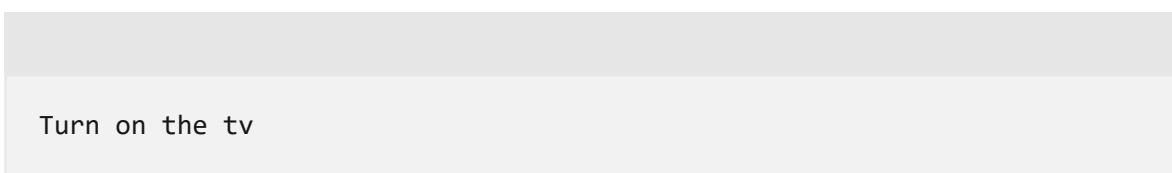
Configuration	Description
Example sentences	Example utterances the user can say to trigger this command.
Parameters	Information required to complete the command.
Completion rules	Actions to be taken to fulfill the command. Examples: responding to the user or communicating with a web service.
Interaction rules	Other rules to handle more specific or complex situations.



Add example sentences

In the **Example sentences** section, you provide an example of what the user can say.

1. In the middle pane, select **Example sentences**.
2. In the pane on the right, add examples:



3. At the top of the pane, select **Save**.

You don't have parameters yet, so you can move to the **Completion rules** section.

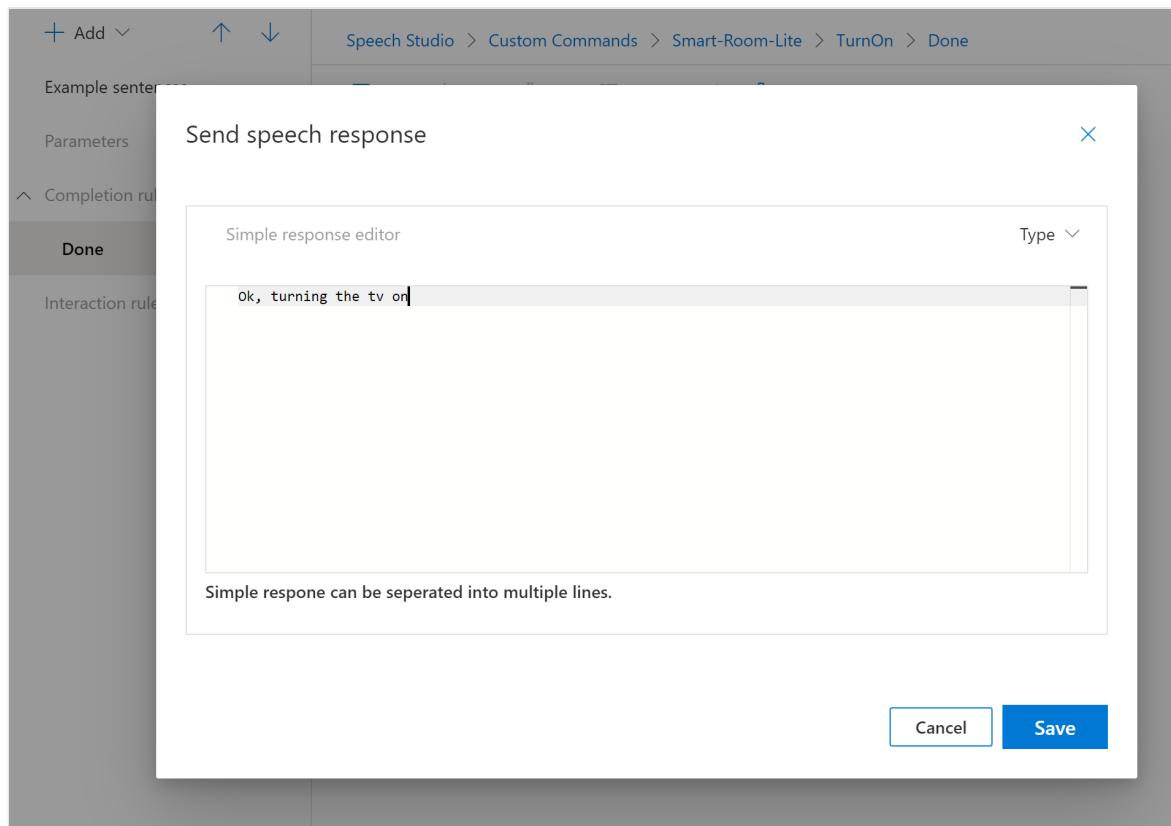
Add a completion rule

Next, the command needs a completion rule. This rule tells the user that a fulfillment action is being taken.

For more information about rules and completion rules, see [Custom Commands concepts and definitions](#).

1. Select the default completion rule **Done**. Then edit it as follows:

Setting	Suggested value	Description
Name	ConfirmationResponse	A name describing the purpose of the rule
Conditions	None	Conditions that determine when the rule can run
Actions	Send speech response > Simple editor > Ok, turning the tv on	The action to take when the rule condition is true



2. Select **Save** to save the action.

3. Back in the **Completion rules** section, select **Save** to save all changes.

! Note

You don't have to use the default completion rule that comes with the command. You can delete the default completion rule and add your own rule.

Add a SetTemperature command

Now add one more command, `SetTemperature`. This command will take a single utterance, `Set the temperature to 40 degrees`, and respond with the message `Ok, setting temperature to 40 degrees`.

To create the new command, follow the steps you used for the `TurnOn` command, but use the example sentence `Set the temperature to 40 degrees`.

Then edit the existing `Done` completion rules as follows:

Setting	Suggested value
Name	<code>ConfirmationResponse</code>
Conditions	None
Actions	<code>Send speech response > Simple editor > First variation > Ok, setting temperature to 40 degrees</code>

Select `Save` to save all changes to the command.

Add a SetAlarm command

Create a new `SetAlarm` command. Use the example sentence `Set an alarm for 9 am tomorrow`. Then edit the existing `Done` completion rules as follows:

Setting	Suggested value
Name	<code>ConfirmationResponse</code>
Conditions	None
Actions	<code>Send speech response > Simple editor > First variation > Ok, setting an alarm for 9 am tomorrow</code>

Select `Save` to save all changes to the command.

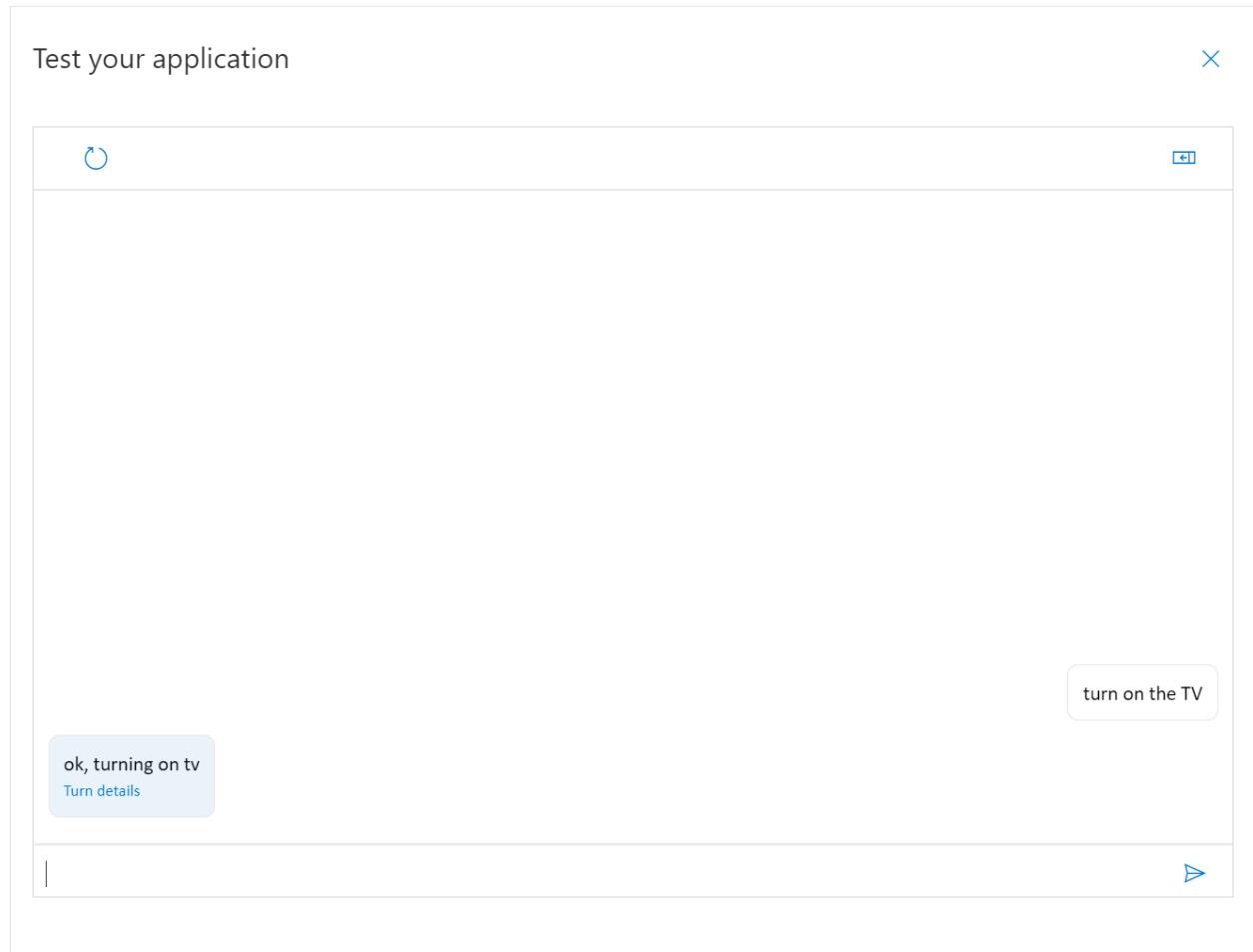
Try it out

Test the application's behavior by using the test pane:

1. In the upper-right corner of the pane, select the **Train** icon.
2. When the training finishes, select **Test**.

Try out the following utterance examples by using voice or text:

- You type: *set the temperature to 40 degrees*
Expected response: Ok, setting temperature to 40 degrees
- You type: *turn on the tv*
Expected response: Ok, turning the tv on
- You type: *set an alarm for 9 am tomorrow*
Expected response: Ok, setting an alarm for 9 am tomorrow



💡 Tip

In the test pane, you can select **Turn details** for information about how this voice input or text input was processed.

Add parameters to commands

In this section, you learn how to add parameters to your commands. Commands require parameters to complete a task. In complex scenarios, parameters can be used to define conditions that trigger custom actions.

Configure parameters for a TurnOn command

Start by editing the existing `TurnOn` command to turn on and turn off multiple devices.

1. Now that the command will handle both on and off scenarios, rename the command as *TurnOnOff*.
 - a. In the pane on the left, select the **TurnOn** command. Then next to **New command** at the top of the pane, select the edit button.
 - b. In the **Rename command** window, change the name to *TurnOnOff*.
2. Add a new parameter to the command. The parameter represents whether the user wants to turn the device on or off.
 - a. At top of the middle pane, select **Add**. From the drop-down menu, select **Parameter**.
 - b. In the pane on the right, in the **Parameters** section, in the **Name** box, add `OnOff`.
 - c. Select **Required**. In the **Add response for a required parameter** window, select **Simple editor**. In the **First variation** field, add *On or Off?*.
 - d. Select **Update**.

Add response for a required parameter

X

When a parameter is required, we need to configure a response to ask for its value.

Simple editor Template editor

First variation

On or Off?

Second variation

Cancel

Update

- e. Configure the parameter's properties by using the following table. For information about all of the configuration properties of a command, see [Custom Commands concepts and definitions](#).

Configuration	Suggested value	Description
Name	OnOff	A descriptive name for the parameter
Required	Selected	Check box indicating whether a value for this parameter is required before the command finishes.
Response for required parameter	Simple editor > On or off?	A prompt asking for the value of this parameter when it isn't known.
Type	String	Parameter type, such as Number, String, Date Time, or Geography.
Configuration	Accept predefined input values from an internal catalog	For strings, this setting limits inputs to a set of possible values.
Predefined input values	on, off	Set of possible values and their aliases.

- f. To add predefined input values, select **Add a predefined input**. In **New Item** window, type *Name* as shown in the preceding table. In this case, you're not using aliases, so you can leave this field blank.

Save Train Test Publish | Delete

Parameters

A parameter is a variable that's filled during an interaction with a Custom Commands application.

Name *

Is Global

Required

[Update response for required parameter](#)

Type *

String

Default value

Optional default value for the parameter

Configuration *

Accept predefined input values from internal catalog

Predefined input values *

on

off

+ Add a predefined input

g. Select **Save** to save all configurations of the parameter.

Add a SubjectDevice parameter

- To add a second parameter to represent the name of the devices that can be controlled by using this command, select **Add**. Use the following configuration.

Setting	Suggested value
Name	SubjectDevice
Required	Selected
Response for required parameter	Simple editor > Which device do you want to control?
Type	String
Configuration	Accept predefined input values from an internal catalog
Predefined input values	tv, fan
Aliases (tv)	television, telly

2. Select **Save**.

Modify example sentences

For commands that use parameters, it's helpful to add example sentences that cover all possible combinations. For example:

- Complete parameter information: `turn {OnOff} the {SubjectDevice}`
- Partial parameter information: `turn it {OnOff}`
- No parameter information: `turn something`

Example sentences that use varying degrees of information allow the Custom Commands application to resolve both one-shot resolutions and multiple-turn resolutions by using partial information.

With that information in mind, edit the example sentences to use these suggested parameters:

```
turn {OnOff} the {SubjectDevice}  
{SubjectDevice} {OnOff}  
turn it {OnOff}  
turn something {OnOff}  
turn something
```

Select **Save**.

Tip

In the example-sentences editor, use curly braces to refer to your parameters. For example, `turn {OnOff} the {SubjectDevice}`. Use a tab for automatic completion backed by previously created parameters.

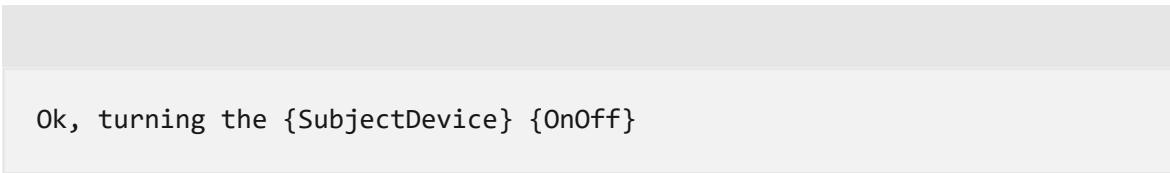
Modify completion rules to include parameters

Modify the existing completion rule `ConfirmationResponse`.

1. In the **Conditions** section, select **Add a condition**.
2. In the **New Condition** window, in the **Type** list, select **Required parameters**. In the list that follows, select both **OnOff** and **SubjectDevice**.

3. Select **Create**.

4. In the **Actions** section, edit the **Send speech response** action by hovering over it and selecting the edit button. This time, use the newly created `OnOff` and `SubjectDevice` parameters:



Ok, turning the {SubjectDevice} {OnOff}

5. Select **Save**.

Try out the changes by selecting the **Train** icon at the top of the pane on the right.

When the training finishes, select **Test**. A **Test your application** window appears. Try the following 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 do you want to control?
- Input: *the tv*
- Output: Ok, turning off the tv

Configure parameters for a SetTemperature command

Modify the `SetTemperature` command to enable it to set the temperature as the user directs.

Add a `TemperatureValue` parameter. Use the following configuration:

Configuration	Suggested value
Name	<code>TemperatureValue</code>
Required	Selected
Response for required parameter	<code>Simple editor > What temperature would you like?</code>
Type	<code>Number</code>

Edit the example utterances to use the following values.

```
set the temperature to {TemperatureValue} degrees
change the temperature to {TemperatureValue}
set the temperature
change the temperature
```

Edit the existing completion rules. Use the following configuration.

Configuration	Suggested value
Conditions	Required parameter > TemperatureValue
Actions	Send speech response > Ok, setting temperature to {TemperatureValue} degrees

Configure parameters for a SetAlarm command

Add a parameter called `DateTime`. Use the following configuration.

Setting	Suggested value
Name	<code>DateTime</code>
Required	Selected
Response for required parameter	Simple editor > For what time?
Type	<code>DateTime</code>
Date Defaults	If the date is missing, use today.
Time Defaults	If the time is missing, use the start of the day.

ⓘ Note

This article mostly uses String, Number, and DateTime parameter types. For a list of all supported parameter types and their properties, see [Custom Commands concepts and definitions](#).

Edit the example utterances. Use the following values.

```
set an alarm for {DateTime}
set alarm {DateTime}
alarm for {DateTime}
```

Edit the existing completion rules. Use the following configuration.

Setting	Suggested value
Actions	Send speech response > <code>Ok, alarm set for {DateTime}</code>

Test the three commands together by using utterances related to different commands.
(You can switch between the different commands.)

- Input: *Set an alarm*
- Output: For what time?
- Input: *Turn on the tv*
- Output: Ok, turning the tv on
- Input: *Set an alarm*
- Output: For what time?
- Input: *5 pm*
- Output: Ok, alarm set for 2020-05-01 17:00:00

Add configurations to command parameters

In this section, you learn more about advanced parameter configuration, including:

- How parameter values can belong to a set that's defined outside of the Custom Commands application.
- How to add validation clauses on the parameter values.

Configure a parameter as an external catalog entity

The Custom Commands feature allows you to configure string-type parameters to refer to external catalogs hosted over a web endpoint. So you can update the external catalog independently without editing the Custom Commands application. This approach is useful in cases where the catalog entries are numerous.

Reuse the `SubjectDevice` parameter from the `TurnOnOff` command. The current configuration for this parameter is **Accept predefined inputs from internal catalog**. This configuration refers to a static list of devices in the parameter configuration. Move out this content to an external data source that can be updated independently.

To move the content, start by adding a new web endpoint. In the pane on the left, go to the **Web endpoints** section. There, add a new web endpoint URL. Use the following configuration.

Setting	Suggested value
Name	getDevices
URL	<Your endpoint of getDevices.json>
Method	GET

Then, configure and host a web endpoint that returns a JSON file that lists the devices that can be controlled. The web endpoint should return a JSON file that's formatted like this example:

JSON
<pre>{ "fan" : [], "refrigerator" : ["fridge"], "lights" : ["bulb", "bulbs", "light", "light bulb"], "tv" : ["telly", "television"] }</pre>

Next go to the **SubjectDevice** parameter settings page. Set up the following properties.

Setting	Suggested value
Configuration	Accept predefined inputs from external catalog
Catalog endpoint	getDevices
Method	GET

Then select **Save**.

ⓘ Important

You won't see an option to configure a parameter to accept inputs from an external catalog unless you have the web endpoint set in the **Web endpoint** section in the

pane on the left.

Try it out by selecting **Train**. After the training finishes, select **Test** and try a few interactions.

- Input: *turn on*
- Output: Which device do you want to control?
- Input: *lights*
- Output: Ok, turning the lights on

ⓘ Note

You can now control all the devices hosted on the web endpoint. But you still need to train the application to test the new changes and then republish the application.

Add validation to parameters

Validations are constructs that apply to certain parameter types that allow you to configure constraints on the parameter's value. They prompt you for corrections if values don't fall within the constraints. For a list of parameter types that extend the validation construct, see [Custom Commands concepts and definitions](#).

Test out validations by using the `SetTemperature` command. Use the following steps to add a validation for the `Temperature` parameter.

1. In the pane on the left, select the **SetTemperature** command.
2. In the middle pane, select **Temperature**.
3. In the pane on the right, select **Add a validation**.
4. In the **New validation** window, configure validation as shown in the following table. Then select **Create**.

Parameter configuration	Suggested value	Description
Min Value	60	For Number parameters, the minimum value this parameter can assume

Parameter configuration	Suggested value	Description
Max Value	80	For Number parameters, the maximum value this parameter can assume
Failure response	Simple editor > First variation > Sorry, I can only set temperature between 60 and 80 degrees. What temperature do you want?	A prompt to ask for a new value if the validation fails

New Validation X

Min Value (inclusive) *

Max Value (inclusive) *

Failure response

Simple editor Template editor (radio button)

First variation

Sorry, I can only set temperature between 60 and 80 degrees

Second variation

Cancel
Create

Try it out by selecting the **Train** icon at the top of the pane on the right. After the training finishes, select **Test**. Try a few interactions:

- Input: *Set the temperature to 72 degrees*
- Output: Ok, setting temperature to 72 degrees
- Input: *Set the temperature to 45 degrees*
- Output: Sorry, I can only set temperature between 60 and 80 degrees

- Input: *make it 72 degrees instead*
- Output: Ok, setting temperature to 72 degrees

Add interaction rules

Interaction rules are *additional* rules that handle specific or complex situations. Although you're free to author your own interaction rules, in this example you use interaction rules for the following scenarios:

- Confirming commands
- Adding a one-step correction to commands

For more information about interaction rules, see [Custom Commands concepts and definitions](#).

Add confirmations to a command

To add a confirmation, you use the `SetTemperature` command. To achieve confirmation, create interaction rules by using the following steps:

1. In the pane on the left, select the `SetTemperature` command.
2. In the middle pane, add interaction rules by selecting **Add**. Then select **Interaction rules > Confirm command**.

This action adds three interaction rules. The rules ask the user to confirm the date and time of the alarm. They expect a confirmation (yes or no) for the next turn.

- a. Modify the **Confirm command** interaction rule by using the following configuration:
 - i. Change the name to **Confirm temperature**.
 - ii. The condition **All required parameters** has already been added.
 - iii. Add a new action: **Type > Send speech response > Are you sure you want to set the temperature as {TemperatureValue} degrees?**
 - iv. In the **Expectations** section, leave the default value of **Expecting confirmation from user**.

Save Train Test Publish | Delete

Interaction rules

The interaction rules are executed on each interaction with the user. You can use them to add custom business logic to your commands. The rules are executed in the order they are defined.

Name *

Confirm temperature

Conditions

All required parameters

+ Add a condition

Actions

Send speech response → Are you sure you want to set the temperature as {Tem...

+ Add an action

Expectations

Expecting confirmation from user

- b. Modify the **Confirmation succeeded** interaction rule to handle a successful confirmation (the user said yes).
 - i. Change the name to **Confirmation temperature succeeded**.
 - ii. Leave the existing **Confirmation was successful** condition.
 - iii. Add a new condition: **Type > Required parameters > TemperatureValue**.
 - iv. Leave the default **Post-execution state** value as **Execute completion rules**.
- c. Modify the **Confirmation denied** interaction rule to handle scenarios when confirmation is denied (the user said no).
 - i. Change the name to **Confirmation temperature denied**.
 - ii. Leave the existing **Confirmation was denied** condition.
 - iii. Add a new condition: **Type > Required parameters > TemperatureValue**.
 - iv. Add a new action: **Type > Send speech response > No problem. What temperature then?**
 - v. Change the default **Post-execution state** value to **Wait for user's input**.

Important

In this article, you use the built-in confirmation capability. You can also manually add interaction rules one by one.

Try out the changes by selecting **Train**. When the training finishes, select **Test**.

- **Input:** Set temperature to 80 degrees
- **Output:** are you sure you want to set the temperature as 80 degrees?
- **Input:** No
- **Output:** No problem. What temperature then?
- **Input:** 72 degrees
- **Output:** are you sure you want to set the temperature as 72 degrees?
- **Input:** Yes
- **Output:** OK, setting temperature to 72 degrees

Implement corrections in a command

In this section, you'll configure a one-step correction. This correction is used after the fulfillment action has run. You'll also see an example of how a correction is enabled by default if the command isn't fulfilled yet. To add a correction when the command isn't finished, add the new parameter `AlarmTone`.

In the left pane, select the **SetAlarm** command. Then and add the new parameter `AlarmTone`.

- **Name** > `AlarmTone`
- **Type** > `String`
- **Default Value** > `Chimes`
- **Configuration** > **Accept predefined input values from the internal catalog**
- **Predefined input values** > `Chimes`, `Jingle`, and `Echo` (These values are individual predefined inputs.)

Next, update the response for the **DateTime** parameter to **Ready to set alarm with tone as {AlarmTone}**. For what time?. Then modify the completion rule as follows:

1. Select the existing completion rule **ConfirmationResponse**.
2. In the pane on the right, hover over the existing action and select **Edit**.
3. Update the speech response to `OK, alarm set for {DateTime}. The alarm tone is {AlarmTone}`.

Important

The alarm tone can change without any explicit configuration in an ongoing command. For example, it can change when the command hasn't finished yet. A correction is enabled *by default* for all of the command parameters, regardless of the turn number, if the command is yet to be fulfilled.

Implement a correction when a command is finished

The Custom Commands platform allows for one-step correction even when the command has finished. This feature isn't enabled by default. It must be explicitly configured.

Use the following steps to configure a one-step correction:

1. In the **SetAlarm** command, add an interaction rule of the type **Update previous command** to update the previously set alarm. Rename the interaction rule as **Update previous alarm**.
2. Leave the default condition: **Previous command needs to be updated**.
3. Add a new condition: **Type > Required Parameter > DateTime**.
4. Add a new action: **Type > Send speech response > Simple editor > Updating previous alarm time to {DateTime}**.
5. Leave the default **Post-execution state** value as **Command completed**.

Try out the changes by selecting **Train**. Wait for the training to finish, and then select **Test**.

- **Input:** Set an alarm.
- **Output:** Ready to set alarm with tone as Chimes. For what time?
- **Input:** Set an alarm with the tone as Jingle for 9 am tomorrow.
- **Output:** OK, alarm set for 2020-05-21 09:00:00. The alarm tone is Jingle.
- **Input:** No, 8 am.
- **Output:** Updating previous alarm time to 2020-05-29 08:00.

Note

In a real application, in the **Actions** section of this correction rule, you'll also need to send back an activity to the client or call an HTTP endpoint to update the alarm time in your system. This action should be solely responsible for updating the alarm time. It shouldn't be responsible for any other attribute of the command. In this case, that attribute would be the alarm tone.

Add language-generation templates for speech responses

Language-generation (LG) templates allow you to customize the responses sent to the client. They introduce variance into the responses. You can achieve language generation by using:

- Language-generation templates.
- Adaptive expressions.

Custom Commands templates are based on the Bot Framework's [LG templates](#). Because the Custom Commands feature creates a new LG template when required (for speech responses in parameters or actions), you don't have to specify the name of the LG template.

So you don't need to define your template like this:

```
# CompletionAction
- Ok, turning {OnOff} the {SubjectDevice}
- Done, turning {OnOff} the {SubjectDevice}
- Proceeding to turn {OnOff} {SubjectDevice}
```

Instead, you can define the body of the template without the name, like this:

Template editor

```
1 - Ok, turning {OnOff} the {SubjectDevice}
2 - Done, turned {OnOff} the {SubjectDevice}
3 - Proceeding to turn {OnOff} {SubjectDevice}
```

This change introduces variation into the speech responses that are sent to the client. For an utterance, the corresponding speech response is randomly picked out of the provided options.

By taking advantage of LG templates, you can also define complex speech responses for commands by using adaptive expressions. For more information, see the [LG templates format](#).

By default, the Custom Commands feature supports all capabilities, with the following minor differences:

- In the LG templates, entities are represented as `${entityName}`. The Custom Commands feature doesn't use entities. But you can use parameters as variables with either the `${parameterName}` representation or the `{parameterName}` representation.
- The Custom Commands feature doesn't support template composition and expansion, because you never edit the `.lg` file directly. You edit only the responses of automatically created templates.
- The Custom Commands feature doesn't support custom functions that LG injects. Predefined functions are supported.
- The Custom Commands feature doesn't support options, such as `strict`, `replaceNull`, and `lineBreakStyle`.

Add template responses to a TurnOnOff command

Modify the `TurnOnOff` command to add a new parameter. Use the following configuration.

Setting	Suggested value
Name	<code>SubjectContext</code>
Required	Unselected
Type	<code>String</code>
Default value	<code>all</code>
Configuration	Accept predefined input values from internal catalog
Predefined input values	<code>room</code> , <code>bathroom</code> , <code>all</code>

Modify a completion rule

Edit the **Actions** section of the existing completion rule `ConfirmationResponse`. In the **Edit action** window, switch to **Template Editor**. Then replace the text with the following example.

```
- IF: @{SubjectContext == "all" && SubjectDevice == "lights"}  
- Ok, turning all the lights {OnOff}
```

```
- ELSEIF: {@{SubjectDevice == "lights"}}
  - Ok, turning {OnOff} the {SubjectContext} {SubjectDevice}
- ELSE:
  - Ok, turning the {SubjectDevice} {OnOff}
  - Done, turning {OnOff} the {SubjectDevice}
```

Train and test your application by using the following input and output. Notice the variation of responses. The variation is created by multiple alternatives of the template value and also by use of adaptive expressions.

- Input: *turn on the tv*
- Output: Ok, turning the tv on
- Input: *turn on the tv*
- Output: Done, turned on the tv
- Input: *turn off the lights*
- Output: Ok, turning all the lights off
- Input: *turn off room lights*
- Output: Ok, turning off the room lights

Use a custom voice

Another way to customize Custom Commands responses is to select an output voice. Use the following steps to switch the default voice to a custom voice:

1. In your Custom Commands application, in the pane on the left, select **Settings**.
2. In the middle pane, select **Custom Voice**.
3. In the table, select a custom voice or public voice.
4. Select **Save**.

Speech Studio > Custom Commands

Smart-Room-Lite English (United States)

+ New command ...

Commands

- FallbackCommand
- TurnOnOff
- SetTemperature
- SetAlarm

Web endpoints

Settings

Settings

General

LUIS resources

Custom keywords

Custom voice

Save

Custom voice

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.

Note that a Custom Voice must be associated with the same Speech Subscription as this project.

Language

English (US) ▾

Custom voice Public voice

	Name ↑↓	Gender ↑↓	Type ↑
⟨⟩ Aria24kRUS	Female	Standard	
⟨⟩ AriaRUS	Female	Standard	
⟨⟩ BenjaminRUS	Male	Standard	
⟨⟩ Guy24kRUS	Male	Standard	
⟨⟩ Jessa24kRUS	Female	Standard	
⟨⟩ JessaRUS	Female	Standard	
⟨⟩ ZiraRUS	Female	Standard	

ⓘ Note

For public voices, neural types are available only for specific regions. For more information, see [Speech service supported regions](#).

You can create custom voices on the [Custom Voice](#) project page. For more information, see [Get started with Custom Voice](#).

Now the application will respond in the selected voice, instead of the default voice.

Next steps

- Learn how to [integrate your Custom Commands application](#) with a client app by using the Speech SDK.
- [Set up continuous deployment](#) for your Custom Commands application by using Azure DevOps.

Integrate with a client application using Speech SDK

Article • 09/20/2022 • 7 minutes to read

In this article, you learn how to make requests to a published Custom Commands application from the Speech SDK running in an UWP application. In order to establish a connection to the Custom Commands application, you need:

- Publish a Custom Commands application and get an application identifier (App ID)
- Create a Universal Windows Platform (UWP) 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, you can do so following the quickstarts:

- ✓ [Create a Custom Commands application](#)

You'll also need:

- ✓ [Visual Studio 2019](#) or higher. This guide is based on Visual Studio 2019.
- ✓ An Azure Cognitive Services Speech resource key and region: Create a Speech resource on the [Azure portal](#). For more information, see [Create a new Azure Cognitive Services resource](#).
- ✓ [Enable your device for development](#)

Step 1: Publish Custom Commands application

1. Open your previously created Custom Commands application
2. Go to **Settings**, select **Luis resource**
3. If **Prediction resource** is not assigned, select a query prediction key or create a new one

Query prediction key is always required before publishing an application. For more information about LUIS resources, reference [Create LUIS Resource](#)

4. Go back to editing Commands, Select **Publish**

Publish your application

PeletonTest has been successfully published.

To use your application from the Speech SDK enter the custom commands application id and the subscription key in your client application.

Application id

abcdefgh-ijkl-mnop-qrst-uvwsyz123456

Speech resource key

.....

Show Key

To use your application from other Bot using the BotFramework, download the Remote skills manifest from the [Settings → Remote Skills](#) section.

[Close](#)

5. Copy the App ID from the publish notification for later use

6. Copy the Speech Resource Key for later use

Step 2: Create a Visual Studio project

Create a Visual Studio project for UWP development and [install the Speech SDK](#).

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:

XML

```
<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="0,10,10,0"
                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>
        </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:

```
C#
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Dialog;
using System;
using System.IO;
using System.Text;
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.FromEle-
                ment(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
        }
    }
```

ⓘ Note

If you see error: "The type 'Object' is defined in an assembly that is not referenced"

- a. Right-click your solution.
- b. Choose **Manage NuGet Packages for Solution**, Select **Updates**
- c. If you see **Microsoft.NETCore.UniversalWindowsPlatform** in the update list, Update **Microsoft.NETCore.UniversalWindowsPlatform** to newest version

3. Add the following code to the method body of `InitializeDialogServiceConnector`

C#

```
// This code creates the `DialogServiceConnector` with your resource
information.
// create a DialogServiceConfig by providing a Custom Commands
application id and Speech resource 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
Speech resource key
const string region = "YourServiceRegion"; // The Speech resource
region.

var speechCommandsConfig =
CustomCommandsConfig.FromSubscription(speechCommandsApplicationId,
speechSubscriptionKey, region);
speechCommandsConfig SetProperty(PropertyId.SpeechServiceConnection_Rec
oLanguage, "en-us");
connector = new DialogServiceConnector(speechCommandsConfig);
```

4. Replace the strings `YourApplicationId`, `YourSpeechSubscriptionKey`, and `YourServiceRegion` with your own values for your app, speech key, and `region`

5. Append the following code snippet to the end of the method body of

InitializeDialogServiceConnector

C#

```
//  
// 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 += (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

C#

```

// 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();
}

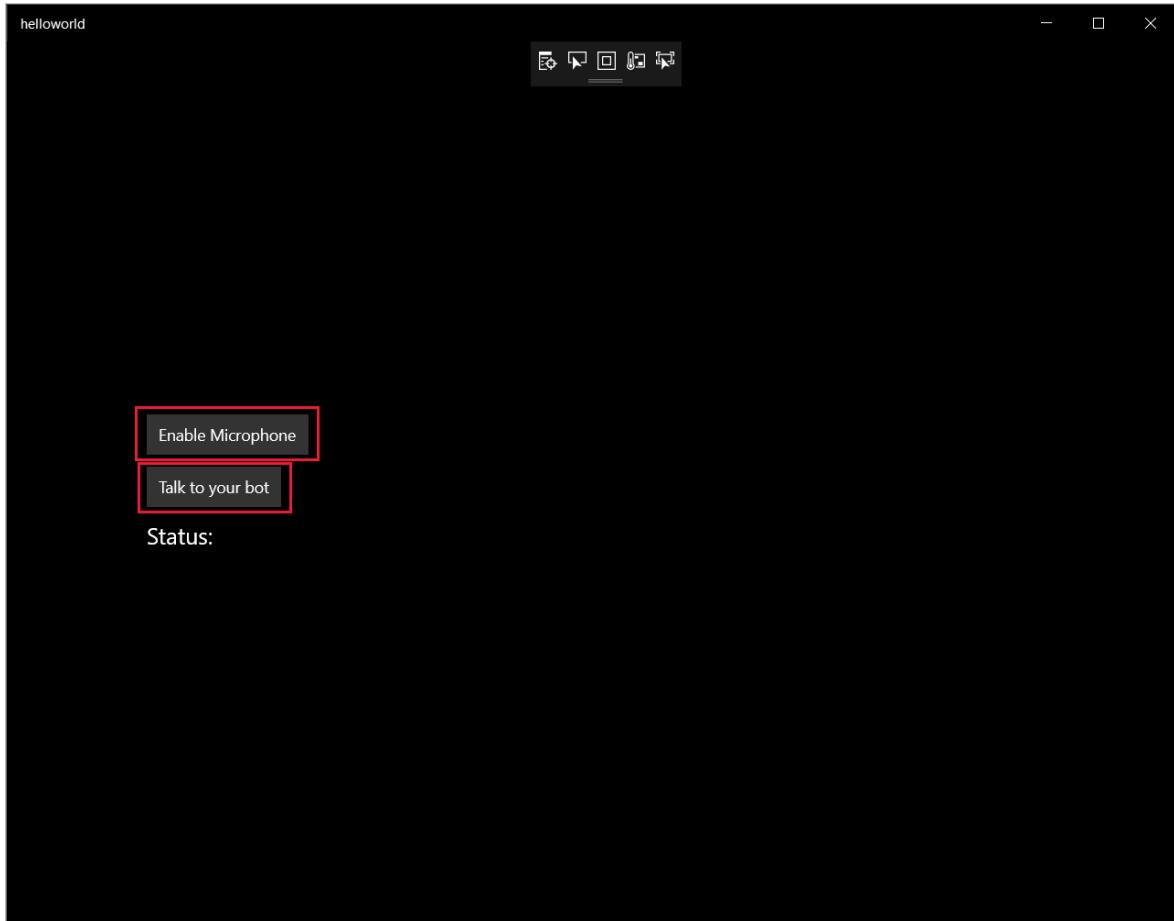
try
{
    // Start sending audio
    await connector.ListenOnceAsync();
}
catch (Exception ex)
{
    NotifyUser($"Exception: {ex.ToString()}", NotifyType.ErrorMessage);
}

```

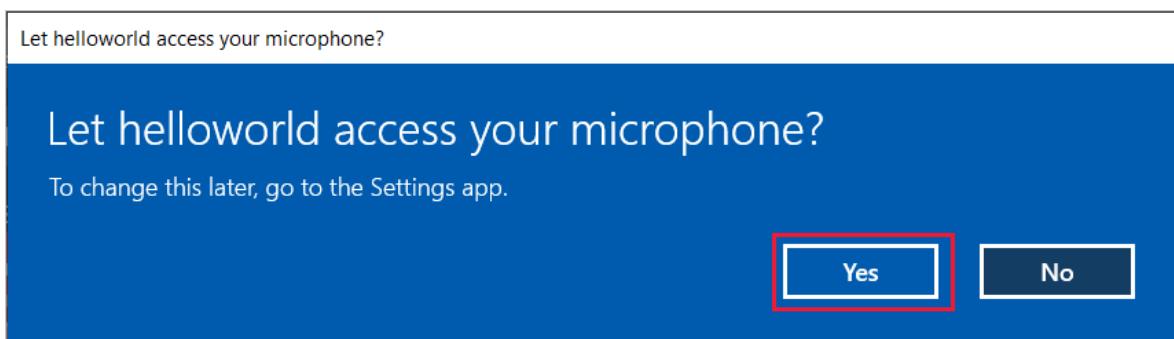
7. From the menu bar, choose **File > Save All** to save your changes

Try it out

1. From the menu bar, choose **Build > Build Solution** to build the application. The code should compile without errors.
2. Choose **Debug > Start Debugging** (or press **F5**) to start the application. The **helloworld** window appears.



3. Select **Enable Microphone**. If the access permission request pops up, select **Yes**.



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: send activity to client application \(Preview\)](#)

Send Custom Commands activity to client application

Article • 05/12/2022 • 2 minutes to read

In this article, you learn how to send activity from a Custom Commands application to a client application running the Speech SDK.

You complete the following tasks:

- 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 [↗](#) or higher. This guide uses Visual Studio 2019
- ✓ An Azure Cognitive Services Speech resource key and region: Create a Speech resource on the [Azure portal ↗](#). For more information, see [Create a new Azure Cognitive Services resource](#).
- ✓ A previously [created Custom Commands app](#)
- ✓ A Speech SDK enabled client app: [How-to: Integrate with a client application using Speech SDK](#)

Setup Send activity to client

1. Open the Custom Commands application you previously created
2. Select **TurnOnOff** command, select **ConfirmationResponse** under completion rule, then select **Add an action**
3. Under **New Action-Type**, select **Send activity to client**
4. Copy the JSON below to **Activity content**

JSON

```
{  
  "type": "event",  
  "name": "UpdateDeviceState",  
  "value": {  
    "state": "{Onoff}",  
  }  
}
```

```
        "device": "{SubjectDevice}"  
    }  
}
```

5. Click **Save** to create a new rule with a Send Activity action, **Train** and **Publish** the change

Completion rules

Completion rules run when all conditions are met and execute actions to fulfill the command.

Name *

ConfirmationResponse

Conditions

Required parameters → OnOff, SubjectDevice

+ Add a condition

Actions

Send speech response → Ok, turning {OnOff} the {SubjectDevice}

Send activity to client → (type, event)

+ Add an action

Integrate with client application

In [How-to: Setup client application with Speech SDK \(Preview\)](#), you created a UWP client application with Speech SDK that handled commands such as `turn on the tv`, `turn off the fan`. With some visuals added, you can see the result of those commands.

To Add labeled boxes with text indicating **on** or **off**, add the following XML block of StackPanel to `MainPage.xaml`.

XML

```
<StackPanel Orientation="Vertical" H.....>  
.....
```

```

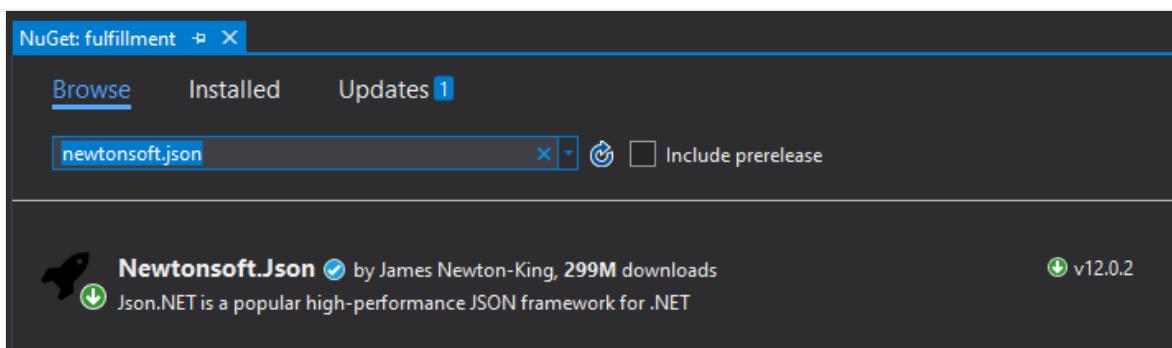
</StackPanel>
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
Margin="20">
    <Grid x:Name="Grid_TV" Margin="50, 0" Width="100" Height="100"
Background="LightBlue">
        <StackPanel>
            <TextBlock Text="TV" Margin="0, 10" TextAlignment="Center"/>
            <TextBlock 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>
<MediaElement ...../>

```

Add reference libraries

Since you've created a JSON payload, you need to add a reference to the [JSON.NET](#) library to handle deserialization.

1. Right-click your solution.
2. Choose **Manage NuGet Packages for Solution**, Select **Browse**
3. If you already installed **Newtonsoft.json**, make sure its version is at least 12.0.3. If not, go to **Manage NuGet Packages for Solution - Updates**, search for **Newtonsoft.json** to update it. This guide is using version 12.0.3.



4. Also, make sure NuGet package **Microsoft.NETCore.UniversalWindowsPlatform** is at least 6.2.10. This guide is using version 6.2.10.

In ` MainPage.xaml.cs`, add

C#

```
using Newtonsoft.Json;
using Windows.ApplicationModel.Core;
using Windows.UI.Core;
```

Handle the received payload

In `InitializeDialogServiceConnector`, replace the `ActivityReceived` event handler with following code. The modified `ActivityReceived` event handler will extract the payload from the activity and change the visual state of the tv or fan respectively.

C#

```
connector.ActivityReceived += async (sender, activityReceivedEventArgs) =>
{
    NotifyUser($"Activity received, hasAudio=
{activityReceivedEventArgs.HasAudio} activity=
{activityReceivedEventArgs.Activity}");

    dynamic activity =
JsonConvert.DeserializeObject(activityReceivedEventArgs.Activity);
    var name = activity?.name != null ? activity.name.ToString() :
string.Empty;

    if (name.Equals("UpdateDeviceState"))
    {
        Debug.WriteLine("Here");
        var state = activity?.value?.state != null ?
activity.value.state.ToString() : string.Empty;
        var device = activity?.value?.device != null ?
activity.value.device.ToString() : string.Empty;

        if (state.Equals("on") || state.Equals("off"))
        {
            switch (device)
            {
                case "tv":
                    await
CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(
                        CoreDispatcherPriority.Normal, () => { State_TV.Text
= state; });
                    break;
                case "fan":
                    await
CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(
                        CoreDispatcherPriority.Normal, () => {
State_Fan.Text = state; });
                    break;
                default:
                    NotifyUser($"Received request to set unsupported device
{device}");
            }
        }
    }
}
```

```

{device} to {state}");
        break;
    }
}
else {
    NotifyUser($"Received request to set unsupported state
{state}");
}
}

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: set up web endpoints](#)

Set up web endpoints

Article • 05/12/2022 • 7 minutes to read

In this article, you will learn how to setup web endpoints in a Custom Commands application that allow you to make HTTP requests from a client application. You will complete the following tasks:

- Set up web endpoints in Custom Commands application
- Call web endpoints in Custom Commands application
- Receive the web endpoints response
- Integrate the web endpoints response into a custom JSON payload, send, and visualize it from a C# UWP Speech SDK client application

Prerequisites

- ✓ [Visual Studio 2019](#)
- ✓ An Azure Cognitive Services Speech resource key and region: Create a Speech resource on the [Azure portal](#). For more information, see [Create a new Azure Cognitive Services resource](#).
- ✓ A Custom Commands app (see [Create a voice assistant using Custom Commands](#))
- ✓ A Speech SDK enabled client app (see [Integrate with a client application using Speech SDK](#))

Deploy an external web endpoint using Azure Function app

For this tutorial, you need an HTTP endpoint that maintains states for all the devices you set up in the **TurnOnOff** command of your Custom Commands application.

If you already have a web endpoint you want to call, skip to the [next section](#).

Alternatively, the next section provides details about a default hosted web endpoint you can use if you want to skip this section.

Input format of Azure function

Next, you will deploy an endpoint using [Azure Functions](#). The following is the format of a Custom Commands event that is passed to your Azure function. Use this information when you're writing your Azure Function app.

JSON

```
{  
    "conversationId": "string",  
    "currentCommand": {  
        "name": "string",  
        "parameters": {  
            "SomeParameterName": "string",  
            "SomeOtherParameterName": "string"  
        }  
    },  
    "currentGlobalParameters": {  
        "SomeGlobalParameterName": "string",  
        "SomeOtherGlobalParameterName": "string"  
    }  
}
```

The following table describes the key attributes of this input:

Attribute	Explanation
conversationId	The unique identifier of the conversation. Note that this ID can be generated by the client app.
currentCommand	The command that's currently active in the conversation.
name	The name of the command. The <code>parameters</code> attribute is a map with the current values of the parameters.
currentGlobalParameters	A map like <code>parameters</code> , but used for global parameters.

For the **DeviceState** Azure Function, an example Custom Commands event will look like following. This will act as an **input** to the function app.

JSON

```
{  
    "conversationId": "someConversationId",  
    "currentCommand": {  
        "name": "TurnOnOff",  
        "parameters": {  
            "item": "tv",  
            "value": "on"  
        }  
    }  
}
```

Azure Function output for a Custom Command app

If output from your Azure Function is consumed by a Custom Commands app, it should appear in the following format. See [Update a command from a web endpoint](#) for details.

JSON

```
{  
  "updatedCommand": {  
    "name": "SomeCommandName",  
    "updatedParameters": {  
      "SomeParameterName": "SomeParameterValue"  
    },  
    "cancel": false  
  },  
  "updatedGlobalParameters": {  
    "SomeGlobalParameterName": "SomeGlobalParameterValue"  
  }  
}
```

Azure Function output for a client application

If output from your Azure Function is consumed by a client application, the output can take whatever form the client application requires.

For our **DeviceState** endpoint, output of your Azure function is consumed by a client application instead of the Custom Commands application. Example output of the Azure function should look like the following:

JSON

```
{  
  "TV": "on",  
  "Fan": "off"  
}
```

This output should be written to an external storage, so that you can maintain the state of devices. The external storage state will be used in the [Integrate with client application](#) section below.

Deploy Azure function

We provide a sample you can configure and deploy as an Azure Functions app. To create a storage account for our sample, follow these steps.

1. Create table storage to save device state. In the Azure portal, create a new resource of type **Storage account** by name **devicestate**.

2. Copy the **Connection string** value from **devicestate -> Access keys**. You will need to add this string secret to the downloaded sample Function App code.
3. Download sample [Function App code ↗](#).
4. Open the downloaded solution in Visual Studio 2019. In **Connections.json**, replace **STORAGE_ACCOUNT_SECRET_CONNECTION_STRING** with the secret from Step 2.
5. Download the **DeviceStateAzureFunction** code.

To deploy the sample app to Azure Functions, follow these steps.

1. [Deploy](#) the Azure Functions app.
2. Wait for deployment to succeed and go the deployed resource on the Azure portal.
3. Select **Functions** in the left pane, and then select **DeviceState**.
4. In the new window, select **Code + Test** and then select **Get function URL**.

Setup web endpoints in Custom Commands

Let's hook up the Azure function with the existing Custom Commands application. In this section, you will use an existing default **DeviceState** endpoint. If you created your own web endpoint using Azure Function or otherwise, use that instead of the default <https://webendpointexample.azurewebsites.net/api/DeviceState>.

1. Open the Custom Commands application you previously created.
2. Go to **Web endpoints**, click **New web endpoint**.

Setting	Suggested value	Description
Name	UpdateDeviceState	A Web endpoint is an external HTTP-based API, or URL, that's used to integrate commands with external systems.
URL *	https://webendpointexample.azurewebsites.net/api/DeviceState	
Method *	POST	
Headers	app → your-app-name	
	+ Add a header	

Setting	Suggested value	Description
Name	UpdateDeviceState	Name for the web endpoint.
URL	https://webendpointexample.azurewebsites.net/api/DeviceState	The URL of the endpoint you wish your custom command app to talk to.
Method	POST	The allowed interactions (such as GET, POST) with your endpoint.
Headers	Key: app, Value: take the first 8 digits of your applicationId	The header parameters to include in the request header.

ⓘ Note

- The example web endpoint created using **Azure Functions**, which hooks up with the database that saves the device state of the tv and fan.
- The suggested header is only needed for the example endpoint.
- To make sure the value of the header is unique in our example endpoint, take the first 8 digits of your **applicationId**.
- In real world, the web endpoint can be the endpoint to the **IOT hub** that manages your devices.

3. Click **Save**.

Call web endpoints

1. Go to **TurnOnOff** command, select **ConfirmationResponse** under completion rule, then select **Add an action**.

2. Under New Action-Type, select **Call web endpoint**
3. In **Edit Action - Endpoints**, select **UpdateDeviceState**, which is the web endpoint we created.
4. In **Configuration**, put the following values:

Edit Action

Type *

Call web endpoint

Endpoint *

UpdateDeviceState

Configuration

Query parameters

item={SubjectDevice}&&value={OnOff}

Body content

1 { }

Setting	Suggested value	Description
Endpoints	UpdateDeviceState	The web endpoint you wish to call in this action.
Query parameters	item={SubjectDevice}&&value={OnOff}	The query parameters to append to the web endpoint URL.

Setting	Suggested value	Description
Body content	N/A	The body content of the request.

! Note

- The suggested query parameters are only needed for the example endpoint

5. In **On Success - Action to execute**, select **Send speech response**.

In **Simple editor**, enter `{SubjectDevice} is {OnOff}`.

On Success

Action to execute *

Send speech response

Simple editor Template editor

First variation

`{SubjectDevice} is {OnOff}.`

Second variation

Setting	Suggested value	Description
Action to execute	Send speech response	Action to execute if the request to web endpoint succeeds

! Note

- You can also directly access the fields in the http response by using `{YourWebEndpointName.FieldName}`. For example: `{UpdateDeviceState.TV}`

6. In **On Failure - Action to execute**, select **Send speech response**

In **Simple editor**, enter `Sorry, {WebEndpointErrorMessage}`.

On Failure

Action to execute *

Send speech response

Simple editor

Template editor

First variation

Sorry, {WebEndpointErrorMessage}

Second variation

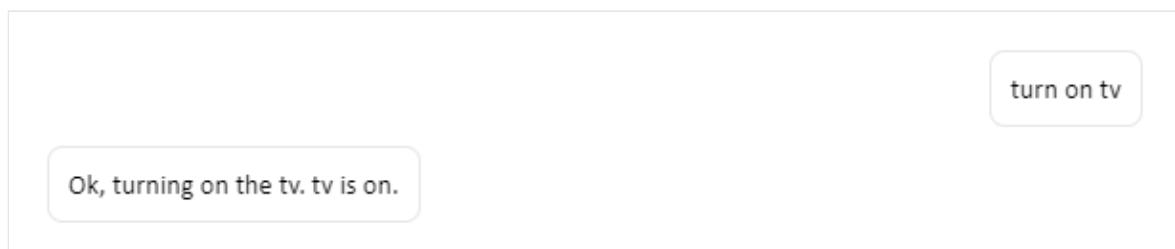
Setting	Suggested value	Description
Action to execute	Send speech response	Action to execute if the request to web endpoint fails

ⓘ Note

- {WebEndpointErrorMessage} is optional. You are free to remove it if you don't want to expose any error message.
- Within our example endpoint, we send back http response with detailed error messages for common errors such as missing header parameters.

Try it out in test portal

- On Success response, save, train and test.



- On Fail response, remove one of the query parameters, save, retrain, and test.

turn on tv

Ok, turning on the tv. Sorry, You need to pass item and value as a query parameter

Integrate with client application

In [Send Custom Commands activity to client application](#), you added a **Send activity to client** action. The activity is sent to the client application whether or not **Call web endpoint** action is successful or not. However, typically you only want to send activity to the client application when the call to the web endpoint is successful. In this example, this is when the device's state is successfully updated.

1. Delete the **Send activity to client** action you previously added.
2. Edit call web endpoint:
 - a. In **Configuration**, make sure **Query Parameters** is `item={SubjectDevice}&&value={OnOff}`
 - b. In **On Success**, change **Action to execute** to **Send activity to client**
 - c. Copy the JSON below to the **Activity Content**

JSON

```
{  
  "type": "event",  
  "name": "UpdateDeviceState",  
  "value": {  
    "state": "{OnOff}",  
    "device": "{SubjectDevice}"  
  }  
}
```

Now you only send activity to the client when the request to the web endpoint is successful.

Create visuals for syncing device state

Add the following XML to `MainPage.xaml` above the **EnableMicrophoneButton** block.

XML

```
<Button x:Name="SyncDeviceStateButton" Content="Sync Device State"  
       Margin="0,10,10,0" Click="SyncDeviceState_ButtonClicked"
```

```
        Height="35"/>
<Button x:Name="EnableMicrophoneButton" .....>
...../>
```

Sync device state

In `MainPage.xaml.cs`, add the reference `using Windows.Web.Http;`. Add the following code to the `MainPage` class. This method will send a GET request to the example endpoint, and extract the current device state for your app. Make sure to change `<your_app_name>` to what you used in the `header` in Custom Command web endpoint.

C#

```
private async void SyncDeviceState_ButtonClicked(object sender,
RoutedEventArgs e)
{
    //Create an HTTP client object
    var httpClient = new HttpClient();

    //Add a user-agent header to the GET request.
    var your_app_name = "<your-app-name>";

    Uri endpoint = new
Uri("https://webendpointexample.azurewebsites.net/api/DeviceState");
    var requestMessage = new HttpRequestMessage(HttpMethod.Get, endpoint);
    requestMessage.Headers.Add("app", $"{your_app_name}");

    try
    {
        //Send the GET request
        var httpResponse = await
httpClient.SendRequestAsync(requestMessage);
        httpResponse.EnsureSuccessStatusCode();
        var httpResponseBody = await
httpResponse.Content.ReadAsStringAsync();
        dynamic deviceState =
JsonConvert.DeserializeObject(httpResponseBody);
        var TVState = deviceState.TV.ToString();
        var FanState = deviceState.Fan.ToString();
        await CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(
            CoreDispatcherPriority.Normal,
            () =>
            {
                State_TV.Text = TVState;
                State_Fan.Text = FanState;
            });
    }
    catch (Exception ex)
    {
        NotifyUser(
            $"Unable to sync device status: {ex.Message}");
    }
}
```

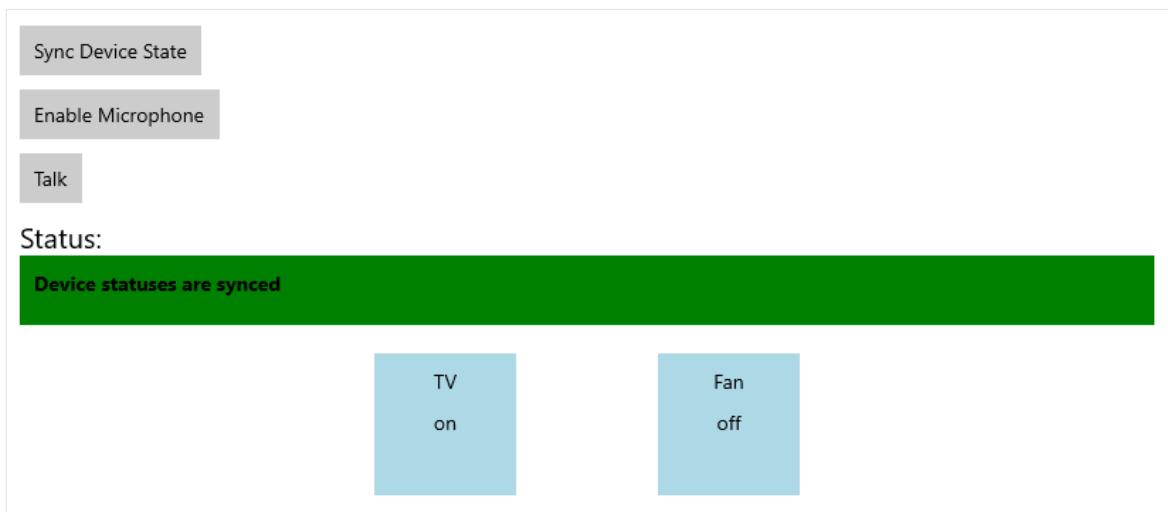
```
    }  
}
```

Try it out

1. Start the application.

2. Click Sync Device State.

If you tested out the app with `turn on tv` in previous section, you would see the TV shows as **on**.



3. Select **Enable microphone**.

4. Select the **Talk** button.

5. Say `turn on the fan`. The visual state of the fan should change to **on**.



Next steps

[Export Custom Commands application as a remote skill](#)

Update a command from a client app

Article • 04/22/2022 • 3 minutes to read

In this article, you'll learn how to update an ongoing command from a client application.

Prerequisites

- ✓ A previously [created Custom Commands app](#)

Update the state of a command

If your client application requires you to update the state of an ongoing command without voice input, you can send an event to update the command.

To illustrate this scenario, send the following event activity to update the state of an ongoing command (`TurnOnOff`):

```
JSON

{
  "type": "event",
  "name": "RemoteUpdate",
  "value": {
    "updatedCommand": {
      "name": "TurnOnOff",
      "updatedParameters": {
        "OnOff": "on"
      },
      "cancel": false
    },
    "updatedGlobalParameters": {},
    "processTurn": true
  }
}
```

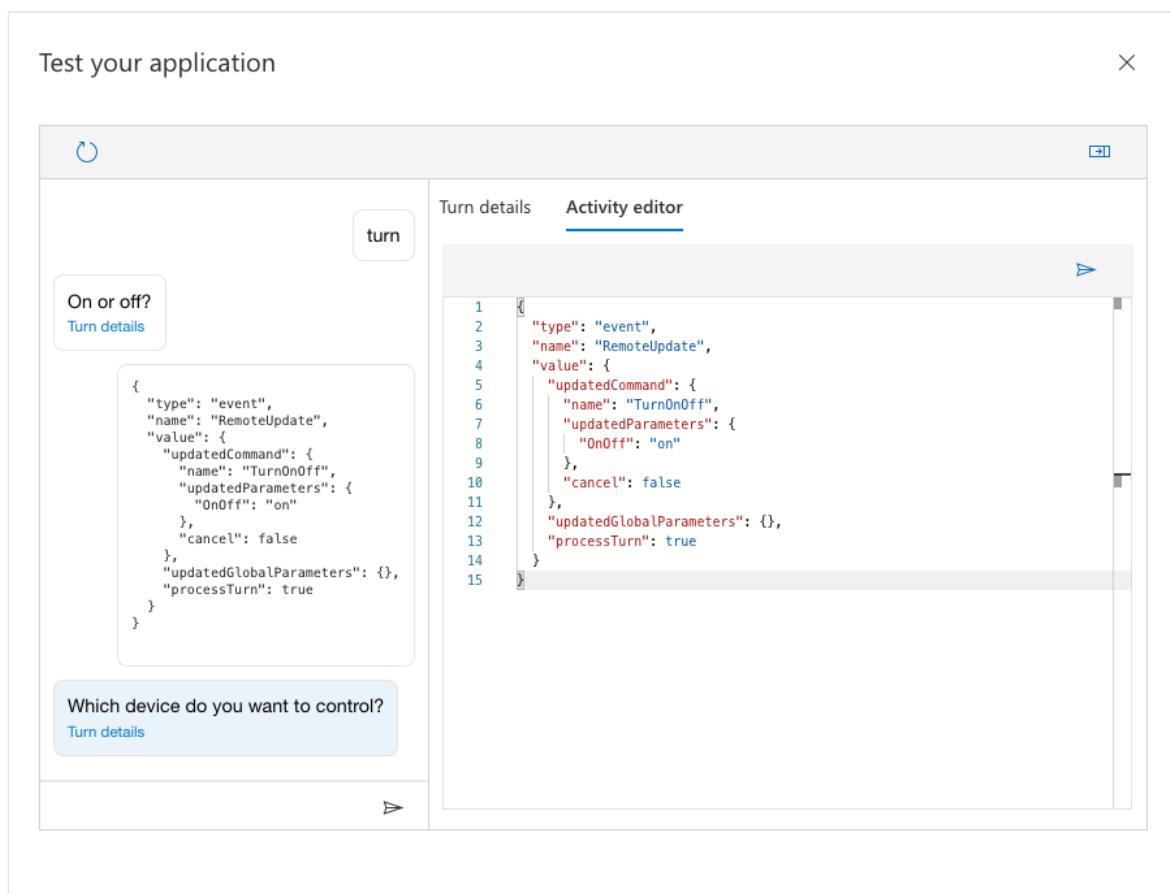
Let's review the key attributes of this activity:

Attribute	Explanation
<code>type</code>	The activity is of type <code>"event"</code> .
<code>name</code>	The name of the event needs to be <code>"RemoteUpdate"</code> .
<code>value</code>	The attribute <code>"value"</code> contains the attributes required to update the current command.

Attribute	Explanation
updatedCommand	The attribute "updatedCommand" contains the name of the command. Within that attribute, "updatedParameters" is a map with the names of the parameters and their updated values.
cancel	If the ongoing command needs to be canceled, set the attribute "cancel" to true.
updatedGlobalParameters	The attribute "updatedGlobalParameters" is a map just like "updatedParameters", but it's used for global parameters.
processTurn	If the turn needs to be processed after the activity is sent, set the attribute "processTurn" to true.

You can test this scenario in the Custom Commands portal:

1. Open the Custom Commands application that you previously created.
2. Select Train and then Test.
3. Send turn.
4. Open the side panel and select Activity editor.
5. Type and send the RemoteCommand event specified in the previous section.



Note how the value for the parameter "OnOff" was set to "on" through an activity from the client instead of speech or text.

Update the catalog of the parameter for a command

When you configure the list of valid options for a parameter, the values for the parameter are defined globally for the application.

In our example, the `SubjectDevice` parameter will have a fixed list of supported values regardless of the conversation.

If you want to add new entries to the parameter's catalog per conversation, you can send the following activity:

JSON

```
{  
  "type": "event",  
  "name": "RemoteUpdate",  
  "value": {  
    "catalogUpdate": {  
      "commandParameterCatalogs": {  
        "TurnOnOff": [  
          {  
            "name": "SubjectDevice",  
            "values": {  
              "stereo": [  
                "cd player"  
              ]  
            }  
          }  
        ]  
      },  
      "processTurn": false  
    }  
  }  
}
```

With this activity, you added an entry for `"stereo"` to the catalog of the parameter `"SubjectDevice"` in the command `"TurnOnOff"`.

Test your application

X

The screenshot shows the Custom Commands application interface. On the left, there is a card titled "Turn details" containing JSON code for a turn-on command. The code includes fields like "type": "event", "name": "RemoteUpdate", and "catalogUpdate". On the right, there is an "Activity editor" window with a code editor showing the same JSON structure. The code editor has line numbers from 1 to 22. The "Activity editor" tab is selected at the top. Below the code editor, there is a "Done, turned on the stereo" message and a "Turn details" button.

```
{  
  "type": "event",  
  "name": "RemoteUpdate",  
  "value": {  
    "catalogUpdate": {  
      "commandParameterCatalogs": {  
        "TurnOnOff": [  
          {  
            "name": "SubjectDevice",  
            "values": {  
              "stereo": [  
                "cd player"  
              ]  
            }  
          }  
        ]  
      },  
      "processTurn": false  
    }  
},  
true  
turn on the stereo  
Done, turned on the stereo  
Turn details  
➡
```

```
1 {  
2   "type": "event",  
3   "name": "RemoteUpdate",  
4   "value": {  
5     "catalogUpdate": {  
6       "commandParameterCatalogs": {  
7         "TurnOnOff": [  
8           {  
9             "name": "SubjectDevice",  
10            "values": {  
11              "stereo": [  
12                "cd player"  
13              ]  
14            }  
15          }  
16        ]  
17      },  
18      "processTurn": false  
19    }  
20  }  
21 }  
22 }
```

Note a couple of things:

- You need to send this activity only once (ideally, right after you start a connection).
- After you send this activity, you should wait for the event `ParameterCatalogsUpdated` to be sent back to the client.

Add more context from the client application

You can set additional context from the client application per conversation that can later be used in your Custom Commands application.

For example, think about the scenario where you want to send the ID and name of the device connected to the Custom Commands application.

To test this scenario, let's create a new command in the current application:

1. Create a new command called `GetDeviceInfo`.
2. Add an example sentence of `get device info`.
3. In the completion rule **Done**, add a **Send speech response** action that contains the attributes of `clientContext`.

Send speech response



Simple response editor

Type ▾

First variation

```
{clientContext.deviceId} - {clientContext.deviceName}
```

Second variation

Cancel

Save

4. Save, train, and test your application.

5. In the testing window, send an activity to update the client context.

JSON

```
{
  "type": "event",
  "name": "RemoteUpdate",
  "value": {
    "clientContext": {
      "deviceId": "12345",
      "deviceName": "My device"
    },
    "processTurn": false
  }
}
```

6. Send the text `get device info`.

Test your application X

Turn details Activity editor

```
1 {  
2   "type": "event",  
3   "name": "RemoteUpdate",  
4   "value": {  
5     "clientContext": {  
6       "deviceId": "12345",  
7       "deviceName": "My device"  
8     },  
9     "processTurn": false  
10   }  
11 }  
12
```

`get device info`

12345 - My device
[Turn details](#)

Note a few things:

- You need to send this activity only once (ideally, right after you start a connection).
- You can use complex objects for `clientContext`.
- You can use `clientContext` in speech responses, for sending activities and for calling web endpoints.

Next steps

[Update a command from a web endpoint](#)

Update a command from a web endpoint

Article • 04/22/2022 • 2 minutes to read

If your client application requires an update to the state of an ongoing command without voice input, you can use a call to a web endpoint to update the command.

In this article, you'll learn how to update an ongoing command from a web endpoint.

Prerequisites

- ✓ A previously [created Custom Commands app](#)

Create an Azure function

For this example, you'll need an HTTP-triggered [Azure function](#) that supports the following input (or a subset of this input):

JSON

```
{  
  "conversationId": "SomeConversationId",  
  "currentCommand": {  
    "name": "SomeCommandName",  
    "parameters": {  
      "SomeParameterName": "SomeParameterValue",  
      "SomeOtherParameterName": "SomeOtherParameterValue"  
    }  
  },  
  "currentGlobalParameters": {  
    "SomeGlobalParameterName": "SomeGlobalParameterValue",  
    "SomeOtherGlobalParameterName": "SomeOtherGlobalParameterValue"  
  }  
}
```

Let's review the key attributes of this input:

Attribute	Explanation
conversationId	The unique identifier of the conversation. Note that this ID can be generated from the client app.
currentCommand	The command that's currently active in the conversation.

Attribute	Explanation
name	The name of the command. The <code>parameters</code> attribute is a map with the current values of the parameters.
currentGlobalParameters	A map like <code>parameters</code> , but used for global parameters.

The output of the Azure function needs to support the following format:

JSON

```
{
  "updatedCommand": {
    "name": "SomeCommandName",
    "updatedParameters": {
      "SomeParameterName": "SomeParameterValue"
    },
    "cancel": false
  },
  "updatedGlobalParameters": {
    "SomeGlobalParameterName": "SomeGlobalParameterValue"
  }
}
```

You might recognize this format because it's the same one that you used when [updating a command from the client](#).

Now, create an Azure function based on Node.js. Copy/paste this code:

Node.js

```
module.exports = async function (context, req) {
  context.log(req.body);
  context.res = {
    body: {
      updatedCommand: {
        name: "IncrementCounter",
        updatedParameters: {
          Counter: req.body.currentCommand.parameters.Counter + 1
        }
      }
    }
  };
}
```

When you call this Azure function from Custom Commands, you'll send the current values of the conversation. You'll return the parameters that you want to update or if you want to cancel the current command.

Update the existing Custom Commands app

Let's hook up the Azure function with the existing Custom Commands app:

1. Add a new command named `IncrementCounter`.
2. Add just one example sentence with the value `increment`.
3. Add a new parameter called `Counter` (same name as specified in the Azure function) of type `Number` with a default value of `0`.
4. Add a new web endpoint called `IncrementEndpoint` with the URL of your Azure function, with **Remote updates** set to **Enabled**.

Web endpoints

A Web endpoint is an external HTTP-based API, or URL, that's used to integrate commands with external systems.

Name *

URL *

Method *

Remote updates

Enabled

5. Create a new interaction rule called `IncrementRule` and add a **Call web endpoint** action.

The screenshot shows the Microsoft Bot Framework Emulator interface. On the left, a sidebar lists navigation options: Add, Example sentences, Parameters, Counter, Completion rules, Done, Interaction rules, and IncrementRule (which is selected and highlighted in blue). The main pane is titled "Interaction rules". It contains a descriptive text about interaction rules and a configuration form. The form includes fields for Name (set to "IncrementRule"), Conditions (with a "Add a condition" button), Actions (with a "Call web endpoint → IncrementEndpoint" section and a "Add an action" button), Expectations (with a "Add an expectation" button), and Post-execution state (set to "Wait for user's input").

Interaction rules

The interaction rules are executed on each interaction with the user. You can use them to add custom business logic to your commands. The rules are executed in the order they are defined.

Name *

IncrementRule

Conditions

Add a condition

Actions

Call web endpoint → IncrementEndpoint

Add an action

Expectations

Add an expectation

Post-execution state

Wait for user's input

6. In the action configuration, select `IncrementEndpoint`. Configure **On success** to **Send speech response** with the value of `Counter`, and configure **On failure** with an error message.

Call web endpoint

Endpoint *

IncrementEndpoint

Additional configuration **On success (required)** On failure (required)

Action to execute *

Send speech response

Simple response editor Type ▾

First variation
 {Counter}

Second variation

Cancel **Save**

The screenshot shows a configuration dialog for a 'Call web endpoint' rule. At the top, there's a header 'Call web endpoint' and a close button 'X'. Below it is a section for 'Endpoint *' with a dropdown menu containing 'IncrementEndpoint'. There are three tabs at the top: 'Additional configuration' (disabled), 'On success (required)' (selected and highlighted in blue), and 'On failure (required)'. Under 'Action to execute *', the dropdown is set to 'Send speech response'. Below this is a 'Simple response editor' with a 'Type' dropdown set to 'Type'. It contains two sections: 'First variation' with the placeholder '{Counter}' and an empty 'Second variation' section.

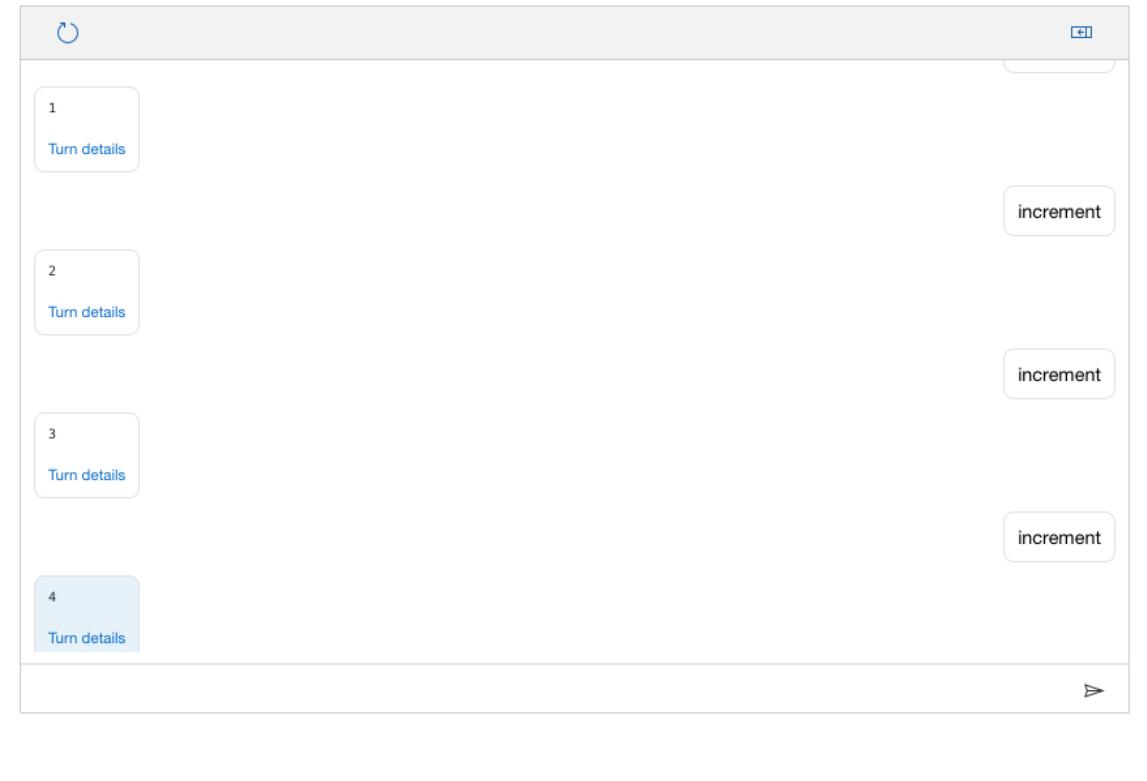
7. Set the post-execution state of the rule to **Wait for user's input**.

Test it

1. Save and train your app.
2. Select **Test**.
3. Send `increment` a few times (which is the example sentence for the `IncrementCounter` command).

Test your application

X



Notice how the Azure function increments the value of the `Counter` parameter on each turn.

Next steps

[Enable a CI/CD process for your Custom Commands application](#)

Continuous Deployment with Azure DevOps

Article • 12/01/2022 • 3 minutes to read

In this article, you learn how to set up continuous deployment for your Custom Commands applications. The scripts to support the CI/CD workflow are provided to you.

Prerequisite

- ✓ A Custom Commands application for development (DEV)
- ✓ A Custom Commands application for production (PROD)
- ✓ Sign up for [Azure Pipelines](#)

Export/Import/Publish

The scripts are hosted at [Cognitive Services Voice Assistant - Custom Commands ↗](#). Clone the scripts in the bash directory to your repository. Make sure you maintain the same path.

Set up a pipeline

1. Go to [Azure DevOps - Pipelines](#) and click "New Pipeline"
2. In **Connect** section, select the location of your repository where these scripts are located
3. In **Select** section, select your repository
4. In **Configure** section, select "Starter pipeline"
5. Next you'll get an editor with a YAML file, replace the "steps" section with this script.

```
YAML

steps:
- task: Bash@3
  displayName: 'Export source app'
  inputs:
    targetType: filePath
    filePath: ./bash/export.sh
    arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a'
```

```
$(SourceAppId) -f ExportedDialogModel.json'
  workingDirectory: bash
  failOnStderr: true

- task: Bash@3
  displayName: 'Import to target app'
  inputs:
    targetType: filePath
    filePath: ./bash/import.sh
    arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a
$(TargetAppId) -f ExportedDialogModel.json'
  workingDirectory: bash
  failOnStderr: true

- task: Bash@3
  displayName: 'Train and Publish target app'
  inputs:
    targetType: filePath
    filePath: './bash/train-and-publish.sh'
    arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a
$(TargetAppId)'
  workingDirectory: bash
  failOnStderr: true
```

6. Note that these scripts assume that you are using the region `westus2`, if that's not the case update the arguments of the tasks accordingly

New pipeline

Review your pipeline YAML

```
◆ TahitiScripts / azure-pipelines.yml * ≡  
6 trigger:  
7 -· master  
8  
9 pool:  
10 |· vmImage: 'ubuntu-latest'  
11  
12 steps:  
    Settings  
13     - task: Bash@3  
14     |· displayName: 'Export source app'  
15     |· inputs:  
16     |     · targetType: filePath  
17     |     · filePath: ./bash/export.sh  
18     |     · arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a $(SourceAppId) -f ExportedDialogModel.json'  
19     |     · workingDirectory: bash  
20     |     · failOnStderr: true  
21  
    Settings  
22     - task: Bash@3  
23     |· displayName: 'Import to target app'  
24     |· inputs:  
25     |     · targetType: filePath  
26     |     · filePath: ./bash/import.sh  
27     |     · arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a $(TargetAppId) -f ExportedDialogModel.json'  
28     |     · workingDirectory: bash  
29     |     · failOnStderr: true  
30  
    Settings  
31     - task: Bash@3  
32     |· displayName: 'Train and Publish target app'  
33     |· inputs:  
34     |     · targetType: filePath  
35     |     · filePath: './bash/train-and-publish.sh'  
36     |     · arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a $(TargetAppId)'  
37     |     · workingDirectory: bash  
38     |     · failOnStderr: true  
39
```

7. In the "Save and run" button, open the dropdown and click "Save"

Hook up the pipeline with your application

1. Navigate to the main page of the pipeline.
2. In the top-right corner dropdown, select **Edit pipeline**. It gets you to a YAML editor.
3. In the top-right corner next to "Run" button, select **Variables**. Click **New variable**.
4. Add these variables:

Variable	Description
SourceAppId	ID of the DEV application
TargetAppId	ID of the PROD application
SubscriptionKey	The key used for both applications
Culture	Culture of the applications (en-us)

[←](#) New variable

Name

Value

- Keep this value secret
- Let users override this value when running this pipeline

To reference a variable in YAML, prefix it with a dollar sign and enclose it in parentheses. For example: `$(SourceAppId)`

To use a variable in a script, use environment variable syntax.

Replace `.` and space with `_`, capitalize the letters, and then use your platform's syntax for referencing an environment variable. Examples:

Batch script: `%SOURCEAPPID%`

PowerShell script: `$env:SOURCEAPPID`

Bash script: `$SOURCEAPPID`

5. Click "Run" and then click in the "Job" running.

You should see a list of tasks running that contains: "Export source app", "Import to target app" & "Train and Publish target app"

Deploy from source code

In case you want to keep the definition of your application in a repository, we provide the scripts for deployments from source code. Since the scripts are in bash, If you are using Windows you'll need to install the [Linux subsystem](#).

The scripts are hosted at [Cognitive Services Voice Assistant - Custom Commands ↗](#). Clone the scripts in the bash directory to your repository. Make sure you maintain the same path.

Prepare your repository

1. Create a directory for your application, in our example create one called "apps".

2. Update the arguments of the bash script below, and run. It will import the dialog model of your application to the file myapp.json

BASH

```
bash/export.sh -r <region> -s <subscriptionkey> -c en-us -a <appid> -f  
apps/myapp.json
```

Arguments	Description
region	Your Speech resource region. For example: <code>westus2</code>
subscriptionkey	Your Speech resource key.
appid	the Custom Commands' application ID you want to export.

3. Push these changes to your repository.

Set up a pipeline

1. Go to [Azure DevOps - Pipelines](#) and click "New Pipeline"
2. In **Connect** section, select the location of your repository where these scripts are located
3. In **Select** section, select your repository
4. In **Configure** section, select "Starter pipeline"
5. Next you'll get an editor with a YAML file, replace the "steps" section with this script.

YAML

```
steps:  
- task: Bash@3  
  displayName: 'Import app'  
  inputs:  
    targetType: filePath  
    filePath: ./bash/import.sh  
    arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a  
$(TargetAppId) -f ..//apps/myapp.json'  
    workingDirectory: bash  
    failOnStderr: true  
  
- task: Bash@3  
  displayName: 'Train and Publish app'  
  inputs:
```

```
targetType: filePath
filePath: './bash/train-and-publish.sh'
arguments: '-r westus2 -s $(SubscriptionKey) -c $(Culture) -a
$(TargetAppId)'
workingDirectory: bash
failOnStderr: true
```

ⓘ Note

these scripts assume that you are using the region westus2, if that's not the case update the arguments of the tasks accordingly

6. In the "Save and run" button, open the dropdown and click "Save"

Hook up the pipeline with your target applications

1. Navigate to the main page of the pipeline.
2. In the top-right corner dropdown, select **Edit pipeline**. It gets you to a YAML editor.
3. In the top-right corner next to "Run" button, select **Variables**. Click **New variable**.
4. Add these variables:

Variable	Description
TargetAppId	ID of the PROD application
SubscriptionKey	The key used for both applications
Culture	Culture of the applications (en-us)

5. Click "Run" and then click in the "Job" running. You should see a list of tasks running that contains: "Import app" & "Train and Publish app"

Next steps

[See samples on GitHub](#)

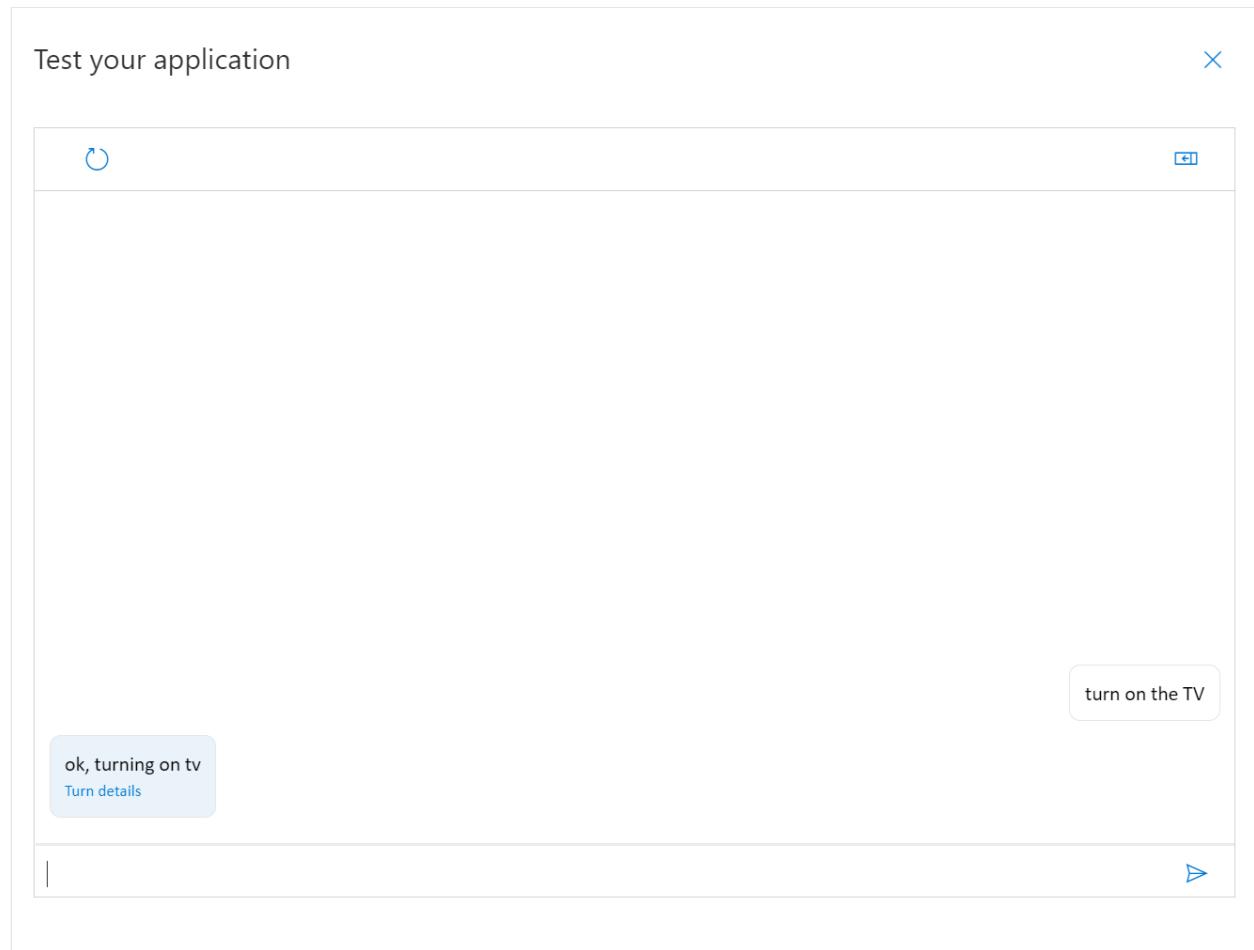
Test your Custom Commands Application

Article • 04/22/2022 • 2 minutes to read

In this article, you learn different approaches to testing a custom commands application.

Test in the portal

Test in the portal is the simplest and quickest way to check if your custom command application work as expected. After the app is successfully trained, click `Test` button to start testing.

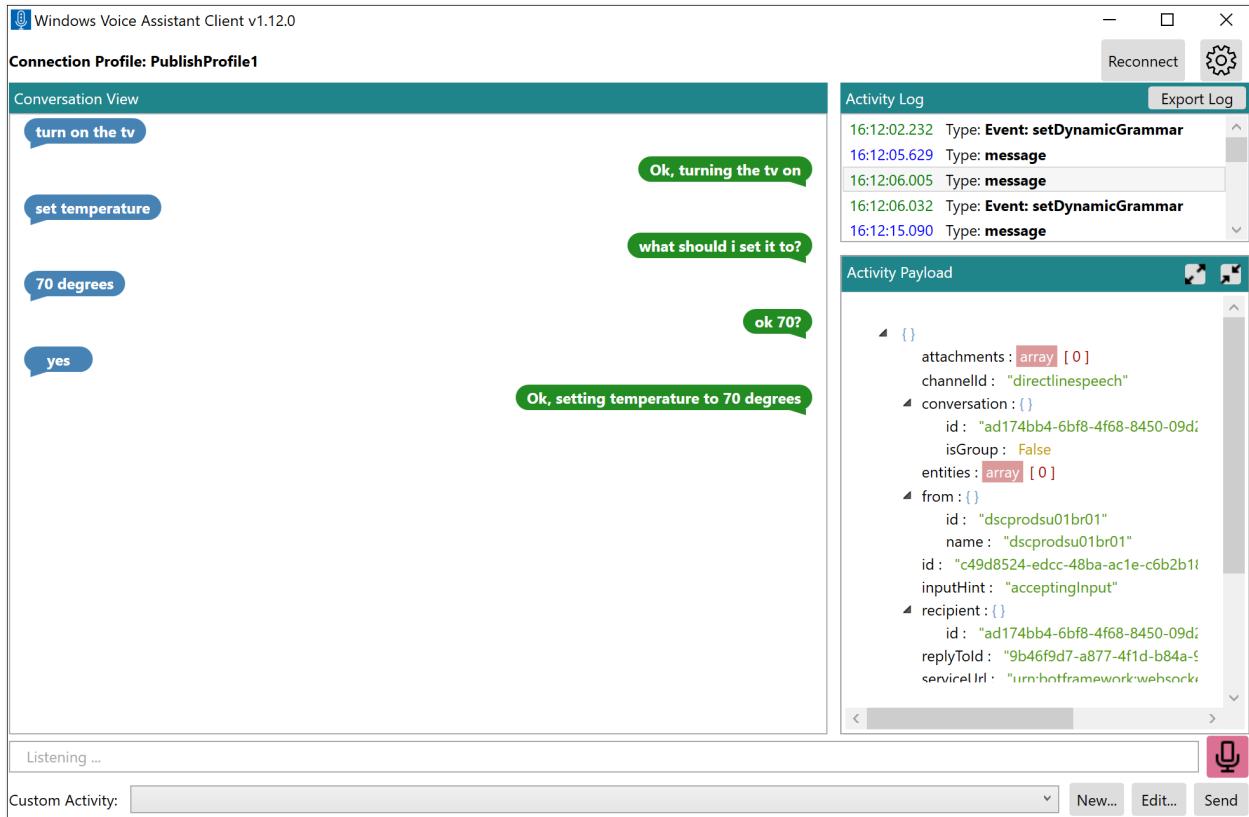


Test with Windows Voice Assistant Client

The Windows Voice Assistant Client is a Windows Presentation Foundation (WPF) application in C# that makes it easy to test interactions with your bot before creating a custom client application.

The tool can accept a custom command application ID. It allows you to test your task completion or command-and-control scenario hosted on the Custom Commands service.

To set up the client, checkout [Windows Voice Assistant Client](#).



Test programmatically with the Cognitive Services Voice Assistant Test Tool

The Voice Assistant Test Tool is a configurable .NET Core C# console application for end-to-end functional regression tests for your Microsoft Voice Assistant.

The tool can run manually as a console command or automated as part of an Azure DevOps CI/CD pipeline to prevent regressions in your bot.

To learn how to set up the tool, see [Voice Assistant Test Tool](#).

Test with Speech SDK-enabled client applications

The Speech software development kit (SDK) exposes many of the Speech service capabilities, which allows you to develop speech-enabled applications. It's available in many programming languages on most platforms.

To set up a Universal Windows Platform (UWP) client application with Speech SDK, and integrate it with your custom command application:

- [How to: Integrate with a client application using Speech SDK](#)
- [How to: Send activity to client application](#)
- [How to: Set up web endpoints](#)

For other programming languages and platforms:

- [Speech SDK programming languages, platforms, scenario capacities](#)
- [Voice assistant sample codes ↗](#)

Next steps

[See samples on GitHub](#)

Debug errors when authoring a Custom Commands application

Article • 04/22/2022 • 2 minutes to read

This article describes how to debug when you see errors while building Custom Commands application.

Errors when creating an application

Custom Commands also creates an application in [LUIS](#) when creating a Custom Commands application.

[LUIS limits 500 applications per authoring resource](#). Creation of LUIS application could fail if you are using an authoring resource that already has 500 applications.

Make sure the selected LUIS authoring resource has less than 500 applications. If not, you can create new LUIS authoring resource, switch to another one, or try to clean up your LUIS applications.

Errors when deleting an application

Can't delete LUIS application

When deleting a Custom Commands application, Custom Commands may also try to delete the LUIS application associated with the Custom Commands application.

If the deletion of LUIS application failed, go to your [LUIS](#) account to delete them manually.

TooManyRequests

When you try to delete large number of applications all at once, it's likely you would see 'TooManyRequests' errors. These errors mean your deletion requests get throttled by Azure.

Refresh your page and try to delete fewer applications.

Errors when modifying an application

Can't delete a parameter or a Web Endpoint

You are not allowed to delete a parameter when it is being used. Remove any reference of the parameter in any speech responses, sample sentences, conditions, actions, and try again.

Can't delete a Web Endpoint

You are not allowed to delete a Web Endpoint when it is being used. Remove any **Call Web Endpoint** action that uses this Web Endpoint before removing a Web Endpoint.

Errors when training an application

Built-In intents

LUIS has built-in Yes/No intents. Having sample sentences with only "yes", "no" would fail the training.

Keyword	Variations
Yes	Sure, OK
No	Nope, Not

Common sample sentences

Custom Commands does not allow common sample sentences shared among different commands. The training of an application could fail if some sample sentences in one command are already defined in another command.

Make sure you don't have common sample sentences shared among different commands.

For best practice of balancing your sample sentences across different commands, refer [LUIS best practice](#).

Empty sample sentences

You need to have at least one sample sentence for each Command.

Undefined parameter in sample sentences

One or more parameters are used in the sample sentences but not defined.

Training takes too long

LUIS training is meant to learn quickly with fewer examples. Don't add too many example sentences.

If you have many example sentences that are similar, define a parameter, abstract them into a pattern and add it to Example Sentences.

For example, you can define a parameter {vehicle} for the example sentences below, and only add "Book a {vehicle}" to Example Sentences.

Example sentences	Pattern
Book a car	Book a {vehicle}
Book a flight	Book a {vehicle}
Book a taxi	Book a {vehicle}

For best practice of LUIS training, refer [LUIS best practice](#).

Can't update LUIS key

Reassign to E0 authoring resource

LUIS does not support reassigning LUIS application to E0 authoring resource.

If you need to change your authoring resource from F0 to E0, or change to a different E0 resource, recreate the application.

For quickly export an existing application and import it into a new application, refer to [Continuous Deployment with Azure DevOps](#).

Save button is disabled

If you never assign a LUIS prediction resource to your application, the Save button would be disabled when you try to change your authoring resource without adding a prediction resource.

Next steps

[See samples on GitHub](#)

Troubleshoot a Custom Commands application at runtime

Article • 04/22/2022 • 3 minutes to read

This article describes how to debug when you see errors while running Custom Commands application.

Connection failed

If you run Custom Commands application from [client application \(with Speech SDK\)](#) or [Windows Voice Assistant Client](#), you may experience connection errors as listed below:

Error code	Details
401	AuthenticationFailure: WebSocket Upgrade failed with an authentication error
1002	The server returned status code '404' when status code '101' was expected.

Error 401

- The region specified in client application does not match with the region of the custom command application
- Speech resource Key is invalid

Make sure your speech resource key is correct.

Error 1002

- Your custom command application is not published

Publish your application in the portal.

- Your custom command applicationId is not valid

Make sure your custom command application ID is correct. custom command application outside your speech resource

Make sure the custom command application is created under your speech resource.

For more information on troubleshooting the connection issues, reference [Windows Voice Assistant Client Troubleshooting](#)

Dialog is canceled

When running your Custom Commands application, the dialog would be canceled when some errors occur.

- If you are testing the application in the portal, it would directly display the cancellation description and play out an error earcon.
- If you are running the application with [Windows Voice Assistant Client](#), it would play out an error earcon. You can find the **Event: CancelledDialog** under the **Activity Logs**.
- If you are following our client application example [client application \(with Speech SDK\)](#), it would play out an error earcon. You can find the **Event: CancelledDialog** under the **Status**.
- If you are building your own client application, you can always design your desired logics to handle the **CancelledDialog** events.

The **CancelledDialog** event consists of cancellation code and description, as listed below:

Cancellation Code	Cancellation Description
MaxTurnThresholdReached	No progress was made after the max number of turns allowed
RecognizerQuotaExceeded	Recognizer usage quota exceeded
RecognizerConnectionFailed	Connection to the recognizer failed
RecognizerUnauthorized	This application cannot be accessed with the current subscription
RecognizerInputExceededAllowedLength	Input exceeds the maximum supported length for the recognizer
RecognizerNotFound	Recognizer not found
RecognizerInvalidQuery	Invalid query for the recognizer
RecognizerError	Recognizer returns an error

No progress was made after the max number of turns allowed

The dialog is canceled when a required slot is not successfully updated after certain number of turns. The build-in max number is 3.

Recognizer usage quota exceeded

Language Understanding (LUIS) has limits on resource usage. Usually "Recognizer usage quota exceeded error" can be caused by:

- Your LUIS authoring exceeds the limit

Add a prediction resource to your Custom Commands application:

1. go to **Settings**, LUIS resource
2. Choose a prediction resource from **Prediction resource**, or click **Create new resource**

- Your LUIS prediction resource exceeds the limit

If you are on a F0 prediction resource, it has limit of 10 thousand/month, 5 queries/second.

For more details on LUIS resource limits, refer [Language Understanding resource usage and limit](#)

Connection to the recognizer failed

Usually it means transient connection failure to Language Understanding (LUIS) recognizer. Try it again and the issue should be resolved.

This application cannot be accessed with the current subscription

Your subscription is not authorized to access the LUIS application.

Input exceeds the maximum supported length

Your input has exceeded 500 characters. We only allow at most 500 characters for input utterance.

Invalid query for the recognizer

Your input has exceeded 500 characters. We only allow at most 500 characters for input utterance.

Recognizer return an error

The LUIS recognizer returned an error when trying to recognize your input.

Recognizer not found

Cannot find the recognizer type specified in your custom commands dialog model.

Currently, we only support [Language Understanding \(LUIS\) Recognizer](#).

Other common errors

Unexpected response

Unexpected responses may be caused multiple things. A few checks to start with:

- Yes/No Intents in example sentences

As we currently don't support Yes/No Intents except when using with confirmation feature. All the Yes/No Intents defined in example sentences would not be detected.

- Similar intents and examples sentences among commands

The LUIS recognition accuracy may get affected when two commands share similar intents and examples sentences. You can try to make commands functionality and example sentences as distinct as possible.

For best practice of improving recognition accuracy, refer [LUIS best practice](#).

- Dialog is canceled

Check the reasons of cancellation in the section above.

Error while rendering the template

An undefined parameter is used in the speech response.

Object reference not set to an instance of an object

You have an empty parameter in the JSON payload defined in **Send Activity to Client** action.

Next steps

[See samples on GitHub](#)

Custom Commands concepts and definitions

Article • 04/22/2022 • 5 minutes to read

This article serves as a reference for concepts and definitions for Custom Commands applications.

Commands configuration

Commands are the basic building blocks of a Custom Commands application. A command is a set of configurations required to complete a specific task defined by a user.

Example sentences

Example utterances are the set examples the user can say to trigger a particular command. You need to provide only a sample of utterances and not an exhaustive list.

Parameters

Parameters are information required by the commands to complete a task. In complex scenarios, parameters can also be used to define conditions that trigger custom actions.

Completion rules

Completion rules are a series of rules to be executed after the command is ready to be fulfilled, for example, when all the conditions of the rules are satisfied.

Interaction rules

Interaction rules are additional rules to handle more specific or complex situations. You can add additional validations or configure advanced features such as confirmations or a one-step correction. You can also build your own custom interaction rules.

Parameters configuration

Parameters are information required by commands to complete a task. In complex scenarios, parameters can also be used to define conditions that trigger custom actions.

Name

A parameter is identified by the name property. You should always give a descriptive name to a parameter. A parameter can be referred across different sections, for example, when you construct conditions, speech responses, or other actions.

Required

This check box indicates whether a value for this parameter is required for command fulfillment or completion. You must configure responses to prompt the user to provide a value if a parameter is marked as required.

Note that, if you configured a **required** parameter to have a **Default value**, the system will still explicitly prompt for the parameter's value.

Type

Custom Commands supports the following parameter types:

- Age
- Currency
- DateTime
- Dimension
- Email
- Geography
- Number
- Ordinal
- Percentage
- PersonName
- PhoneNumber
- String
- Temperature
- Url

Every locale supports the "String" parameter type, but availability of all other types differs by locale. Custom Commands uses LUIS's prebuilt entity resolution, so the availability of a parameter type in a locale depends on LUIS's prebuilt entity support in that locale. You can find [more details on LUIS's prebuilt entity support per locale](#).

Custom LUIS entities (such as machine learned entities) are currently not supported.

Some parameter types like Number, String and DateTime support default value configuration, which you can configure from the portal.

Configuration

Configuration is a parameter property defined only for the type String. The following values are supported:

- **None.**
- **Accept full input:** When enabled, a parameter accepts any input utterance. This option is useful when the user needs a parameter with the full utterance. An example is postal addresses.
- **Accept predefined input values from an external catalog:** This value is used to configure a parameter that can assume a wide variety of values. An example is a sales catalog. In this case, the catalog is hosted on an external web endpoint and can be configured independently.
- **Accept predefined input values from internal catalog:** This value is used to configure a parameter that can assume a few values. In this case, values must be configured in the Speech Studio.

Validation

Validations are constructs applicable to certain parameter types that let you configure constraints on a parameter's value. Currently, Custom Commands supports validations on the following parameter types:

- DateTime
- Number

Rules configuration

A rule in Custom Commands is defined by a set of *conditions* that, when met, execute a set of *actions*. Rules also let you configure *post-execution state* and *expectations* for the next turn.

Types

Custom Commands supports the following rule categories:

- **Completion rules:** These rules must be executed upon command fulfillment. All the rules configured in this section for which the conditions are true will be executed.
- **Interaction rules:** These rules can be used to configure additional custom validations, confirmations, and a one-step correction, or to accomplish any other

custom dialog logic. Interaction rules are evaluated at each turn in the processing and can be used to trigger completion rules.

The different actions configured as part of a rule are executed in the order in which they appear in the authoring portal.

Conditions

Conditions are the requirements that must be met for a rule to execute. Rules conditions can be of the following types:

- **Parameter value equals:** The configured parameter's value equals a specific value.
- **No parameter value:** The configured parameters shouldn't have any value.
- **Required parameters:** The configured parameter has a value.
- **All required parameters:** All the parameters that were marked as required have a value.
- **Updated parameters:** One or more parameter values were updated as a result of processing the current input (utterance or activity).
- **Confirmation was successful:** The input utterance or activity was a successful confirmation (yes).
- **Confirmation was denied:** The input utterance or activity was not a successful confirmation (no).
- **Previous command needs to be updated:** This condition is used in instances when you want to catch a negated confirmation along with an update. Behind the scenes, this condition is configured for when the dialog engine detects a negative confirmation where the intent is the same as the previous turn, and the user has responded with an update.

Actions

- **Send speech response:** Send a speech response back to the client.
- **Update parameter value:** Update the value of a command parameter to a specified value.
- **Clear parameter value:** Clear the command parameter value.
- **Call web endpoint:** Make a call to a web endpoint.
- **Send activity to client:** Send a custom activity to the client.

Expectations

Expectations are used to configure hints for the processing of the next user input. The following types are supported:

- **Expecting confirmation from user:** This expectation specifies that the application is expecting a confirmation (yes/no) for the next user input.
- **Expecting parameter(s) input from user:** This expectation specifies one or more command parameters that the application is expecting from the user input.

Post-execution state

The post-execution state is the dialog state after processing the current input (utterance or activity). It's of the following types:

- **Keep current state:** Keep current state only.
- **Complete the command:** Complete the command and no additional rules of the command will be processed.
- **Execute completion rules:** Execute all the valid completion rules.
- **Wait for user's input:** Wait for the next user input.

Next steps

[See samples on GitHub](#)

Custom Commands encryption of data at rest

Article • 04/22/2022 • 6 minutes to read

Custom Commands automatically encrypts your data when it is persisted to the cloud. The Custom Commands service encryption protects your data and to help you to meet your organizational security and compliance commitments.

ⓘ Note

Custom Commands service doesn't automatically enable encryption for the LUIS resources associated with your application. If needed, you must enable encryption for your LUIS resource from [here](#).

About Cognitive Services encryption

Data is encrypted and decrypted using [FIPS 140-2](#) compliant [256-bit AES](#) encryption. Encryption and decryption are transparent, meaning encryption and access are managed for you. Your data is secure by default and you don't need to modify your code or applications to take advantage of encryption.

About encryption key management

When you use Custom Commands, speech service will store following data in the cloud:

- Configuration JSON behind the Custom Commands application
- LUIS authoring and prediction key

By default, your subscription uses Microsoft-managed encryption keys. However, you can also manage your subscription with your own encryption keys. Customer-managed keys (CMK), also known as bring your own key (BYOK), offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data.

ⓘ Important

Customer-managed keys are only available for resources created after 27 June, 2020. To use CMK with Speech Services, you will need to create a new Speech resource.

Once the resource is created, you can use Azure Key Vault to set up your managed identity.

To request the ability to use customer-managed keys, fill out and submit Customer-Managed Key Request Form. It will take approximately 3-5 business days to hear back on the status of your request. Depending on demand, you may be placed in a queue and approved as space becomes available. Once approved for using CMK with Speech Services, you'll need to create a new Speech resource from the Azure portal.

 **Note**

Customer-managed keys (CMK) are supported only for Custom Commands.

Custom Speech and Custom Voice still support only Bring Your Own Storage (BYOS). [Learn more](#)

If you're using the given speech resource for accessing these service, compliance needs must be met by explicitly configuring BYOS.

Customer-managed keys with Azure Key Vault

You must use Azure Key Vault to store customer-managed keys. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The Speech resource and the key vault must be in the same region and in the same Azure Active Directory (Azure AD) tenant, but they can be in different subscriptions. For more information about Azure Key Vault, see [What is Azure Key Vault?](#).

When a new Speech resource is created and used to provision Custom Commands application - data is always encrypted using Microsoft-managed keys. It's not possible to enable customer-managed keys at the time that the resource is created. Customer-managed keys are stored in Azure Key Vault, and the key vault must be provisioned with access policies that grant key permissions to the managed identity that is associated with the Cognitive Services resource. The managed identity is available only after the resource is created using the Pricing Tier required for CMK.

Enabling customer managed keys will also enable a system assigned [managed identity](#), a feature of Azure AD. Once the system assigned managed identity is enabled, this resource will be registered with Azure Active Directory. After being registered, the managed identity will be given access to the Key Vault selected during customer managed key setup.

ⓘ Important

If you disable system assigned managed identities, access to the key vault will be removed and any data encrypted with the customer keys will no longer be accessible. Any features depended on this data will stop working.

ⓘ Important

Managed identities do not currently support cross-directory scenarios. When you configure customer-managed keys in the Azure portal, a managed identity is automatically assigned under the covers. If you subsequently move the subscription, resource group, or resource from one Azure AD directory to another, the managed identity associated with the resource is not transferred to the new tenant, so customer-managed keys may no longer work. For more information, see [Transferring a subscription between Azure AD directories](#) in [FAQs and known issues with managed identities for Azure resources](#).

Configure Azure Key Vault

Using customer-managed keys requires that two properties be set in the key vault, **Soft Delete** and **Do Not Purge**. These properties are not enabled by default, but can be enabled using either PowerShell or Azure CLI on a new or existing key vault.

ⓘ Important

If you do not have the **Soft Delete** and **Do Not Purge** properties enabled and you delete your key, you won't be able to recover the data in your Cognitive Service resource.

To learn how to enable these properties on an existing key vault, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in one of the following articles:

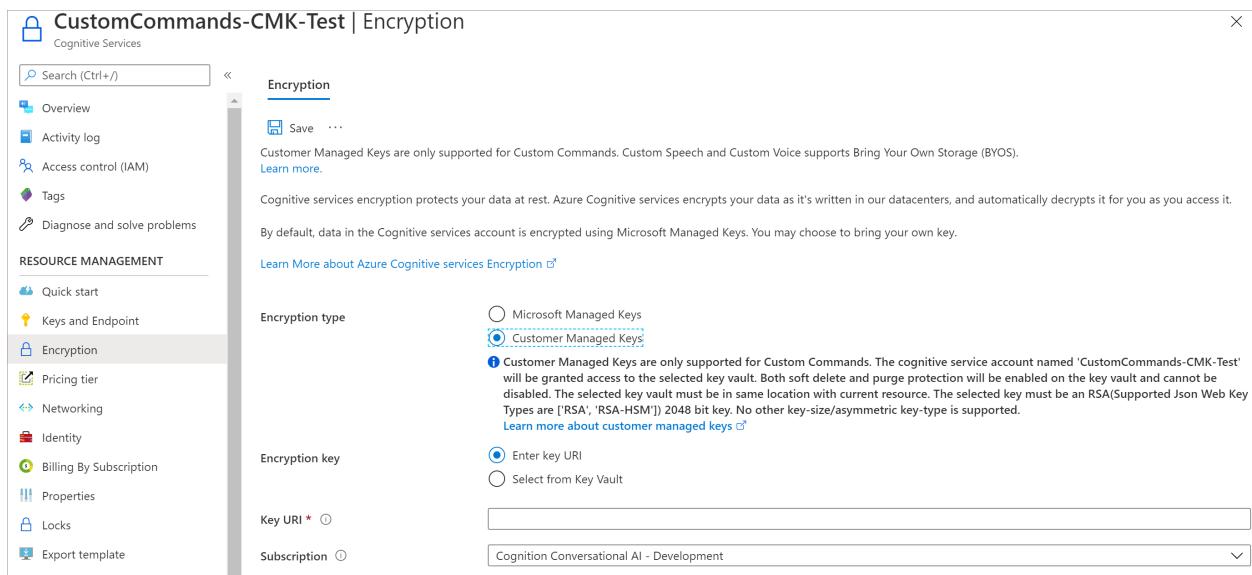
- [How to use soft-delete with PowerShell](#).
- [How to use soft-delete with CLI](#).

Only RSA keys of size 2048 are supported with Azure Storage encryption. For more information about keys, see **Key Vault keys** in [About Azure Key Vault keys, secrets and certificates](#).

Enable customer-managed keys for your Speech resource

To enable customer-managed keys in the Azure portal, follow these steps:

1. Navigate to your Speech resource.
2. On the **Settings** blade for your Speech resource, click **Encryption**. Select the **Customer Managed Keys** option, as shown in the following figure.



Specify a key

After you enable customer-managed keys, you'll have the opportunity to specify a key to associate with the Cognitive Services resource.

Specify a key as a URI

To specify a key as a URI, follow these steps:

1. To locate the key URI in the Azure portal, navigate to your key vault, and select the **Keys** setting. Select the desired key, then click the key to view its versions. Select a key version to view the settings for that version.
2. Copy the value of the **Key Identifier** field, which provides the URI.

The screenshot shows the Azure Key Vault 'Keys' blade. At the top, it displays the key identifier: 17bf9182bb694f109b8dc6d1e9b69f29. Below this, there are sections for 'Properties', 'Key Identifier' (containing '<key-uri>'), 'Settings' (with checkboxes for activation and expiration dates), 'Enabled?' (set to 'Yes'), and 'Tags' (0 tags). Under 'Permitted operations', all six checkboxes are selected: Encrypt, Sign, Wrap Key, Decrypt, Verify, and Unwrap Key.

3. In the **Encryption** settings for your Speech service, choose the **Enter key URI** option.
4. Paste the URI that you copied into the **Key URI** field.
5. Specify the subscription that contains the key vault.
6. Save your changes.

Specify a key from a key vault

To specify a key from a key vault, first make sure that you have a key vault that contains a key. To specify a key from a key vault, follow these steps:

1. Choose the **Select from Key Vault** option.
2. Select the key vault containing the key you want to use.
3. Select the key from the key vault.

Select key from Azure Key Vault

Subscription *	Cognition Conversational AI - Development
Key vault *	CMKTest-CustomCommands Create new
Key *	CMKTest-CustomCommands Create new
Version * ⓘ	cd990f81585d49a7ae3a18d91e87caa0 Create new

4. Save your changes.

Update the key version

When you create a new version of a key, update the Speech resource to use the new version. Follow these steps:

1. Navigate to your Speech resource and display the **Encryption** settings.
2. Enter the URI for the new key version. Alternately, you can select the key vault and the key again to update the version.
3. Save your changes.

Use a different key

To change the key used for encryption, follow these steps:

1. Navigate to your Speech resource and display the **Encryption** settings.
2. Enter the URI for the new key. Alternately, you can select the key vault and choose a new key.
3. Save your changes.

Rotate customer-managed keys

You can rotate a customer-managed key in Azure Key Vault according to your compliance policies. When the key is rotated, you must update the Speech resource to use the new key URI. To learn how to update the resource to use a new version of the key in the Azure portal, see [Update the key version](#).

Rotating the key does not trigger re-encryption of data in the resource. There is no further action required from the user.

Revoke access to customer-managed keys

To revoke access to customer-managed keys, use PowerShell or Azure CLI. For more information, see [Azure Key Vault PowerShell](#) or [Azure Key Vault CLI](#). Revoking access effectively blocks access to all data in the Cognitive Services resource, as the encryption key is inaccessible by Cognitive Services.

Disable customer-managed keys

When you disable customer-managed keys, your Speech resource is then encrypted with Microsoft-managed keys. To disable customer-managed keys, follow these steps:

1. Navigate to your Speech resource and display the **Encryption** settings.
2. Deselect the checkbox next to the **Use your own key** setting.

Next steps

- [Speech Customer-Managed Key Request Form](#)
- [Learn more about Azure Key Vault](#)
- [What are managed identities](#)

What is Direct Line Speech?

Article • 01/11/2023 • 2 minutes to read

Direct Line Speech is a robust, end-to-end solution for creating a flexible, extensible voice assistant. It is powered by the Bot Framework and its Direct Line Speech channel, that is optimized for voice-in, voice-out interaction with bots.

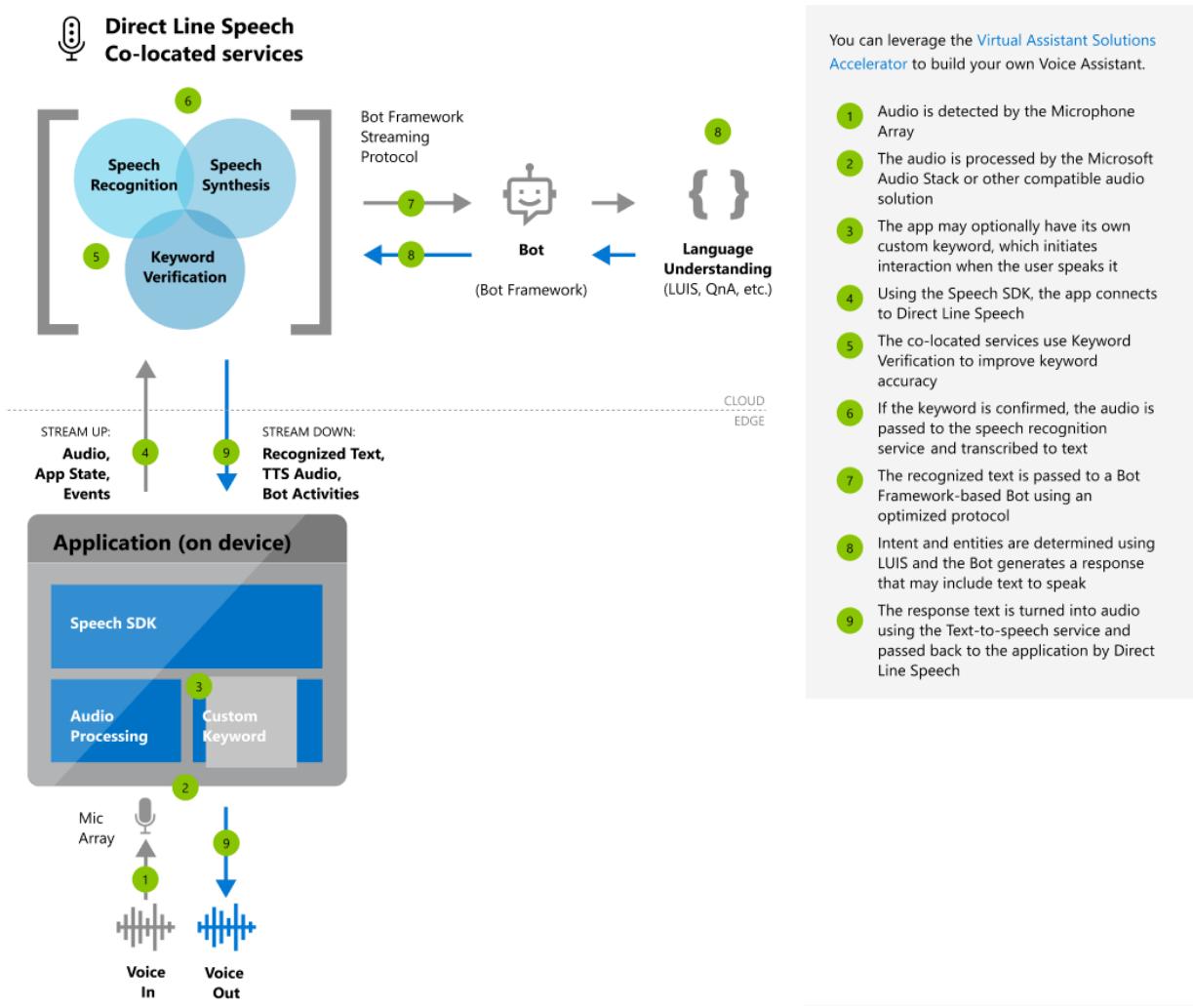
Voice assistants listen to users and take an action in response, often speaking back. They use speech-to-text to transcribe the user's speech, then take action on the natural language understanding of the text. This action frequently includes spoken output from the assistant generated with text-to-speech.

Direct Line Speech offers the highest levels of customization and sophistication for voice assistants. It's designed for conversational scenarios that are open-ended, natural, or hybrids of the two 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](#) for a streamlined solution experience.

Direct Line Speech supports these locales: ar-eg, ar-sa, ca-es, da-dk, de-de, en-au, en-ca, en-gb, en-in, en-nz, en-us, es-es, es-mx, fi-fi, fr-ca, fr-fr, gu-in, hi-in, hu-hu, it-it, ja-jp, ko-kr, mr-in, nb-no, nl-nl, pl-pl, pt-br, pt-pt, ru-ru, sv-se, ta-in, te-in, th-th, tr-tr, zh-cn, zh-hk, and zh-tw.

Getting started with Direct Line Speech

To create a voice assistant using Direct Line Speech, create a Speech resource and Azure Bot resource in the [Azure portal](#). Then [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, 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 and learning the APIs quickly. This table includes a list of voice assistant quickstarts organized by language and platform.

Quickstart	Platform	API reference
C#, UWP	Windows	Browse
Java	Windows, macOS, Linux	Browse
Java	Android	Browse

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 to get started quickly.

Reference docs

- [Speech SDK](#)
- [Azure Bot Service](#)

Next steps

- [Get the Speech SDK](#)
- [Create and deploy a basic bot](#)
- [Get the Virtual Assistant Solution and Enterprise Template ↗](#)

Quickstart: Create a custom voice assistant

Article • 04/22/2022 • 24 minutes to read

In this quickstart, you will use the [Speech SDK](#) to create a custom voice assistant application that connects to a bot that you have already authored and configured. If you need to create a bot, see [the related tutorial](#) for a more comprehensive guide.

After satisfying a few prerequisites, connecting your custom voice assistant takes only a few steps:

- ✓ Create a `BotFrameworkConfig` object from your subscription key and region.
- ✓ Create a `DialogServiceConnector` object using the `BotFrameworkConfig` object from above.
- ✓ Using the `DialogServiceConnector` object, start the listening process for a single utterance.
- ✓ Inspect the `ActivityReceivedEventArgs` returned.

ⓘ Note

The Speech SDK for C++, JavaScript, Objective-C, Python, and Swift support custom voice assistants, but we haven't yet included a guide here.

You can view or download all [Speech SDK C# Samples](#) ↗ on GitHub.

Prerequisites

Before you get started, make sure to:

- ✓ [Create a Speech resource](#)
- ✓ [Set up your development environment and create an empty project](#)
- ✓ Create a bot connected to the Direct Line Speech channel
- ✓ Make sure that you have access to a microphone for audio capture

ⓘ Note

Please refer to [the list of supported regions for voice assistants](#) and ensure your resources are deployed in one of those regions.

Open your project in Visual Studio

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

Start with some boilerplate code

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

1. In **Solution Explorer**, open `MainPage.xaml`.
2. In the designer's XAML view, replace the entire contents with the following snippet that defines a rudimentary user interface:

```
XML

<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/expressionblend/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="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>
</StackPanel>
<MediaElement x:Name="mediaElement"/>
</Grid>
</Page>

```

The Design view is updated to show the application's user interface.

1. In **Solution Explorer**, open the code-behind source file `MainPage.xaml.cs`. (It's grouped under `MainPage.xaml`.) Replace the contents of this file with the below, which 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

C#

```

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.FromEle-
        ment(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
}
}
}

```

1. Add the following code snippet to the method body of

`InitializeDialogServiceConnector`. This code creates the `DialogServiceConnector` with your subscription information.

C#

```

// Create a BotFrameworkConfig by providing a Speech service
subscription key
// the botConfig.Language 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.Language = "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, see the Bot Framework documentation for the Direct Line Speech channel.

2. Replace the strings `YourSpeechSubscriptionKey` and `YourServiceRegion` with your own values for your speech subscription and `region`.
3. 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.

C#

```
// 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}");
};

```

4. 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.

```

C#
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);
}

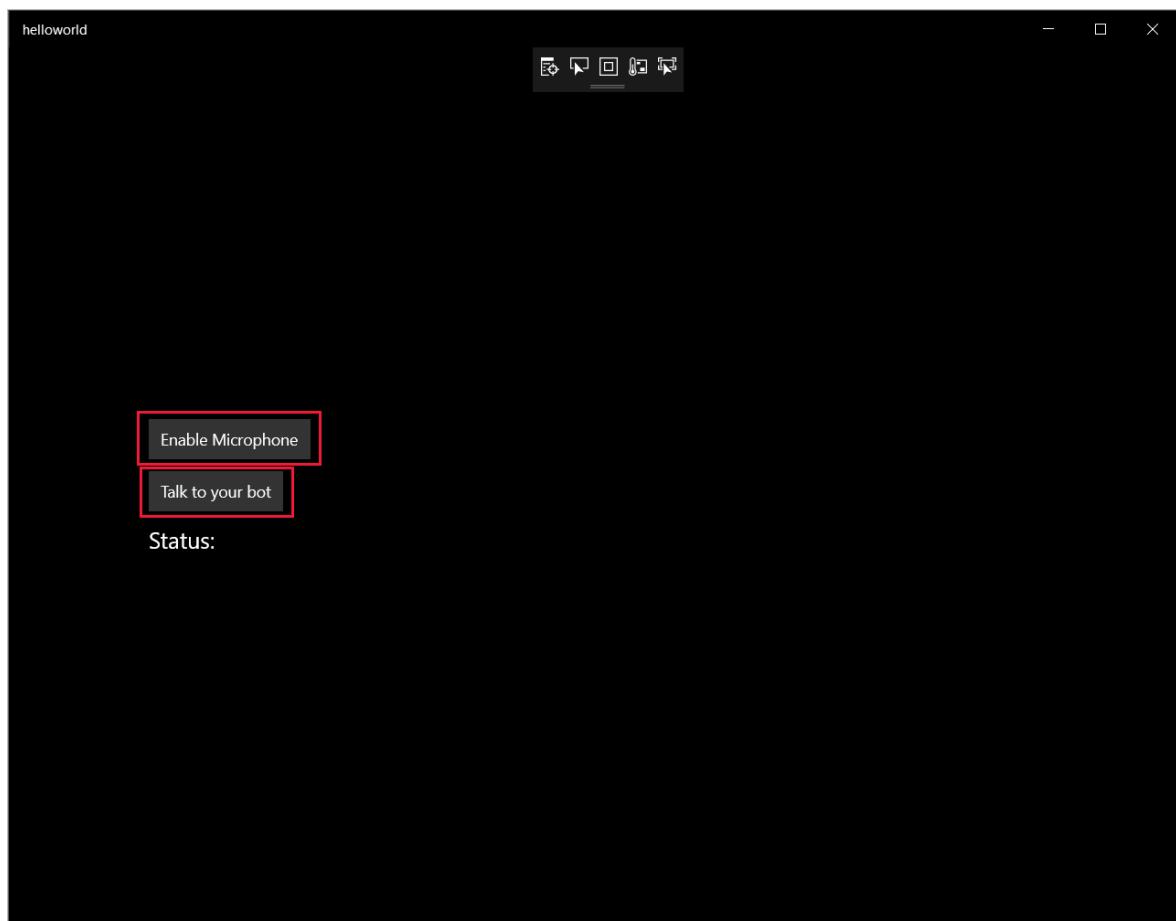
```

```
    }
    catch (Exception ex)
    {
        NotifyUser($"Exception: {ex.ToString()}\"", NotifyType.ErrorMessage);
    }
}
```

Build and run your app

Now you're ready to build your app and test your custom voice assistant using the Speech service.

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**.

Let helloworld access your microphone?

Let helloworld access your microphone?

To change this later, go to the Settings app.

Yes

No

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

[Explore C# samples on GitHub](#)

Tutorial: Voice-enable your bot

Article • 01/11/2023 • 21 minutes to read

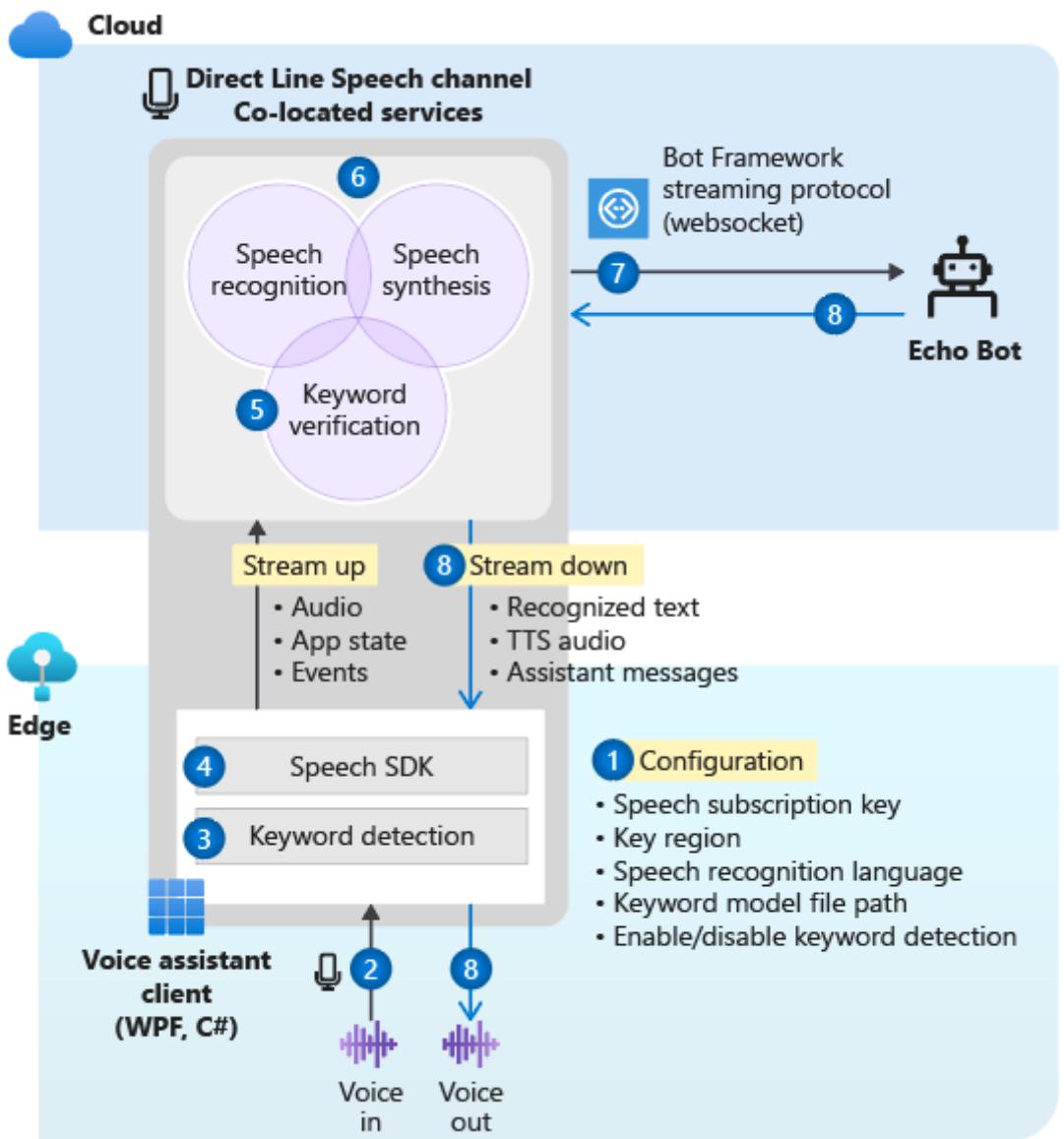
You can use Azure Cognitive Services Speech to voice-enable a chat bot.

In this tutorial, you'll use the Microsoft Bot Framework to create a bot that responds to what you say. You'll deploy your bot 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 speak back to you.

To complete the tutorial, you don't need extensive experience or familiarity with Azure, Bot Framework bots, or Direct Line Speech.

The voice-enabled chat bot that you make in this tutorial follows these steps:

1. The sample client application is configured to connect to the Direct Line Speech channel and the echo bot.
2. When the user presses a button, voice audio streams from the microphone. Or audio is continuously recorded when a custom keyword is used.
3. If a custom keyword is used, keyword detection happens on the local device, gating audio streaming to the cloud.
4. By using the Speech SDK, the sample client application connects to the 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 service, and streamed back to the client application for playback.



ⓘ Note

The steps in this tutorial don't require a paid service. As a new Azure user, you can 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 Azure App Service.
- ✓ Register your bot with a Direct Line Speech channel.
- ✓ Build and run the Windows Voice Assistant 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 later, with the **ASP.NET and web development** workload installed.
- [.NET Framework Runtime 4.6.1](#) 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 in the same Azure region.

This section walks you through creating a resource group in the West US region. You'll use this resource group when you're creating individual resources for the Bot Framework, the Direct Line Speech channel, and the Speech service.

1. Go to the [Azure portal page for creating a resource group](#).
2. Provide the following information:
 - Set **Subscription** to **Free Trial**. (You can also use an existing subscription.)
 - Enter a name for **Resource group**. We recommend **SpeechEchoBotTutorial-ResourceGroup**.
 - From the **Region** dropdown menu, select **West US**.
3. Select **Review and create**. You should see a banner that reads **Validation passed**.
4. Select **Create**. It might take a few minutes to create the resource group.
5. 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 want to pin this resource group, select the pin icon next to the name.

Choose an Azure region

Ensure that you use a [supported Azure region](#). The Direct Line Speech channel uses the text-to-speech service, which has neural and standard voices. Neural voices are used at [these Azure regions](#), and standard voices (retiring) are used at [these Azure regions](#).

For more information about regions, see [Azure locations](#).

Create resources

Now that you have a resource group in a supported region, the next step is to create individual resources for each service that you'll use in this tutorial.

Create a Speech service resource

1. Go to the [Azure portal page for creating a Speech service resource](#).
2. Provide the following information:
 - For **Name**, we recommend **SpeechEchoBotTutorial-Speech** as the name of your resource.
 - For **Subscription**, make sure that **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**.
3. After you've entered all required information, select **Create**. It might take a few minutes to create the resource.
4. 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** area (under **Manage keys**) or the **Keys** area.

At this point, check that your resource group (**SpeechEchoBotTutorial-ResourceGroup**) has a Speech service resource:

Name	Type	Location
SpeechEchoBotTutorial-Speech	Cognitive Services	West US

Create an Azure App Service plan

An App Service plan defines a set of compute resources for a web app to run.

1. Go to the [Azure portal page for creating an Azure App Service plan](#).
2. Provide the following information:
 - Set **Subscription** to **Free Trial**. (You can also use an existing subscription.)
 - For **Resource group**, select **SpeechEchoBotTutorial-ResourceGroup**.

- For **Name**, we recommend **SpeechEchoBotTutorial-AppServicePlan** as the name of your plan.
- 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, set **Operating System** to **Windows**.

3. Select **Review and create**. You should see a banner that reads **Validation passed**.

4. Select **Create**. It might take a few minutes to create the resource.

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 resources, start with the echo bot sample, which echoes the text that you've entered as its response. The sample code is already configured to work with the Direct Line Speech channel, which you'll connect after you've deployed the bot to Azure.

 **Note**

The instructions that follow, along with more 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:

Bash

```
git clone https://github.com/Microsoft/botbuilder-samples.git
```

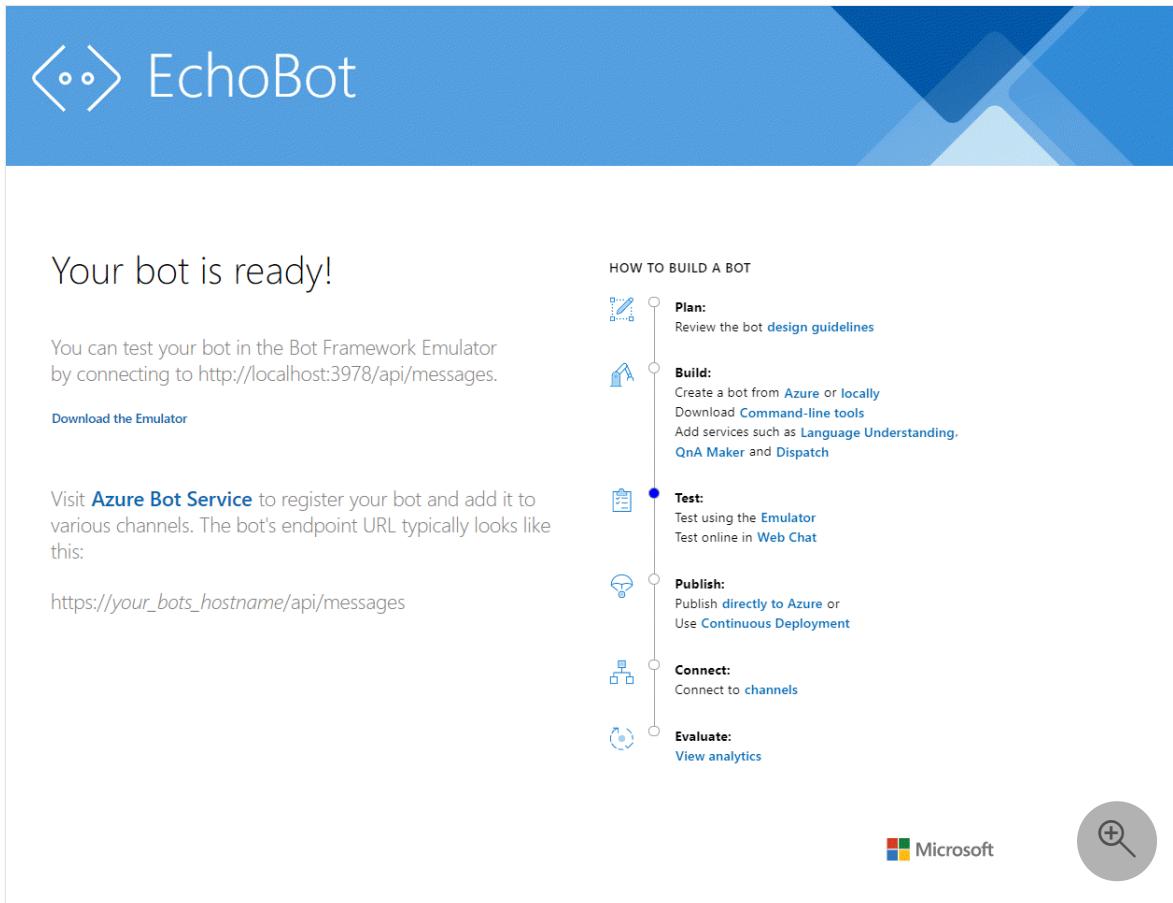
2. Open Visual Studio.

3. From the toolbar, select **File > Open > Project/Solution**. Then open the project solution:



4. After the project is loaded, select the **F5** key to build and run the project.

In the browser that opens, you'll see a screen similar to this one:



Test the bot sample with the Bot Framework Emulator

The [Bot Framework Emulator](#) is a desktop app that lets bot developers test and debug their bots locally (or remotely through a tunnel). The emulator accepts typed text as the input (not voice). The bot will also respond 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 you deploy the bot to Azure, you'll test it with voice input and voice output.

1. Install [Bot Framework Emulator](#) version 4.3.0 or later.

2. Open the Bot Framework Emulator, and then select **File > Open Bot**.

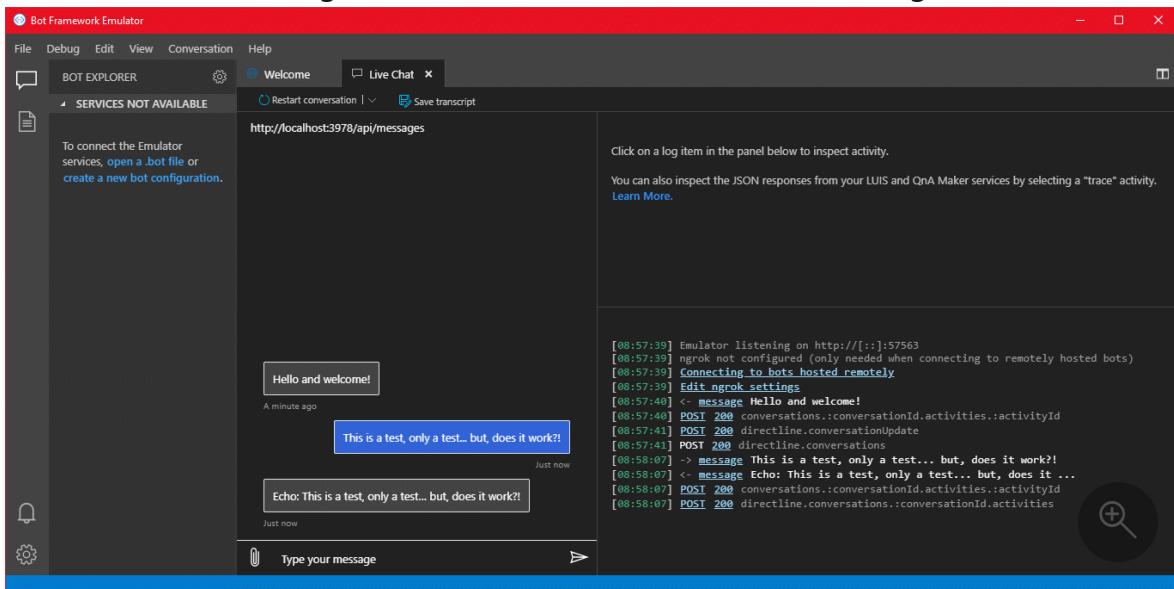
3. Enter the URL for your bot. For example:

http://localhost:3978/api/messages

4. Select **Connect**.

5. The bot should greet you with a "Hello and welcome!" message. Type in any text message and confirm that you get a response from the bot.

This is what an exchange of communication with an echo bot might look like:



Deploy your bot to Azure App Service

The next step is to deploy the echo bot to Azure. There are a few ways to deploy a bot, including the [Azure CLI](#) and [deployment templates](#). This tutorial focuses on publishing directly from Visual Studio.

ⓘ Note

If **Publish** doesn't appear as you perform the following steps, use Visual Studio Installer to add the **ASP.NET and web development** workload.

1. From Visual Studio, open the echo bot that's been configured for use with the Direct Line Speech channel:

samples\csharp_dotnetcore\02.echo-bot\EchoBot.sln

2. In Solution Explorer, right-click the **EchoBot** project and select **Publish**.
3. In the **Publish** window that opens:
 - a. Select **Azure > Next**.
 - b. Select **Azure App Service (Windows) > Next**.
 - c. Select **Create a new Azure App Service** by the green plus sign.
4. When the **App Service (Windows)** window appears:
 - Select **Add an account**, and sign in with your Azure account credentials. If you're already signed in, select your account from the dropdown list.
 - For **Name**, enter a globally unique name for your bot. This name is used to create a unique bot URL.
A default name that includes the date and time appears in the box (for example, **EchoBot20190805125647**). You can use the default name for this tutorial.
 - For **Subscription**, select **Free Trial**.
 - For **Resource Group**, select **SpeechEchoBotTutorial-ResourceGroup**.
 - For **Hosting Plan**, select **SpeechEchoBotTutorial-AppServicePlan**.

5. Select **Create**. On the final wizard screen, select **Finish**.

6. Select **Publish**. Visual Studio deploys the bot to Azure.

You should see a success message in the Visual Studio output window that looks like this:



Your default browser should open and display a page that reads: "Your bot is ready!"

At this point, check your resource group (**SpeechEchoBotTutorial-ResourceGroup**) in the Azure portal. Confirm that it contains these three resources:

Name	Type	Location

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 need to make a small configuration change so that your bot can communicate with the Direct Line Speech channel by using web sockets. Follow these steps to enable web sockets:

1. Go to the [Azure portal](#), and select your App Service resource. The resource name should be similar to **EchoBot20190805125647** (your unique app name).
2. On the left pane, under **Settings**, select **Configuration**.
3. Select the **General settings** tab.
4. Find the toggle for **Web sockets** and set it to **On**.
5. Select **Save**.

Tip

You can use the controls at the top of your Azure App Service page to stop or restart the service. This ability can come in handy when you're troubleshooting.

Create a channel registration

Now that you've created an Azure App Service resource to host your bot, the next step is to create a channel registration. Creating a channel registration is a prerequisite for registering your bot with Bot Framework channels, including the Direct Line Speech channel. If you want to learn more about how bots use channels, see [Connect a bot to channels](#).

1. Go to the [Azure portal page for creating an Azure bot](#).
2. Provide the following information:
 - For **Bot handle**, enter **SpeechEchoBotTutorial-BotRegistration-####**. Replace **####** with a number of your choice.

Note

The bot handle must be globally unique. If you enter one and get the error message "The requested bot ID is not available," pick a different number. The following examples use **8726**.

- For **Subscription**, select **Free Trial**.
 - For **Resource group**, select **SpeechEchoBotTutorial-ResourceGroup**.
 - For **Location**, select **West US**.
 - For **Pricing tier**, select **F0**.
 - Ignore **Auto create App ID and password**.
3. At the bottom of the **Azure Bot** pane, select **Create**.
4. After you create the resource, open your **SpeechEchoBotTutorial-BotRegistration-8726** resource in the Azure portal.
5. From the **Settings** area, select **Configuration**.
6. For **Messaging endpoint**, enter the URL for your web app with the **/api/messages** path appended. For example, if your globally unique app name was **EchoBot20190805125647**, your messaging endpoint would be
`https://EchoBot20190805125647.azurewebsites.net/api/messages/`.

At this point, check your resource group (**SpeechEchoBotTutorial-ResourceGroup**) in the Azure portal. It should now show at least four resources:

Name	Type	Location
EchoBot20190805125647	App Service	West US
SpeechEchoBotTutorial-AppServicePlan	App Service plan	West US
SpeechEchoBotTutorial-BotRegistration-8726	Bot Service	Global
SpeechEchoBotTutorial-Speech	Cognitive Services	West US

ⓘ Important

The Azure Bot Service resource shows the Global region, even though you selected West US. This is expected.

Optional: Test in web chat

The Azure Bot page has a **Test in Web Chat** option under **Settings**. It won't work by default with your bot because the web chat needs to authenticate against your bot.

If you want to test your deployed bot with text input, use the following steps. Note that these steps are optional and aren't required for you to continue with the tutorial.

1. In the [Azure portal](#), find and open your **EchoBotTutorial-BotRegistration-####** resource.
2. From the **Settings** area, select **Configuration**. Copy the value under **Microsoft App ID**.
3. Open the Visual Studio EchoBot solution. In Solution Explorer, find and double-click **appsettings.json**.
4. Replace the empty string next to **MicrosoftAppId** in the JSON file with the copied ID value.
5. Go back to the Azure portal. In the **Settings** area, select **Configuration**. Then select **Manage** next to **Microsoft App ID**.
6. Select **New client secret**. Add a description (for example, **web chat**) and select **Add**. Copy the new secret.
7. Replace the empty string next to **MicrosoftAppPassword** in the JSON file with the copied secret value.
8. Save the JSON file. It should look something like this code:

JSON

```
{  
  "MicrosoftAppId": "3be0abc2-ca07-475e-b6c3-90c4476c4370",  
  "MicrosoftAppPassword": "-zRhJZ~1cnc7ZIlj4Qozs_eKN.8Cq~U38G"  
}
```

9. Republish the app: right-click the **EchoBot** project in Visual Studio Solution Explorer, select **Publish**, and then select the **Publish** button.

Register the Direct Line Speech channel

Now it's time to register your bot with the Direct Line Speech channel. This channel creates a connection between your bot and a client app compiled with the Speech SDK.

1. In the [Azure portal](#), find and open your **SpeechEchoBotTutorial-BotRegistration-####** resource.
2. From the **Settings** area, select **Channels** and then take the following steps:
 - a. Under **More channels**, select **Direct Line Speech**.
 - b. Review the text on the **Configure Direct line Speech** page, and then expand the **Cognitive service account** dropdown menu.
 - c. Select the Speech service resource that you created earlier (for example, **SpeechEchoBotTutorial-Speech**) from the menu to associate your bot with your subscription key.
 - d. Ignore the rest of the optional fields.
 - e. Select **Save**.
3. From the **Settings** area, select **Configuration** and then take the following steps:
 - a. Select the **Enable Streaming Endpoint** checkbox. This step is necessary for creating a communication protocol built on web sockets between your bot and the Direct Line Speech channel.
 - b. Select **Save**.

If you want to learn more, see [Connect a bot to Direct Line Speech](#).

Run the Windows Voice Assistant Client

The Windows Voice Assistant Client is a Windows Presentation Foundation (WPF) app in C# that uses the [Speech SDK](#) to manage communication with your bot through the Direct Line Speech channel. Use it to interact with and test your bot before writing a custom client app. It's open source, so you can either download the executable file and run it, or build it yourself.

The Windows Voice Assistant 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.

Note

At this point, confirm that your microphone and speakers are enabled and working.

1. Go to the [GitHub repository for the Windows Voice Assistant Client](#).
2. Follow the provided instructions to either:

- Download a prebuilt executable file in a .zip package to run
- Build the executable file yourself, by cloning the repository and building the project

3. Open the **VoiceAssistantClient.exe** client application and configure it to connect to your bot, by following the instructions in the GitHub repository.

4. Select **Reconnect** and make sure you see the message "New conversation started - type or press the microphone button."

5. Let's test it out. Select the microphone button, and speak a few words in English.

The recognized text appears as you speak. When you're done speaking, the bot replies in its own voice, saying "echo" followed by the recognized words.

You can also use text to communicate with the bot. Just type in the text on the bottom bar.

Troubleshoot errors in the Windows Voice Assistant Client

If you get an error message in your main app window, use this table to identify and troubleshoot the problem:

Message	What should you do?
Error (AuthenticationFailure) : WebSocket Upgrade failed with an authentication error (401). Check for correct resource key (or authorization token) and region name	On the Settings page of the app, make sure that you entered the key and its region 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 that you selected the Enable Streaming Endpoint checkbox and/or turned on web sockets . Make sure that Azure App Service is running. If it is, try restarting it.
Error (ConnectionFailure) : Connection was closed by the remote host. Error code: 1002. Error details: The server returned status code '503' when status code '101' was expected	Make sure that you selected the Enable Streaming Endpoint checkbox box and/or turned on web sockets . Make sure that Azure App Service is running. If it is, try restarting it.
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 the speak ↴ field of its output activity, but the Azure region associated with your resource key doesn't support neural voices. See neural voices and standard voices .

If the actions in the table don't address your problem, see [Voice assistants: Frequently asked questions](#). If you still can't resolve your problem after following all the steps in this tutorial, please enter a new issue on the [Voice Assistant GitHub page](#).

A note on connection timeout

If you're connected to a bot and no activity has happened in the last five minutes, the service automatically closes the web socket connection with the client and with the bot. This is by design. A message appears on the bottom bar: "Active connection timed out but ready to reconnect on demand."

You don't need to select the **Reconnect** button. Press the microphone button and start talking, enter a text message, or say the keyword (if one is enabled). The connection is automatically reestablished.

View bot activities

Every bot sends and receives activity messages. In the **Activity Log** window of the Windows Voice Assistant Client, timestamped logs show each activity that the client has received from the bot. You can also see the activities that the client sent to the bot by using the [DialogServiceConnector.SendActivityAsync](#) method. When you select a log item, it shows the details of the associated activity as JSON.

Here's sample JSON of an activity that the client received:

```
JSON

{
  "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-8726"
    },
    "id": "89841b4d-46ce-42de-9960-4fe4070c70cc",
    "name": "SpeechEchoBotTutorial-BotRegistration-8726"
  }
}
```

```
    "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='en-US-JennyNeural'>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 the [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 Windows Voice Assistant 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 the file [VoiceAssistantClient\MainWindow.xaml.cs](#), which creates these two Speech SDK objects:

- [DialogServiceConfig](#): For configuration settings like resource key and its 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 a Microsoft 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 multiple-word phrase.

Note

The term *keyword* is often used interchangeably with the term *wake word*. You might see both used in Microsoft documentation.

Keyword detection happens on the client app. If you're using a keyword, audio is streamed to the Direct Line Speech channel only if the keyword is detected. The Direct Line Speech channel includes a component called *keyword verification*, 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 keyword verification succeeds, then the channel will communicate with the bot.

Follow these steps to create a keyword model, configure the Windows Voice Assistant Client to use this model, and test it with your bot:

1. [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 Windows Voice Assistant Client, find the **Settings** menu (the gear icon in the upper right). For **Model file path**, enter the full path name for the **kws.table** file from step 2.
4. Select the **Enabled** checkbox. You should see this message next to the checkbox: "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 the values for **Subscription key** and **Subscription key region**, and then select **OK** to close the **Settings** menu.
6. Select **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 you're finished, two things happen:
 - You see a transcription of what you spoke.
 - You hear the bot's response.
8. Continue to experiment with the three input types that your bot supports:
 - Entering text on the bottom bar
 - Pressing the microphone icon and speaking
 - Saying a phrase that starts with your keyword

View the source code that enables keyword detection

In the source code of the Windows Voice Assistant Client, use these files to review the code that enables keyword detection:

- [VoiceAssistantClient\Models.cs](#) includes a call to the Speech SDK method `KeywordRecognitionModel.fromFile()`. This method is used to instantiate the model from a local file on disk.

- `VoiceAssistantClient\MainWindow.xaml.cs` includes a call to the Speech SDK method `DialogServiceConnector.StartKeywordRecognitionAsync()`. This method 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 US 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'll also learn how to select a different voice for that language.

Change the language

You can choose from any of the languages mentioned in the [speech-to-text](#) table. The following example changes the language to German.

1. Open the Windows Voice Assistant Client app, select 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 step sets the spoken language to be recognized, overriding the default `en-us`. It also instructs the Direct Line Speech channel to use a default German voice for the bot reply.

2. Close the **Settings** page, and then select the **Reconnect** button to establish a new connection to your echo bot.
3. Select the microphone button, and say a phrase in German. The recognized text appears, and the echo bot replies with the default German voice.

Change the default bot voice

You can select the text-to-speech voice and control pronunciation if the bot specifies the reply in the form of a [Speech Synthesis Markup Language](#) (SSML) instead of simple text. The echo bot doesn't use SSML, but you can easily modify the code to do that.

The following example adds SSML to the echo bot reply so that the German voice `de-DE-RalfNeural` (a male voice) is used instead of the default female voice. See the [list of standard voices](#) and [list of neural voices](#) that are supported for your language.

1. Open samples\csharp_dotnetcore\02.echo-bot\echo-bot.cs.

2. Find these lines:

```
C#
```

```
var replyText = $"Echo: {turnContext.Activity.Text}";  
await turnContext.SendActivityAsync(MessageFactory.Text(replyText,  
replyText), cancellationToken);
```

Replace them with this code:

```
C#
```

```
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='de-DE-RalfNeural'>" +  
    $"{replyText}" + "</voice></speak>";  
await turnContext.SendActivityAsync(MessageFactory.Text(replyText,  
replySpeak), cancellationToken);
```

3. 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 preceding change, it has been replaced from simple text to SSML in order to specify a non-default German voice.

Redeploy your bot

Now that you've made the necessary change to the bot, the next step is to republish it to Azure App Service and try it out:

1. In the Solution Explorer window, right-click the **EchoBot** project and select **Publish**.
2. Your previous deployment configuration has already been loaded as the default. Select **Publish** next to **EchoBot20190805125647 - Web Deploy**.

The **Publish Succeeded** message appears in the Visual Studio output window, and a webpage opens with the message "Your bot is ready!"

3. Open the Windows Voice Assistant Client app. Select the **Settings** button (upper-right gear icon), and make sure that you still have **de-de** in the **Language** field.
4. Follow the instructions in [Run the Windows Voice Assistant 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 deleting the Azure resource group:

1. In the [Azure portal](#), select Resource Groups under Azure services.
2. Find the **SpeechEchoBotTutorial-ResourceGroup** resource group. Select the three dots (...).
3. Select Delete resource group.

Explore documentation

- [Deploy to an Azure region near you](#) to see the improvement in bot response time.
- [Deploy to an Azure region that supports high-quality neural text-to-speech voices](#).
- Get pricing associated with the Direct Line Speech channel:
 - [Bot Service pricing](#)
 - [Speech service](#)
- Build and deploy your own voice-enabled bot:
 - Build a [Bot Framework bot](#). Then [register it with the 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](#).

Next steps

[Build your own client app by using the Speech SDK](#)

Additional resources

 Documentation

[Direct Line Speech - Speech service - Azure Cognitive Services](#)

An overview of the features, capabilities, and restrictions for Voice assistants using Direct Line Speech with the Speech Software Development Kit (SDK).

[Connect a bot to Direct Line Speech - Bot Service](#)

Learn how to connect a bot to the Direct Line Speech channel for user's voice interaction with high reliability and low-latency.

[List entity type - LUIS - Azure](#)

List entities represent a fixed, closed set of related words along with their synonyms. LUIS does not discover additional values for list entities. Use the Recommend feature to see suggestions for new words based on the current list.

[How to use train and test - Azure Cognitive Services](#)

Learn how to train and test the application.

[Data sources and content types - QnA Maker - Azure Cognitive Services](#)

Learn how to import question and answer pairs from data sources and supported content types, which include many standard structured documents such as PDF, DOCX, and TXT - QnA Maker.

[What's New - Language Understanding \(LUIS\) - Azure](#)

This article is regularly updated with news about the Azure Cognitive Services Language Understanding API.

[conversational language understanding data formats - Azure Cognitive Services](#)

Learn about the data formats accepted by conversational language understanding.

[Sign in to the LUIS portal and create an app - Azure Cognitive Services](#)

Learn how to sign in to LUIS and create application.

[Show 5 more](#)

What are Voice Assistants on Windows?

Article • 04/22/2022 • 2 minutes to read

Voice assistant applications can take advantage of the Windows ConversationalAgent APIs to achieve a complete voice-enabled assistant experience.

Voice Assistant Features

Voice agent applications can be activated by a spoken keyword to enable a hands-free, voice driven experience. Voice activation works when the application is closed and when the screen is locked.

In addition, Windows provides a set of voice-activation privacy settings that gives users control of voice activation and above lock activation on a per-app basis.

After voice activation, Windows will manage multiple active agents properly and notify each voice assistant if they are interrupted or deactivated. This allows applications to manage interruptions and other inter-agent events properly.

How does voice activation work?

The Agent Activation Runtime (AAR) is the ongoing process in Windows that manages application activation on a spoken keyword or button press. It starts with Windows as long as there is at least one application on the system that is registered with the system. Applications interact with AAR through the ConversationalAgent APIs in the Windows SDK.

When the user speaks a keyword, the software or hardware keyword spotter on the system notifies AAR that a keyword has been detected, providing a keyword ID. AAR in turn sends a request to BackgroundService to start the application with the corresponding application ID.

Registration

The first time a voice activated application is run, it registers its app ID and keyword information through the ConversationalAgent APIs. AAR registers all configurations in the global mapping with the hardware or software keyword spotter on the system, allowing them to detect the application's keyword. The application also [registers with the Background Service](#).

Note that this means an application cannot be activated by voice until it has been run once and registration has been allowed to complete.

Receiving an activation

Upon receiving the request from AAR, the Background Service launches the application. The application receives a signal through the `OnBackgroundActivated` life-cycle method in `App.xaml.cs` with a unique event argument. This argument tells the application that it was activated by AAR and that it should start keyword verification.

If the application successfully verifies the keyword, it can make a request that appears in the foreground. When this request succeeds, the application displays UI and continues its interaction with the user.

AAR still signals active applications when their keyword is spoken. Rather than signaling through the life-cycle method in `App.xaml.cs`, though, it signals through an event in the `ConversationalAgent` APIs.

Keyword verification

The keyword spotter that triggers the application to start has achieved low power consumption by simplifying the keyword model. This allows the keyword spotter to be "always on" without a high power impact, but it also means the keyword spotter will likely have a high number of "false accepts" where it detects a keyword even though no keyword was spoken. This is why the voice activation system launches the application in the background: to give the application a chance to verify that the keyword was spoken before interrupting the user's current session. AAR saves the audio since a few seconds before the keyword was spotted and makes it accessible to the application. The application can use this to run a more reliable keyword spotter on the same audio.

Next steps

- Review the [design guidelines](#) to provide the best experiences for voice activation.
- See the voice assistants on Windows [get started](#) page.
- See the [UWP Voice Assistant Sample](#) page and follow the steps to get the sample client running.

Get started with voice assistants on Windows

Article • 09/20/2022 • 2 minutes to read

This guide will take you through the steps to begin developing a voice assistant on Windows.

Set up your development environment

To start developing a voice assistant for Windows, you will need to make sure you have the proper development environment.

- **Visual Studio:** You will need to install [Microsoft Visual Studio 2017](#), Community Edition or higher
- **Windows version:** A PC with a Windows Insider fast ring build of Windows and the Windows Insider version of the Windows SDK. This sample 1600 using Windows SDK 19018. Any Build or SDK above the specified versions should be compatible.
- **UWP development tools:** The Universal Windows Platform development workload in Visual Studio. See the UWP [Get set up](#) page to get your machine ready for developing UWP Applications.
- **A working microphone and audio output**

Obtain resources from Microsoft

Some resources necessary for a completely customized voice agent on Windows will require resources from Microsoft. The [UWP Voice Assistant Sample](#) provides sample versions of these resources for initial development and testing, so this section is unnecessary for initial development.

- **Keyword model:** Voice activation requires a keyword model from Microsoft in the form of a .bin file. The .bin file provided in the UWP Voice Assistant Sample is trained on the keyword *Contoso*.
- **Limited Access Feature Token:** Since the ConversationalAgent APIs provide access to microphone audio, they are protected under Limited Access Feature restrictions. To use a Limited Access Feature, you will need to obtain a Limited Access Feature token connected to the package identity of your application.

Establish a dialog service

For a complete voice assistant experience, the application will need a dialog service that

- Detect a keyword in a given audio file
- Listen to user input and convert it to text
- Provide the text to a bot
- Translate the text response of the bot to an audio output

These are the requirements to create a basic dialog service using Direct Line Speech.

- **Speech resource:** A resource for Cognitive Speech Services for speech-to-text and text-to-speech conversions. Create a Speech resource on the [Azure portal](#). For more information, see [Create a new Azure Cognitive Services resource](#).
- **Bot Framework bot:** A bot created using Bot Framework version 4.2 or above that's subscribed to [Direct Line Speech](#) to enable voice input and output. [This guide](#) contains step-by-step instructions to make an "echo bot" and subscribe it to Direct Line Speech. You can also go [here](#) for steps on how to create a customized bot, then follow the same steps [here](#) to subscribe it to Direct Line Speech, but with your new bot rather than the "echo bot".

Try out the sample app

With your Speech resource key and echo bot's bot ID, you're ready to try out the [UWP Voice Assistant sample](#). Follow the instructions in the readme to run the app and enter your credentials.

Create your own voice assistant for Windows

Once you've received your Limited Access Feature token and bin file from Microsoft, you can begin on your own voice assistant on Windows.

Next steps

[Read the voice assistant implementation guide](#)

Implementing Voice Assistants on Windows

Article • 04/22/2022 • 6 minutes to read

This guide walks through important implementation details for creating a voice assistant on Windows.

Implementing voice activation

After [setting up your environment](#) and learning [how voice activation works](#), you can start implementing voice activation for your own voice assistant application.

Registration

Ensure that the microphone is available and accessible, then monitor its state

MVA needs a microphone to be present and accessible to be able to detect a voice activation. Use the [AppCapability](#), [DeviceWatcher](#), and [MediaCapture](#) classes to check for microphone privacy access, device presence, and device status (like volume and mute) respectively.

Register the application with the background service

In order for MVA to launch the application in the background, the application needs to be registered with the Background Service. See a full guide for Background Service registration [here](#).

Unlock the Limited Access Feature

Use your Microsoft-provided Limited Access Feature key to unlock the voice assistant feature. Use the [LimitedAccessFeature](#) class from the Windows SDK to do this.

Register the keyword for the application

The application needs to register itself, its keyword model, and its language with Windows.

Start by retrieving the keyword detector. In this sample code, we retrieve the first detector, but you can select a particular detector by selecting it from `configurableDetectors`.

C#

```
private static async Task<ActivationSignalDetector>
GetFirstEligibleDetectorAsync()
{
    var detectorManager = ConversationalAgentDetectorManager.Default;
    var allDetectors = await
detectorManager.GetAllActivationSignalDetectorsAsync();
    var configurableDetectors = allDetectors.Where(candidate =>
candidate.CanCreateConfigurations
        && candidate.Kind == ActivationSignalDetectorKind.AudioPattern
        &&
(candidate.SupportedModelDataTypes.Contains("MICROSOFT_KWSGRAPH_V1")));

    if (configurableDetectors.Count() != 1)
    {
        throw new NotSupportedException($"System expects one eligible
configurable keyword spotter; actual is {configurableDetectors.Count()}.");
    }

    var detector = configurableDetectors.First();

    return detector;
}
```

After retrieving the `ActivationSignalDetector` object, call its

`ActivationSignalDetector.CreateConfigurationAsync` method with the signal ID, model ID, and display name to register your keyword and retrieve your application's `ActivationSignalDetectionConfiguration`. The signal and model IDs should be GUIDs decided on by the developer and stay consistent for the same keyword.

Verify that the voice activation setting is enabled

To use voice activation, a user needs to enable voice activation for their system and enable voice activation for their application. You can find the setting under "Voice activation privacy settings" in Windows settings. To check the status of the voice activation setting in your application, use the instance of the `ActivationSignalDetectionConfiguration` from registering the keyword. The [AvailabilityInfo](#) field on the `ActivationSignalDetectionConfiguration` contains an enum value that describes the state of the voice activation setting.

Retrieve a `ConversationalAgentSession` to register the app with the MVA system

The `ConversationalAgentSession` is a class in the Windows SDK that allows your app to update Windows with the app state (Idle, Detecting, Listening, Working, Speaking) and receive events, such as activation detection and system state changes such as the screen locking. Retrieving an instance of the AgentSession also serves to register the application with Windows as activatable by voice. It is best practice to maintain one reference to the `ConversationalAgentSession`. To retrieve the session, use the `ConversationalAgentSession.GetCurrentSessionAsync` API.

Listen to the two activation signals: the `OnBackgroundActivated` and `OnSignalDetected`

Windows will signal your app when it detects a keyword in one of two ways. If the app is not active (that is, you do not have a reference to a non-disposed instance of `ConversationalAgentSession`), then it will launch your app and call the `OnBackgroundActivated` method in the `App.xaml.cs` file of your application. If the event arguments' `BackgroundActivatedEventArgs.TaskInstance.Task.Name` field matches the string "AgentBackgroundTrigger", then the application startup was triggered by voice activation. The application needs to override this method and retrieve an instance of `ConversationalAgentSession` to signal to Windows that is now active. When the application is active, Windows will signal the occurrence of a voice activation using the `ConversationalAgentSession.OnSignalDetected` event. Add an event handler to this event as soon as you retrieve the `ConversationalAgentSession`.

Keyword verification

Once a voice agent application is activated by voice, the next step is to verify that the keyword detection was valid. Windows does not provide a solution for keyword verification, but it does allow voice assistants to access the audio from the hypothesized activation and complete verification on its own.

Retrieve activation audio

Create an `AudioGraph` and pass it to the `CreateAudioDeviceInputNodeAsync` of the `ConversationalAgentSession`. This will load the graph's audio buffer with the audio *starting approximately 3 seconds before the keyword was detected*. This additional leading audio is included to accommodate a wide range of keyword lengths and speaker

speeds. Then, handle the [QuantumStarted](#) event from the audio graph to retrieve the audio data.

```
C#
```

```
var inputNode = await  
agentSession.CreateAudioDeviceInputNodeAsync(audioGraph);  
var outputNode = inputGraph.CreateFrameOutputNode();  
inputNode.AddOutgoingConnection(outputNode);  
audioGraph.QuantumStarted += OnQuantumStarted;
```

Note: The leading audio included in the audio buffer can cause keyword verification to fail. To fix this issue, trim the beginning of the audio buffer before you send the audio for keyword verification. This initial trim should be tailored to each assistant.

Launch in the foreground

When keyword verification succeeds, the application needs to move to the foreground to display UI. Call the [ConversationalAgentSession.RequestForegroundActivationAsync](#) API to move your application to the foreground.

Transition from compact view to full view

When your application is first activated by voice, it is started in a compact view. Please read the [Design guidance for voice activation preview](#) for guidance on the different views and transitions between them for voice assistants on Windows.

To make the transition from compact view to full app view, use the [ApplicationView API](#) [TryEnterViewModeAsync](#):

```
C#
```

```
var appView = ApplicationView.GetForCurrentView();  
await appView.TryEnterViewModeAsync(ApplicationViewMode.Default);
```

Implementing above lock activation

The following steps cover the requirements to enable a voice assistant on Windows to run above lock, including references to example code and guidelines for managing the application lifecycle.

For guidance on designing above lock experiences, visit the [best practices guide](#).

When an app shows a view above lock, it is considered to be in "Kiosk Mode". For more information on implementing an app that uses Kiosk Mode, see the [kiosk mode documentation](#).

Transitioning above lock

An activation above lock is similar to an activation below lock. If there are no active instances of the application, a new instance will be started in the background and `OnBackgroundActivated` in `App.xaml.cs` will be called. If there is an instance of the application, that instance will get a notification through the `ConversationalAgentSession.SignalDetected` event.

If the application does not appear above lock, it must call `ConversationalAgentSession.RequestForegroundActivationAsync`. This triggers the `OnLaunched` method in `App.xaml.cs` which should navigate to the view that will appear above lock.

Detecting lock screen transitions

The `ConversationalAgent` library in the Windows SDK provides an API to make the lock screen state and changes to the lock screen state easily accessible. To detect the current lock screen state, check the `ConversationalAgentSession.IsUserAuthenticated` field. To detect changes in lock state, add an event handler to the `ConversationalAgentSession` object's `SystemStateChanged` event. It will fire whenever the screen changes from unlocked to locked or vice versa. If the value of the event arguments is `ConversationalAgentSystemStateChangeType.UserAuthentication`, then the lock screen state has changed.

```
C#
```

```
conversationalAgentSession.SystemStateChanged += (s, e) =>
{
    if (e.SystemStateChangeType ==
ConversationalAgentSystemStateChangeType.UserAuthentication)
    {
        // Handle lock state change
    }
};
```

Detecting above lock activation user preference

The application entry in the Voice Activation Privacy settings page has a toggle for above lock functionality. For your app to be able to launch above lock, the user will need to turn this setting on. The status of above lock permissions is also stored in the `ActivationSignalDetectionConfiguration` object. The `AvailabilityInfo.HasLockScreenPermission` status reflects whether the user has given above lock permission. If this setting is disabled, a voice application can prompt the user to navigate to the appropriate settings page at the link "ms-settings:privacy-voiceactivation" with instructions on how to enable above-lock activation for the application.

Closing the application

To properly close the application programmatically while above or below lock, use the `WindowService.CloseWindow()` API. This triggers all UWP lifecycle methods, including `OnSuspend`, allowing the application to dispose of its `ConversationalAgentSession` instance before closing.

Note

The application can close without closing the **below lock** instance. In this case, the above lock view needs to "clean up", ensuring that once the screen is unlocked, there are no event handlers or tasks that will try to manipulate the above lock view.

Next steps

Visit the [UWP Voice Assistant Sample app](#) for examples and code walk-throughs

Privacy guidelines for voice assistants on Windows

Article • 04/22/2022 • 2 minutes to read

It's important that users are given clear information about how their voice data is collected and used and important that they are given control over if and how this collection happens. These core facets of privacy -- *disclosure* and *consent* -- are especially important for voice assistants integrated into Windows given the always-listening nature of their use.

Developers creating voice assistants on Windows must include clear user interface elements within their applications that reflect the listening capabilities of the assistant.

ⓘ Note

Failure to provide appropriate disclosure and consent for an assistant application, including after application updates, may result in the assistant becoming unavailable for voice activation until privacy issues are resolved.

Minimum requirements for feature inclusion

Windows users can see and control the availability of their assistant applications in [Settings > Privacy > Voice activation](#).

Choose which apps can use voice activation

Turning off voice activation for an app here will not change the app's access to the microphone on this device. If microphone access is turned off for an app, these settings will be unavailable.



Cortana

Cortana starts listening when you say "Cortana" On

Use Cortana when your device is locked Off



To become eligible for inclusion in this list, contact Microsoft at winfoiceassistants@microsoft.com to get started. By default, users will need to explicitly enable voice activation for a new assistant in [Settings > Privacy > Voice Activation](#), which an application can protocol link to with `ms-settings:privacy-voiceactivation`. An

allowed application will appear in the list once it has run and used the `Windows.ApplicationModel.ConversationalAgent` APIs. Its voice activation settings will be modifiable once the application has obtained microphone consent from the user.

Because the Windows privacy settings include information about how voice activation works and has standard UI for controlling permission, disclosure and consent are both fulfilled. The assistant will remain in this allowed list as long as it does not:

- Mislead or misinform the user about voice activation or voice data handling by the assistant
- Unduly interfere with another assistant
- Break any other relevant Microsoft policies

If any of the above are discovered, Microsoft may remove an assistant from the allowed list until problems are resolved.

Note

In all cases, voice activation permission requires microphone permission. If an assistant application does not have microphone access, it will not be eligible for voice activation and will appear in the voice activation privacy settings in a disabled state.

Additional requirements for inclusion in Microphone consent

Assistant authors who want to make it easier and smoother for their users to opt in to voice activation can do so by meeting additional requirements to adequately fulfill disclosure and consent without an extra trip to the settings page. Once approved, voice activation will become available immediately once a user grants microphone permission to the assistant application. To qualify for this, an assistant application must do the following **before** prompting for Microphone consent (for example, by using the `AppCapability.RequestAccessAsync` API):

1. Provide clear and prominent indication to the user that the application would like to listen to the user's voice for a keyword, *even when the application is not running*, and would like the user's consent
2. Include relevant information about data usage and privacy policies, such as a link to an official privacy statement

3. Avoid any directive or leading wording (for example, "click yes on the following prompt") in the experience flow that discloses audio capture behavior

If an application accomplishes all of the above, it's eligible to enable voice activation capability together with Microphone consent. Contact winvoiceassistants@microsoft.com for more information and to review a first use experience.

 **Note**

Voice activation above lock is not eligible for automatic enablement with Microphone access and will still require a user to visit the Voice activation privacy page to enable above-lock access for an assistant.

Next steps

[Learn about best practices for voice assistants on Windows](#)

Design assistant experiences for Windows 10

Article • 04/22/2022 • 8 minutes to read

Voice assistants developed on Windows 10 must implement the user experience guidelines below to provide the best possible experiences for voice activation on Windows 10. This document will guide developers through understanding the key work needed for a voice assistant to integrate with the Windows 10 Shell.

Contents

- [Summary of voice activation views supported in Windows 10](#)
- [Requirements summary](#)
- [Best practices for good listening experiences](#)
- [Design guidance for in-app voice activation](#)
- [Design guidance for voice activation above lock](#)
- [Design guidance for voice activation preview](#)

Summary of voice activation views supported in Windows 10

Windows 10 infers an activation experience for the customer context based on the device context. The following summary table is a high-level overview of the different views available when the screen is on.

View (Availability)	Device context	Customer goal	Appears when	Design needs
In-app (19H1)	Below lock, assistant has focus	Interact with the assistant app	Assistant processes the request in-app	Main in-app view listening experience
Above lock (19H2)	Above lock, unauthenticated	Interact with the assistant, but from a distance	System is locked and assistant requests activation	Full-screen visuals for far-field UI. Implement dismissal policies to not block unlocking.

View (Availability)	Device context	Customer goal	Appears when	Design needs
Voice activation preview (20H1)	Below lock, assistant does not have focus	Interact with the assistant, but in a less intrusive way	System is below lock and assistant requests background activation	Minimal canvas. Resize or hand-off to the main app view as needed.

Requirements summary

Minimal effort is required to access the different experiences. However, assistants do need to implement the right design guidance for each view. This table below provides a checklist of the requirements that must be followed.

Voice activation view	Assistant requirements summary
In-app	<ul style="list-style-type: none"> • Process the request in-app • Provides UI indicators for listening states • UI adapts as window sizes change
Above lock	<ul style="list-style-type: none"> • Detect lock state and request activation • Do not provide always persistent UX that would block access to the Windows lock screen • Provide full screen visuals and a voice-first experience • Honor dismissal guidance below • Follow privacy and security considerations below
Voice activation preview	<ul style="list-style-type: none"> • Detect unlock state and request background activation • Draw minimal listening UX in the preview pane • Draw a close X in the top-right and self-dismiss and stop streaming audio when pressed • Resize or hand-off to the main assistant app view as needed to provide answers

Best practices for good listening experiences

Assistants should build a listening experience to provide critical feedback so the customer can understand the state of the assistant. Below are some possible states to consider when building an assistant experience. These are only possible suggestions, not mandatory guidance.

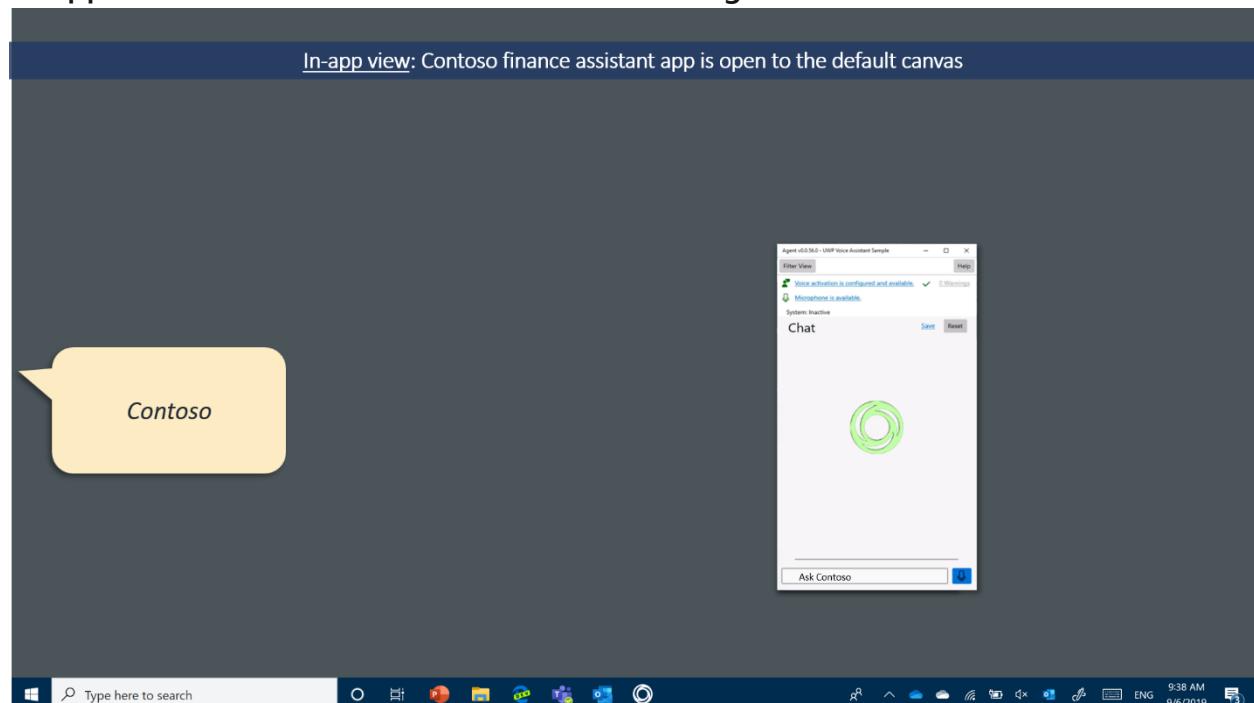
- Assistant is available for speech input
- Assistant is in the process of activating (either a keyword or mic button press)
- Assistant is actively streaming audio to the assistant cloud
- Assistant is ready for the customer to start speaking
- Assistant is hearing that words are being said
- Assistant understands that the customer is done speaking
- Assistant is processing and preparing a response
- Assistant is responding

Even if states change rapidly it is worth considering providing UX for states, since durations are variable across the Windows ecosystem. Visual feedback as well as brief audio chimes or chirps, also called "earcons", can be part of the solution. Likewise, visual cards coupled with audio descriptions make for good response options.

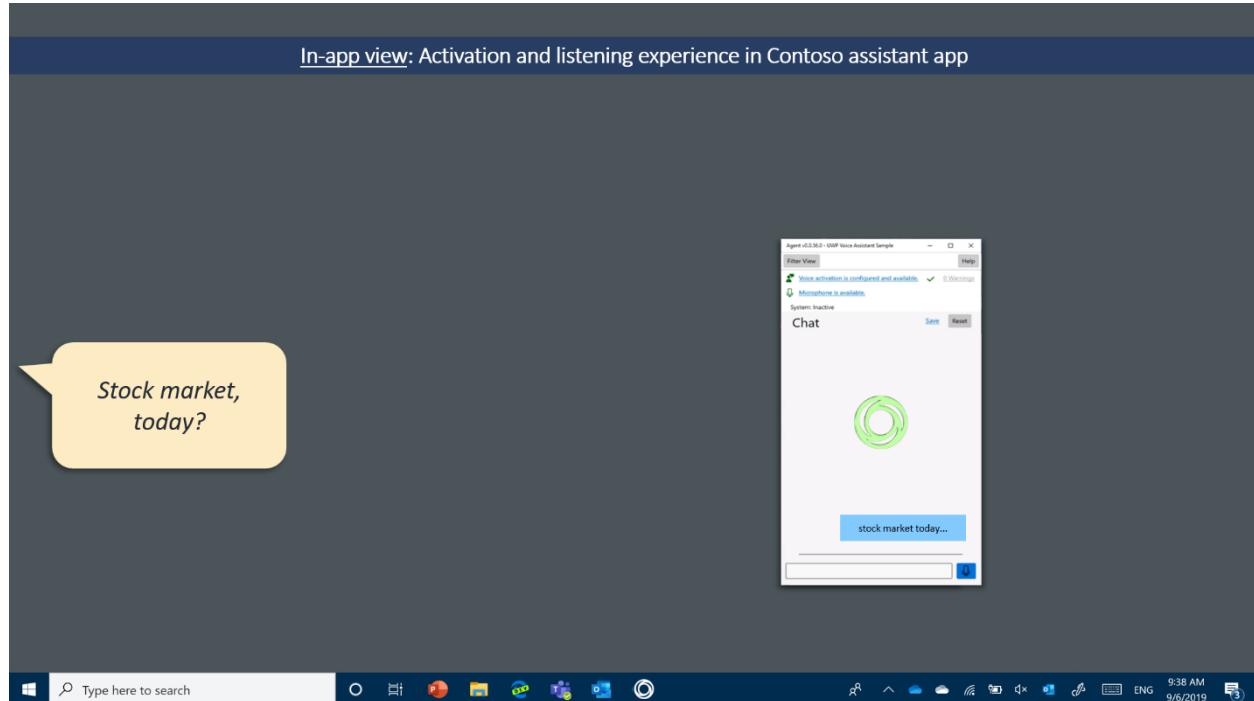
Design guidance for in-app voice activation

When the assistant app has focus, the customer intent is clearly to interact with the app, so all voice activation experiences should be handled by the main app view. This view may be resized by the customer. To help explain assistant shell interactions, the rest of this document uses the concrete example of a financial service assistant named Contoso. In this and subsequent diagrams, what the customer says will appear in cartoon speech bubbles on the left with assistant responses in cartoon bubbles on the right.

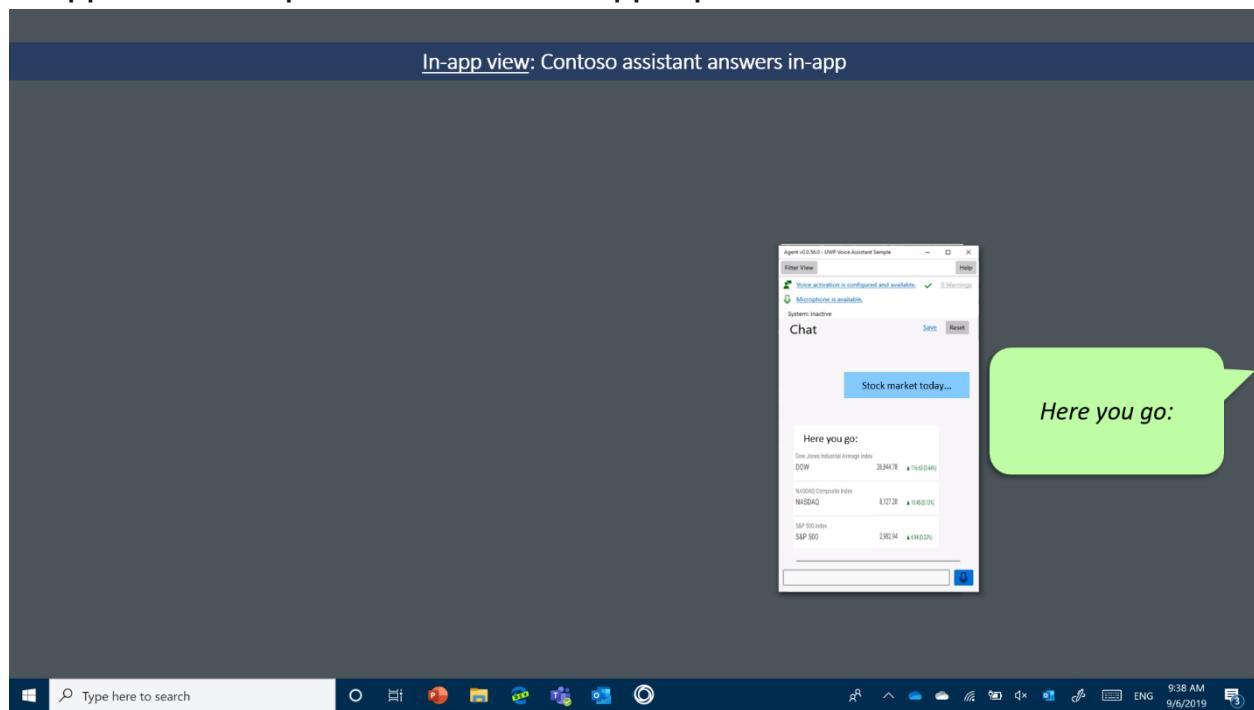
In-app view. Initial state when voice activation begins:



In-app view. After successful voice activation, listening experience begins:



In-app view. All responses remain in the app experience.



Design guidance for voice activation above lock

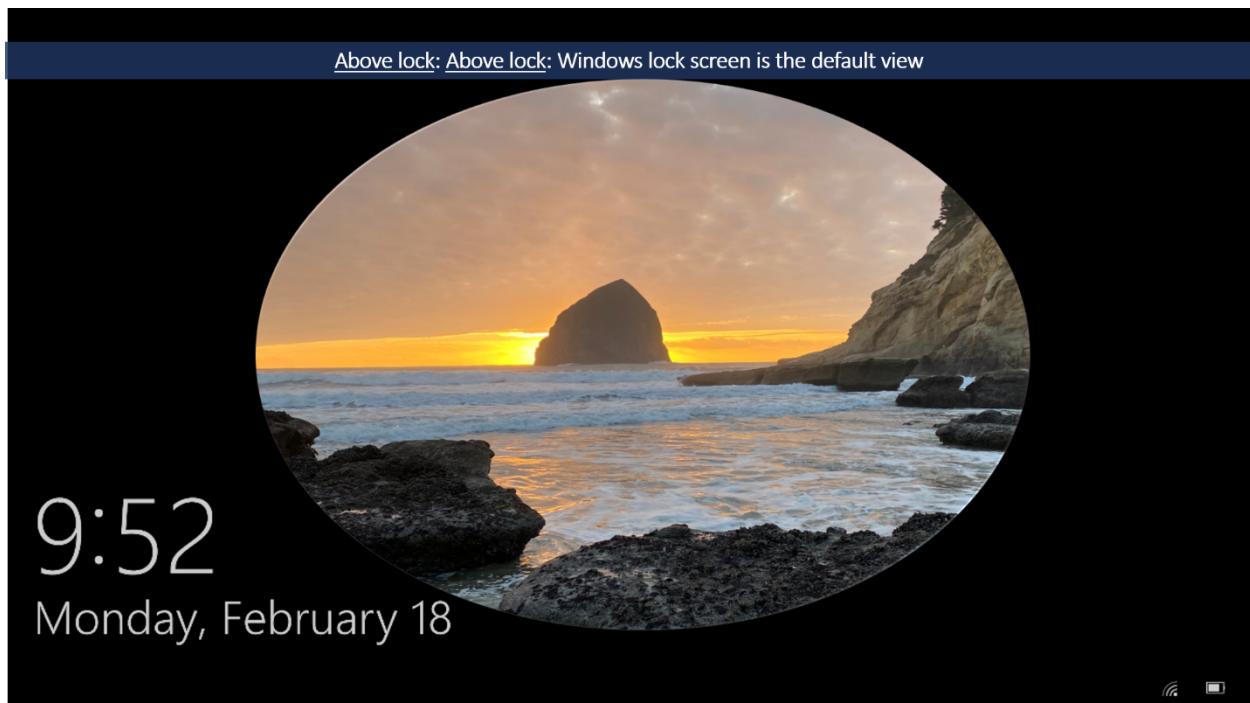
Available with 19H2, assistants built on Windows voice activation platform are available to answer above lock.

Customer opt-in

Voice activation above lock is always disabled by default. Customers opt-in through the Windows settings>Privacy>Voice Activation. For details on monitoring and prompting for this setting, see the [above lock implementation guide](#).

Not a lock-screen replacement

While notifications or other standard app lock-screen integration points remain available for the assistant, the Windows lock screen always defines the initial customer experience until a voice activation occurs. After voice activation is detected, the assistant app temporarily appears above the lock screen. To avoid customer confusion, when active above lock, the assistant app must never present UI to ask for any kind of credentials or log in.

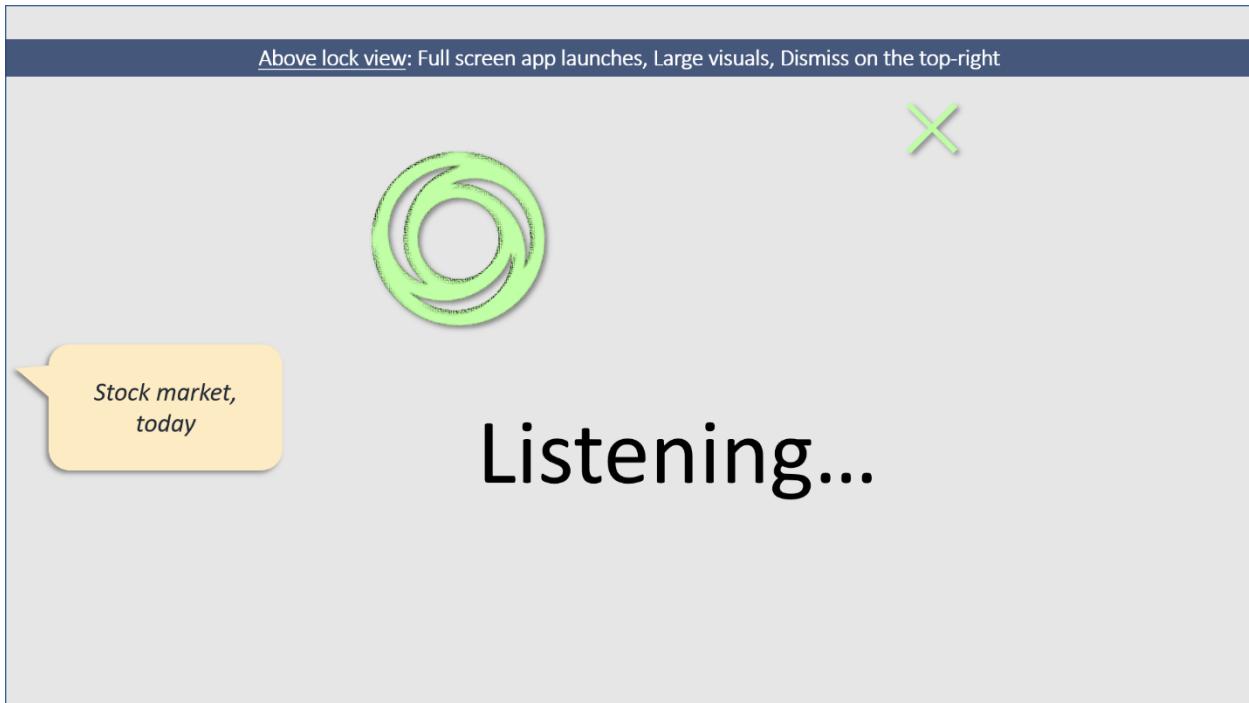


Above lock experience following voice activation

When the screen is on, the assistant app is full screen with no title bar above the lock screen. Larger visuals and strong voice descriptions with strong voice-primary interface allow for cases where the customer is too far away to read UI or has their hands busy with another (non-PC) task.

When the screen remains off, the assistant app could play an earcon to indicate the assistant is activating and provide a voice-only experience.

Above lock view: Full screen app launches, Large visuals, Dismiss on the top-right

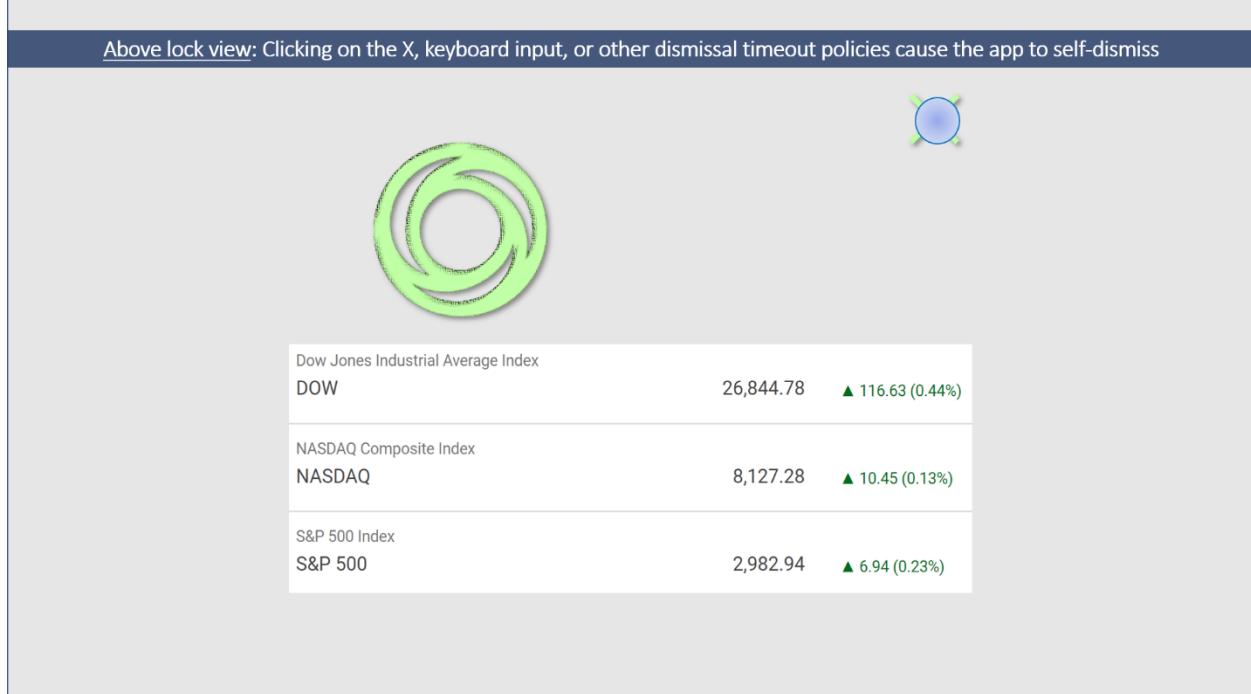


Dismissal policies

The assistant must implement the dismissal guidance in this section to make it easier for customers to log in the next time they want to use their Windows PC. Below are specific requirements, which the assistant must implement:

- **All assistant canvases that show above lock must contain an X in the top right that dismisses the assistant.**
- **Pressing any key must also dismiss the assistant app.** Keyboard input is a traditional lock app signal that the customer wants to log in. Therefore, any keyboard/text input should not be directed to the app. Instead, the app should self-dismiss when keyboard input is detected, so the customer can easily log in to their device.
- **If the screen goes off, the app must self-dismiss.** This ensures that the next time the customer uses their PC, the login screen will be ready and waiting for them.
- If the app is "in use", it may continue above lock. "in use" constitutes any input or output. For example, when streaming music or video the app may continue above lock. "Follow on" and other multturn dialog steps are permitted to keep the app above lock.
- **Implementation details on dismissing the application** can be found [in the above lock implementation guide](#).

Above lock view: Clicking on the X, keyboard input, or other dismissal timeout policies cause the app to self-dismiss



Above lock: Windows lock screen is easily available for customer to log-in



Privacy & security considerations above lock

Many PCs are portable but not always within customer reach. They may be briefly left in hotel rooms, airplane seats, or workspaces, where other people have physical access. If assistants that are enabled above lock aren't prepared, they can become subject to the class of so-called "[evil maid](#)" attacks.

Therefore, assistants should follow the guidance in this section to help keep experience secure. Interaction above lock occurs when the Windows user is unauthenticated. This

means that, in general, **input to the assistant should also be treated as unauthenticated**.

- Assistants should **implement a skill allowed list to identify skills that are confirmed secure and safe** to be accessed above lock.
- Speaker ID technologies can play a role in alleviating some risks, but Speaker ID is not a suitable replacement for Windows authentication.
- The skill allowed list should consider three classes of actions or skills:

Action class	Description	Examples (not a complete list)
Safe without authentication	General purpose information or basic app command and control	"What time is it?", "Play the next track"
Safe with Speaker ID	Impersonation risk, revealing personal information.	"What's my next appointment?", "Review my shopping list", "Answer the call"
Safe only after Windows authentication	High-risk actions that an attacker could use to harm the customer	"Buy more groceries", "Delete my (important) appointment", "Send a (mean) text message", "Launch a (nefarious) webpage"

For the case of Contoso, general information around public stock information is safe without authentication. Customer-specific information such as number of shares owned is likely safe with Speaker ID. However, buying or selling stocks should never be allowed without Windows authentication.

To further secure the experience, **weblinks, or other app-to-app launches will always be blocked by Windows until the customer signs in**. As a last resort mitigation, Microsoft reserves the right to remove an application from the allowed list of enabled assistants if a serious security issue is not addressed in a timely manner.

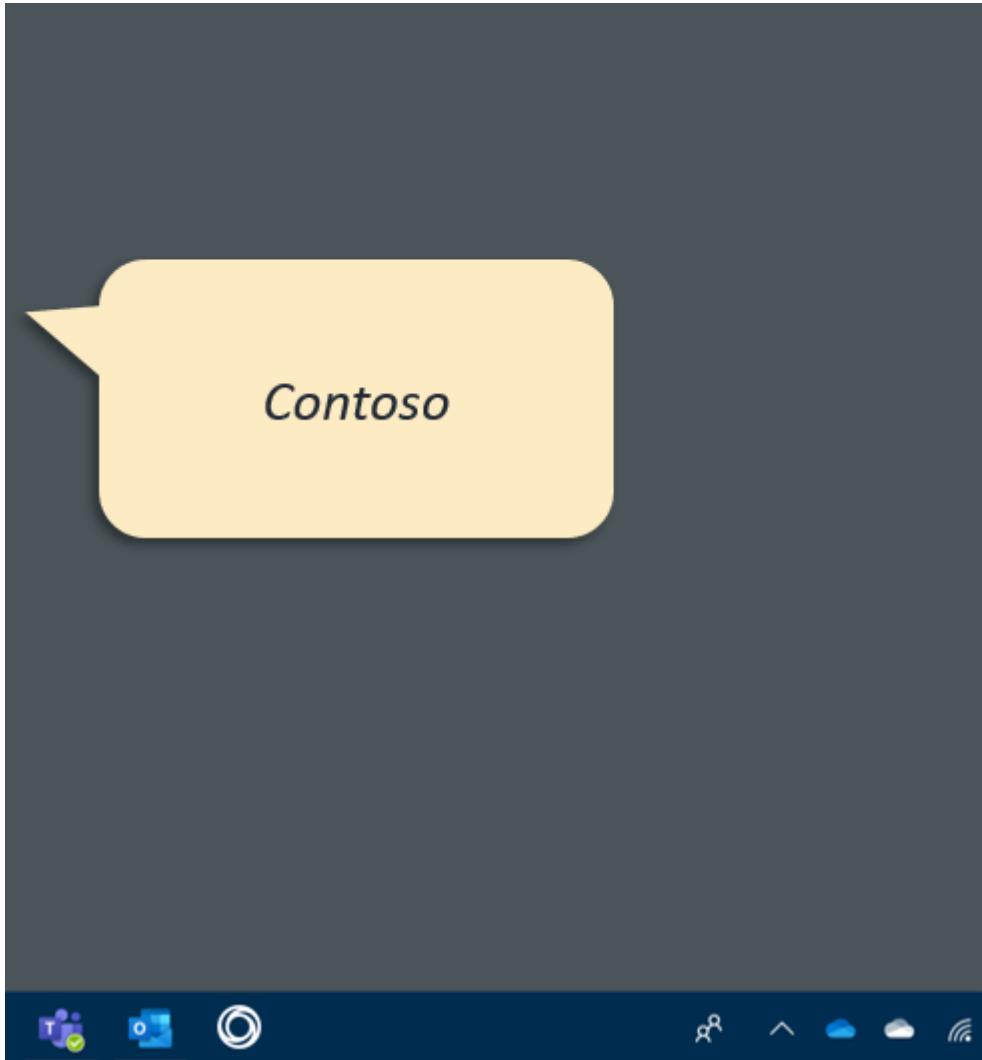
Design guidance for voice activation preview

Below lock, when the assistant app does *not* have focus, Windows provides a less intrusive voice activation UI to help keep the customer in flow. This is especially true for the case of false activations that would be highly disruptive if they launched the full app. The core idea is that each assistant has another home in the Shell, the assistant taskbar icon. When the request for background activation occurs, a small view above the assistant taskbar icon appears. Assistants should provide a small listening experience in this canvas. After processing the requests, assistants can choose to resize this view to show an in-context answer or to hand off their main app view to show larger, more detailed visuals.

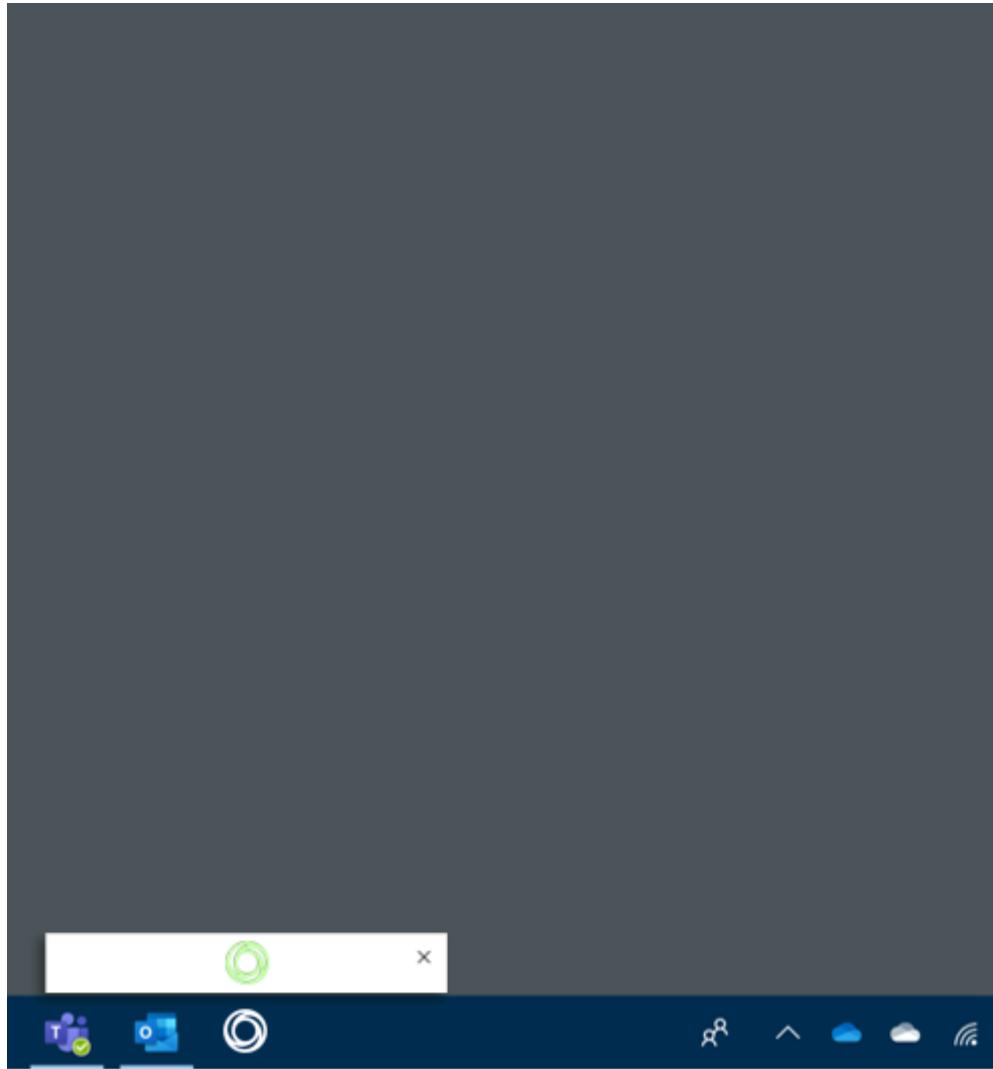
- To stay minimal, the preview does not have a title bar, so **the assistant must draw an X in the top right to allow customers to dismiss the view**. Refer to [Closing the Application](#) for the specific APIs to call when the dismiss button is pressed.
- To support voice activation previews, assistants may invite customers to pin the assistant to the taskbar during first run.

Voice activation preview: Initial state

The Contoso assistant has a home on the taskbar: their swirling, circular icon.



As activation progresses, the assistant requests background activation. The assistant is given a small preview pane (default width 408 and height: 248). If server-side voice activation determines the signal was a false positive, this view could be dismissed for minimal interruption.

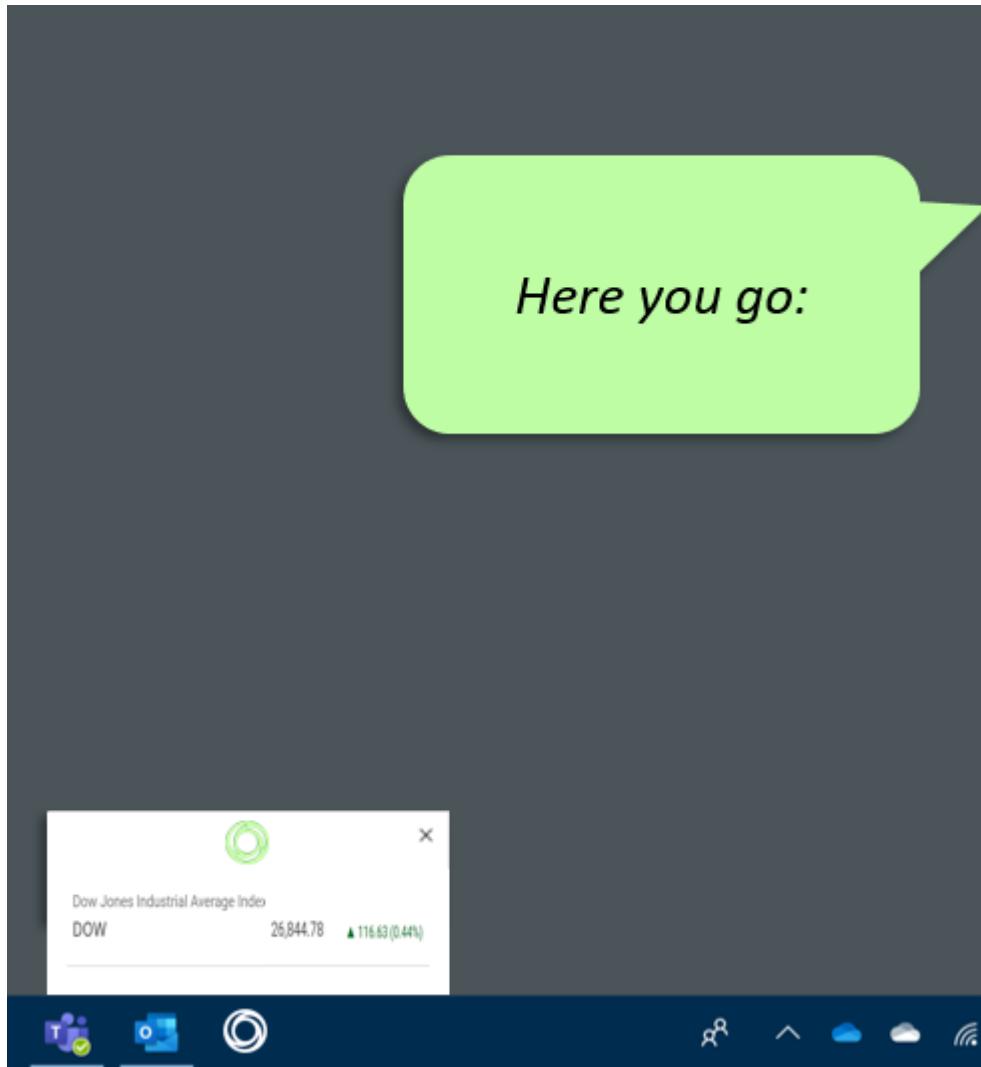


When final activation is confirmed, the assistant presents its listening UX. Assistant must always draw a dismiss X in the top right of the voice activation preview.

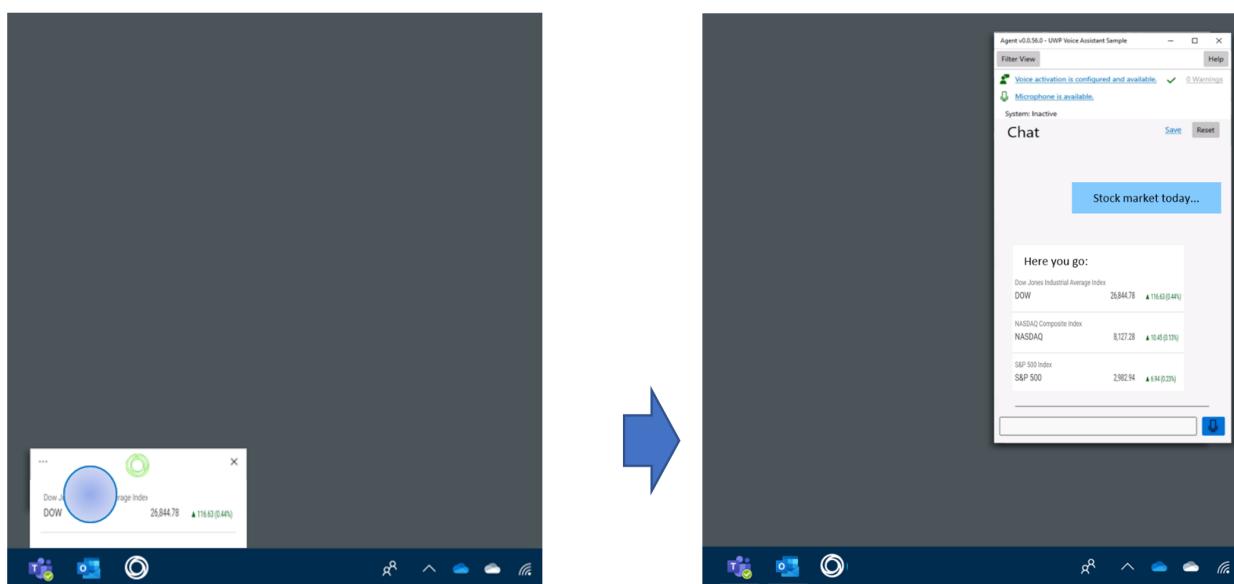
*Stock market,
today?*



Quick answers may appear in the voice activation preview. A TryResizeView will allow assistants to request different sizes.



Hand-off. At any point, the assistant may hand off to its main app view to provide more information, dialogue, or answers that require more screen real estate. Please refer to the [Transition from compact view to full view](#) section for implementation details.



Next steps

[Get started on developing your voice assistant](#)

Samples and FAQs

FAQ

The UWP Voice Assistant Sample

The UWP Voice Assistant Sample is a voice assistant on Windows that serves to

- demonstrate the use of the Windows ConversationalAgent APIs
- display best practices for a voice agent on Windows
- provide an adaptable app and reusable components for your MVP agent application
- show how Direct Line Speech, along with Bot Framework or Custom Commands, can be used together with the Windows ConversationalAgent APIs for an end-to-end voice agent experience

The documentation provided with the sample app walks through the code path of voice activation and agent management, from the prerequisites of startup through proper cleanup.

[Visit the GitHub repo for the UWP Sample](#)

How do I contact Microsoft for resources like Limited Access Feature tokens and keyword model files?

Contact winvoiceassistants@microsoft.com to request these resources.

My app is showing in a small window when I activate it by voice. How can I transition from the compact view to a full application window?

When your application is first activated by voice, it is started in a compact view. Please read the [Design guidance for voice activation preview](#) for guidance on the different views and transitions between them for voice assistants on Windows.

To make the transition from compact view to full app view, use the appView API

TryEnterViewModeAsync:

```
var appView = ApplicationView.GetForCurrentView(); await  
appView.TryEnterViewModeAsync(ApplicationViewMode.Default);
```

Why are voice assistant features on Windows only enabled for UWP applications?

Given the privacy risks associated with features like voice activation, the features of the UWP platform are necessary allow the voice assistant features on Windows to be sufficiently secure.

The UWP Voice Assistant Sample uses Direct Line Speech. Do I have to use Direct Line Speech for my voice assistant on Windows?

The UWP Sample Application was developed using Direct Line Speech and the Speech Services SDK as a demonstration of how to use a dialog service with the Windows Conversational Agent capability. However, you can use any service for local and cloud keyword verification, speech-to-text conversion, bot dialog, and text-to-speech conversion.

Language identification (preview)

Article • 02/02/2023 • 22 minutes to read

Language identification is used to identify languages spoken in audio when compared against a list of [supported languages](#).

Language identification (LID) use cases include:

- [Speech-to-text recognition](#) when you need to identify the language in an audio source and then transcribe it to text.
- [Speech translation](#) when you need to identify the language in an audio source and then translate it to another language.

For speech recognition, the initial latency is higher with language identification. You should only include this optional feature as needed.

Configuration options

Important

Language Identification (preview) APIs have been simplified in the Speech SDK version 1.25. The `SpeechServiceConnection_SingleLanguageIdPriority` and `SpeechServiceConnection_ContinuousLanguageIdPriority` properties have been removed and replaced by a single property `SpeechServiceConnection_LanguageIdMode`. Prioritizing between low latency and high accuracy is no longer necessary following recent model improvements. Now, you only need to select whether to run at-start or continuous Language Identification when doing continuous speech recognition or translation.

Whether you use language identification with [speech-to-text](#) or with [speech translation](#), there are some common concepts and configuration options.

- Define a list of [candidate languages](#) that you expect in the audio.
- Decide whether to use [at-start](#) or [continuous](#) language identification.

Then you make a [recognize once](#) or [continuous recognition](#) request to the Speech service.

Code snippets are included with the concepts described next. Complete samples for each use case are provided later.

Candidate languages

You provide candidate languages with the `AutoDetectSourceLanguageConfig` object, at least one of which is expected to be in the audio. You can include up to four languages for [at-start LID](#) or up to 10 languages for [continuous LID](#). The Speech service returns one of the candidate languages provided even if those languages weren't in the audio. For example, if `fr-FR` (French) and `en-US` (English) are provided as candidates, but German is spoken, either `fr-FR` or `en-US` would be returned.

You must provide the full locale with dash (-) separator, but language identification only uses one locale per base language. Don't include multiple locales (for example, "en-US" and "en-GB") for the same language.

C#

```
var autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.FromLanguages(new string[] { "en-US",
    "de-DE", "zh-CN" });
```

For more information, see [supported languages](#).

At-start and Continuous language identification

Speech supports both at-start and continuous language identification (LID).

Note

Continuous language identification is only supported with Speech SDKs in C#, C++, Java ([for speech to text only](#)), and Python.

- At-start LID identifies the language once within the first few seconds of audio. Use at-start LID if the language in the audio won't change. With at-start LID, a single language is detected and returned in less than 5 seconds.
- Continuous LID can identify multiple languages for the duration of the audio. Use continuous LID if the language in the audio could change. Continuous LID doesn't support changing languages within the same sentence. For example, if you're primarily speaking Spanish and insert some English words, it will not detect the language change per word.

You implement at-start LID or continuous LID by calling methods for [recognize once](#) or [continuous](#). Continuous LID is only supported with continuous recognition.

Recognize once or continuous

Language identification is completed with recognition objects and operations. You'll make a request to the Speech service for recognition of audio.

ⓘ Note

Don't confuse recognition with identification. Recognition can be used with or without language identification.

You'll either call the "recognize once" method, or the start and stop continuous recognition methods. You choose from:

- Recognize once with At-start LID. Continuous LID isn't supported for recognize once.
- Continuous recognition with at-start LID
- Continuous recognition with continuous LID

The `SpeechServiceConnection_LanguageIdMode` property is only required for continuous LID. Without it, the Speech service defaults to at-start lid. The supported values are "AtStart" for at-start LID or "Continuous" for continuous LID.

C#

```
// Recognize once with At-start LID. Continuous LID isn't supported for
// recognize once.
var result = await recognizer.RecognizeOnceAsync();

// Start and stop continuous recognition with At-start LID
await recognizer.StartContinuousRecognitionAsync();
await recognizer.StopContinuousRecognitionAsync();

// Start and stop continuous recognition with Continuous LID
speechConfig SetProperty(PropertyId.SpeechServiceConnection_LanguageIdMode,
    "Continuous");
await recognizer.StartContinuousRecognitionAsync();
await recognizer.StopContinuousRecognitionAsync();
```

Speech-to-text

You use Speech-to-text recognition when you need to identify the language in an audio source and then transcribe it to text. For more information, see [Speech-to-text overview](#).

ⓘ Note

Speech-to-text recognition with at-start language identification is supported with Speech SDKs in C#, C++, Python, Java, JavaScript, and Objective-C. Speech-to-text recognition with continuous language identification is only supported with Speech SDKs in C#, C++, Java, and Python.

Currently for speech-to-text recognition with continuous language identification, you must create a `SpeechConfig` from the

`wss:///{region}.stt.speech.microsoft.com/speech/universal/v2` endpoint string, as shown in code examples. In a future SDK release you won't need to set it.

See more examples of speech-to-text recognition with language identification on [GitHub](#).

Recognize once

C#

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

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

var autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.FromLanguages(
        new string[] { "en-US", "de-DE", "zh-CN" });

using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();
using (var recognizer = new SpeechRecognizer(
    speechConfig,
    autoDetectSourceLanguageConfig,
    audioConfig))
{
    var speechRecognitionResult = await recognizer.RecognizeOnceAsync();
    var autoDetectSourceLanguageResult =
        AutoDetectSourceLanguageResult.FromResult(speechRecognitionResult);
    var detectedLanguage = autoDetectSourceLanguageResult.Language;
}
```

Using Speech-to-text custom models

This sample shows how to use language detection with a custom endpoint. If the detected language is `en-US`, then the default model is used. If the detected language is

`fr-FR`, then the custom model endpoint is used. For more information, see [Deploy a Custom Speech model](#).

C#

```
var sourceLanguageConfigs = new SourceLanguageConfig[]
{
    SourceLanguageConfig.FromLanguage("en-US"),
    SourceLanguageConfig.FromLanguage("fr-FR", "The Endpoint Id for custom
model of fr-FR")
};
var autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.FromSourceLanguageConfigs(
        sourceLanguageConfigs);
```

Speech translation

You use Speech translation when you need to identify the language in an audio source and then translate it to another language. For more information, see [Speech translation overview](#).

ⓘ Note

Speech translation with language identification is only supported with Speech SDKs in C#, C++, and Python. Currently for speech translation with language identification, you must create a `SpeechConfig` from the `wss://{{region}}.stt.speech.microsoft.com/speech/universal/v2` endpoint string, as shown in code examples. In a future SDK release you won't need to set it.

See more examples of speech translation with language identification on [GitHub](#).

Recognize once

C#

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Translation;

public static async Task RecognizeOnceSpeechTranslationAsync()
{
    var region = "YourServiceRegion";
    // Currently the v2 endpoint is required. In a future SDK release
    you won't need to set it.
    var endpointString =
```

```

$"wss://{{region}}.stt.speech.microsoft.com/speech/universal/v2";
    var endpointUrl = new Uri(endpointString);

        var config = SpeechTranslationConfig.FromEndpoint(endpointUrl,
"YourSubscriptionKey");

        // Source language is required, but currently ignored.
        string fromLanguage = "en-US";
        speechTranslationConfig.SpeechRecognitionLanguage = fromLanguage;

        speechTranslationConfig.AddTargetLanguage("de");
        speechTranslationConfig.AddTargetLanguage("fr");

        var autoDetectSourceLanguageConfig =
AutoDetectSourceLanguageConfig.FromLanguages(new string[] { "en-US",
"de-DE", "zh-CN" });

        using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();

        using (var recognizer = new TranslationRecognizer(
            speechTranslationConfig,
            autoDetectSourceLanguageConfig,
            audioConfig))
    {

        Console.WriteLine("Say something or read from file...");
        var result = await
recognizer.RecognizeOnceAsync().ConfigureAwait(false);

        if (result.Reason == ResultReason.TranslatedSpeech)
        {
            var lidResult =
result.Properties.GetProperty(PropertyId.SpeechServiceConnection_AutoDet
ectSourceLanguageResult);

            Console.WriteLine($"RECOGNIZED in '{lidResult}': Text={
{result.Text}}");
            foreach (var element in result.Translations)
            {
                Console.WriteLine($"      TRANSLATED into '{element.Key}':
{element.Value}");
            }
        }
    }
}

```

Next steps

- Try the speech to text quickstart
- Improve recognition accuracy with custom speech

- Use batch transcription
-

Additional resources

Documentation

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Keyword recognition overview - Speech service - Azure Cognitive Services](#)

An overview of the features, capabilities, and restrictions for keyword recognition by using the Speech Software Development Kit (SDK).

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[Configure the Speech CLI datastore - Speech service - Azure Cognitive Services](#)

Learn how to configure the Speech CLI datastore.

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

How to use the audio input stream

Article • 12/01/2022 • 2 minutes to read

The Speech SDK provides a way to stream audio into the recognizer as an alternative to microphone or file input.

The following steps are required when you use audio input streams:

- Identify the format of the audio stream. The format must be supported by the Speech SDK and the Azure Cognitive Services Speech service. Currently, only the following configuration is supported:

Audio samples are:

- PCM format (int-16)
- One channel
- 16 bits per sample, 8,000 or 16,000 samples per second (16,000 bytes or 32,000 bytes per second)
- Two-block aligned (16 bit including padding for a sample)

The corresponding code in the SDK to create the audio format looks like this example:

C#

```
byte channels = 1;
byte bitsPerSample = 16;
int samplesPerSecond = 16000; // or 8000
var audioFormat = AudioStreamFormat.GetWaveFormatPCM(samplesPerSecond,
bitsPerSample, channels);
```

- Make sure that your code provides the RAW audio data according to these specifications. Also, make sure that 16-bit samples arrive in little-endian format. Signed samples are also supported. If your audio source data doesn't match the supported formats, the audio must be transcoded into the required format.
- Create your own audio input stream class derived from `PullAudioInputStreamCallback`. Implement the `Read()` and `Close()` members. The exact function signature is language-dependent, but the code looks similar to this code sample:

C#

```
public class ContosoAudioStream : PullAudioInputStreamCallback {
    ContosoConfig config;
```

```
public ContosoAudioStream(const ContosoConfig& config) {
    this.config = config;
}

public int Read(byte[] buffer, uint size) {
    // Returns audio data to the caller.
    // E.g., return read(config.YYY, buffer, size);
}

public void Close() {
    // Close and clean up resources.
}
};
```

- Create an audio configuration based on your audio format and input stream. Pass in both your regular speech configuration and the audio input configuration when you create your recognizer. For example:

C#

```
var audioConfig = AudioConfig.FromStreamInput(new
ContosoAudioStream(config), audioFormat);

var speechConfig = SpeechConfig.FromSubscription(...);
var recognizer = new SpeechRecognizer(speechConfig, audioConfig);

// Run stream through recognizer.
var result = await recognizer.RecognizeOnceAsync();

var text = result.GetText();
```

Next steps

- [Create a free Azure account ↗](#)
- [See how to recognize speech in C#](#)

How to use compressed input audio

Article • 04/27/2022 • 18 minutes to read

[Reference documentation](#) | [Package \(NuGet\)](#) ↗ | [Additional Samples on GitHub](#) ↗

The Speech SDK and Speech CLI use GStreamer to support different kinds of input audio formats. GStreamer decompresses the audio before it's sent over the wire to the Speech service as raw PCM.

The default audio streaming format is WAV (16 kHz or 8 kHz, 16-bit, and mono PCM). Outside WAV and PCM, the following compressed input formats are also supported through GStreamer:

- MP3
- OPUS/OGG
- FLAC
- ALAW in WAV container
- MULAW in WAV container
- ANY for MP4 container or unknown media format

GStreamer configuration

The Speech SDK can use [GStreamer](#) ↗ to handle compressed audio. For licensing reasons, GStreamer binaries aren't compiled and linked with the Speech SDK. You need to install some dependencies and plug-ins.

GStreamer binaries must be in the system path so that they can be loaded by the Speech SDK at runtime. For example, on Windows, if the Speech SDK finds `libgstreamer-1.0-0.dll` or `gstreamer-1.0-0.dll` (for the latest GStreamer) during runtime, it means the GStreamer binaries are in the system path.

Choose a platform for installation instructions.

Windows

Make sure that packages of the same platform (x64 or x86) are installed. For example, if you installed the x64 package for Python, you need to install the x64 GStreamer package. The following instructions are for the x64 packages.

1. Create the folder `c:\gstreamer`.
2. Download the [installer](#) ↗ .

3. Copy the installer to c:\gstreamer.
4. Open PowerShell as an administrator.
5. Run the following command in PowerShell:

```
PowerShell
```

```
cd c:\gstreamer  
msiexec /passive INSTALLLEVEL=1000 INSTALLDIR=C:\gstreamer /i  
gstreamer-1.0-msvc-x86_64-1.18.3.msi
```

6. Add the system variable `GST_PLUGIN_PATH` with "C:\gstreamer\1.0\msvc_x86_64\lib\gstreamer-1.0" as the variable value.
7. Add the system variable `GSTREAMER_ROOT_X86_64` with "C:\gstreamer\1.0\msvc_x86_64" as the variable value.
8. Edit the system `PATH` variable to add "C:\gstreamer\1.0\msvc_x86_64\bin" as a new entry.
9. Reboot the machine.

For more information about GStreamer, see [Windows installation instructions](#).

Example

To configure the Speech SDK to accept compressed audio input, create `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class that specifies the compression format of the stream. Find related sample code snippets in [About the Speech SDK audio input stream API](#).

Let's assume that you have an input stream class called `pullStream` and are using OPUS/OGG. Your code might look like this:

```
C#
```

```
using Microsoft.CognitiveServices.Speech;  
using Microsoft.CognitiveServices.Speech.Audio;  
  
// ... omitted for brevity  
  
var speechConfig =  
    SpeechConfig.FromSubscription(  
        "YourSubscriptionKey",  
        "YourServiceRegion");
```

```
// Create an audio config specifying the compressed
// audio format and the instance of your input stream class.
var pullStream = AudioInputStream.CreatePullStream(
    AudioStreamFormat.GetCompressedFormat(AudioStreamContainerFormat.OGG_OPUS));
var audioConfig = AudioConfig.FromStreamInput(pullStream);

using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
var result = await recognizer.RecognizeOnceAsync();

var text = result.Text;
```

Next steps

- [Try the speech to text quickstart](#)
- [Improve recognition accuracy with custom speech](#)

Select an audio input device with the Speech SDK

Article • 08/30/2022 • 5 minutes to read

This article describes how to obtain the IDs of the audio devices connected to a system. These IDs can then be used in the Speech SDK to select the audio input. You configure the audio device through the `AudioConfig` object:

C++

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

C#

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

Python

```
audio_config = AudioConfig(device_name="<device id>");
```

Objective-C

```
audioConfig = AudioConfiguration.FromMicrophoneInput("<device id>");
```

Java

```
audioConfig = AudioConfiguration.fromMicrophoneInput("<device id>");
```

JavaScript

```
audioConfig = AudioConfiguration.fromMicrophoneInput("<device id>");
```

ⓘ Note

Microphone use isn't 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++:

C++

```
#include <stdio>
#include <mmdeviceapi.h>

#include <FunctionDiscoverykeys_devpkey.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) = 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 the 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 the 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 the endpoint friendly name and endpoint ID.
        printf("Endpoint %d: \"%S\" (%S)\n", i, varName.pwszVal, pwszID);

        CoTaskMemFree(pwszID);
        pwszID = NULL;
        PropVariantClear(&varName);
    }

    Exit:
    CoTaskMemFree(pwszID);
    pwszID = NULL;
    PropVariantClear(&varName);
    SAFE_RELEASE(pEnumerator);
    SAFE_RELEASE(pCollection);
    SAFE_RELEASE(pEndpoint);
    SAFE_RELEASE(pProps);
}

```

In C#, you can use the [NAudio](#) library to access the CoreAudio API and enumerate devices as follows:

C#

```
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), you can obtain audio input devices by using the `Id()` property of the corresponding `DeviceInformation` object.

The following code samples show how to do this step in C++ and C#:

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());
        }
    });
}

```

C#

```

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 by 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 by 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.

Objective-C

```
#import <Foundation/Foundation.h>
#import <CoreAudio/CoreAudio.h>

CFArrayRef 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(kCFAlocatorDefault, 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 = kAudioDevicePropertyNameCFString;
    status = AudioObjectGetPropertyData(audioDevices[i],
&propertyAddress, 0, NULL, &dataSize, &deviceName);
    if (kAudioHardwareNoError != status) {
        fprintf(stderr, "AudioObjectGetPropertyData
(kAudioDevicePropertyNameCFString) 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);
        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(kCFAlocatorDefault,
                           (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(kCFAlocatorDefault, 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 isn't supported on iOS. Apps that use the SDK can influence audio routing through the [AVAudioSession](#) Framework.

For example, the instruction

Objective-C

```
[[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 samples on GitHub](#)
- [Customize acoustic models](#)
- [Customize language models](#)

Role-based access control for Speech resources

Article • 04/06/2022 • 3 minutes to read

You can manage access and permissions to your Speech resources with Azure role-based access control (Azure RBAC). Assigned roles can vary across Speech resources. For example, you can assign a role to a Speech resource that should only be used to train a Custom Speech model. You can assign another role to a Speech resource that is used to transcribe audio files. Depending on who can access each Speech resource, you can effectively set a different level of access per application or user. For more information on Azure RBAC, see the [Azure RBAC documentation](#).

ⓘ Note

A Speech resource can inherit or be assigned multiple roles. The final level of access to this resource is a combination of all roles permissions from the operation level.

Roles for Speech resources

A role definition is a collection of permissions. When you create a Speech resource, the built-in roles in this table are assigned by default.

Role	Can list resource keys	Access to data, models, and endpoints
Owner	Yes	View, create, edit, and delete
Contributor	Yes	View, create, edit, and delete
Cognitive Services Contributor	Yes	View, create, edit, and delete
Cognitive Services User	Yes	View, create, edit, and delete
Cognitive Services Speech Contributor	No	View, create, edit, and delete
Cognitive Services Speech User	No	View only
Cognitive Services Data Reader (Preview)	No	View only

ⓘ Important

Whether a role can list resource keys is important for [Speech Studio authentication](#). To list resource keys, a role must have permission to run the `Microsoft.CognitiveServices/accounts/listKeys/action` operation. Please note that if key authentication is disabled in the Azure Portal, then none of the roles can list keys.

Keep the built-in roles if your Speech resource can have full read and write access to the projects.

For finer-grained resource access control, you can [add or remove roles](#) using the Azure portal. For example, you could create a custom role with permission to upload Custom Speech datasets, but without permission to deploy a Custom Speech model to an endpoint.

Authentication with keys and tokens

The [roles](#) define what permissions you have. Authentication is required to use the Speech resource.

To authenticate with Speech resource keys, all you need is the key and region. To authenticate with an Azure AD token, the Speech resource must have a [custom subdomain](#) and use a [private endpoint](#). The Speech service uses custom subdomains with private endpoints only.

Speech SDK authentication

For the SDK, you configure whether to authenticate with a Speech resource key or Azure AD token. For details, see [Azure Active Directory Authentication with the Speech SDK](#).

Speech Studio authentication

Once you're signed into [Speech Studio](#), you select a subscription and Speech resource. You don't choose whether to authenticate with a Speech resource key or Azure AD token. Speech Studio gets the key or token automatically from the Speech resource. If one of the assigned [roles](#) has permission to list resource keys, Speech Studio will authenticate with the key. Otherwise, Speech Studio will authenticate with the Azure AD token.

If Speech Studio uses your Azure AD token, but the Speech resource doesn't have a custom subdomain and private endpoint, then you can't use some features in Speech

Studio. In this case, for example, the Speech resource can be used to train a Custom Speech model, but you can't use a Custom Speech model to transcribe audio files.

Authentication credential	Feature availability
Speech resource key	Full access limited only by the assigned role permissions.
Azure AD token with custom subdomain and private endpoint	Full access limited only by the assigned role permissions.
Azure AD token without custom subdomain and private endpoint (not recommended)	Features are limited. For example, the Speech resource can be used to train a Custom Speech model or Custom Neural Voice. But you can't use a Custom Speech model or Custom Neural Voice.

Next steps

- [Azure Active Directory Authentication with the Speech SDK](#).
- [Speech service encryption of data at rest](#).

Azure Active Directory Authentication with the Speech SDK

Article • 02/02/2023 • 13 minutes to read

When using the Speech SDK to access the Speech service, there are three authentication methods available: service keys, a key-based token, and Azure Active Directory (Azure AD). This article describes how to configure a Speech resource and create a Speech SDK configuration object to use Azure AD for authentication.

This article shows how to use Azure AD authentication with the Speech SDK. You'll learn how to:

- ✓ Create a Speech resource
- ✓ Configure the Speech resource for Azure AD authentication
- ✓ Get an Azure AD access token
- ✓ Create the appropriate SDK configuration object.

Create a Speech resource

To create a Speech resource in the [Azure portal](#), see [Get the keys for your resource](#)

Configure the Speech resource for Azure AD authentication

To configure your Speech resource for Azure AD authentication, create a custom domain name and assign roles.

Create a custom domain name

Follow these steps to create a [custom subdomain name for Cognitive Services](#) for your Speech resource.

⊗ Caution

When you turn on a custom domain name, the operation is **not reversible**. The only way to go back to the **regional name** is to create a new Speech resource.

If your Speech resource has a lot of associated custom models and projects created via [Speech Studio](#), we strongly recommend trying the configuration with a test

resource before you modify the resource used in production.

Azure portal

To create a custom domain name using the Azure portal, follow these steps:

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the required Speech resource.
3. In the **Resource Management** group on the left pane, select **Networking**.
4. On the **Firewalls and virtual networks** tab, select **Generate Custom Domain Name**. A new right panel appears with instructions to create a unique custom subdomain for your resource.
5. In the **Generate Custom Domain Name** panel, enter a custom domain name. Your full custom domain will look like: `https://{your custom name}.cognitiveservices.azure.com`.
Remember that after you create a custom domain name, it *cannot* be changed.
After you've entered your custom domain name, select **Save**.
6. After the operation finishes, in the **Resource management** group, select **Keys and Endpoint**. Confirm that the new endpoint name of your resource starts this way: `https://{your custom name}.cognitiveservices.azure.com`.

Assign roles

For Azure AD authentication with Speech resources, you need to assign either the *Cognitive Services Speech Contributor* or *Cognitive Services Speech User* role.

You can assign roles to the user or application using the [Azure portal](#) or [PowerShell](#).

Get an Azure AD access token

To get an Azure AD access token in C#, use the [Azure Identity Client Library](#).

Here's an example of using Azure Identity to get an Azure AD access token from an interactive browser:

```
c#
```

```
TokenRequestContext context = new Azure.Core.TokenRequestContext(new
string[] { "https://cognitiveservices.azure.com/.default" });
InteractiveBrowserCredential browserCredential = new
InteractiveBrowserCredential();
var browserToken = browserCredential.GetToken(context);
string aadToken = browserToken.Token;
```

The token context must be set to "https://cognitiveservices.azure.com/.default".

Get the Speech resource ID

You need your Speech resource ID to make SDK calls using Azure AD authentication.

ⓘ Note

For Intent Recognition use your LUIS Prediction resource ID.

Azure portal

To get the resource ID in the Azure portal:

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select a Speech resource.
3. In the **Resource Management** group on the left pane, select **Properties**.
4. Copy the **Resource ID**

Create the Speech SDK configuration object

With an Azure AD access token, you can now create a Speech SDK configuration object.

The method of providing the token, and the method to construct the corresponding Speech SDK `Config` object varies by the object you'll be using.

SpeechRecognizer, SpeechSynthesizer, IntentRecognizer, ConversationTranscriber, and SourceLanguageRecognizer

For `SpeechRecognizer`, `SpeechSynthesizer`, `IntentRecognizer`, `ConversationTranscriber`, and `SourceLanguageRecognizer` objects, build the authorization token from the resource

ID and the Azure AD access token and then use it to create a `SpeechConfig` object.

C#

```
string resourceId = "Your Resource ID";
string aadToken = "Your Azure AD access token";
string region = "Your Speech Region";

// You need to include the "aad#" prefix and the "#" (hash) separator
// between resource ID and AAD access token.
var authorizationToken = $"aad#{resourceId}#{aadToken}";
var speechConfig = SpeechConfig.FromAuthorizationToken(authorizationToken,
region);
```

TranslationRecognizer

For the `TranslationRecognizer`, build the authorization token from the resource ID and the Azure AD access token and then use it to create a `SpeechTranslationConfig` object.

C#

```
string resourceId = "Your Resource ID";
string aadToken = "Your Azure AD access token";
string region = "Your Speech Region";

// You need to include the "aad#" prefix and the "#" (hash) separator
// between resource ID and AAD access token.
var authorizationToken = $"aad#{resourceId}#{aadToken}";
var speechConfig =
SpeechTranslationConfig.FromAuthorizationToken(authorizationToken, region);
```

DialogServiceConnector

For the `DialogServiceConnection` object, build the authorization token from the resource ID and the Azure AD access token and then use it to create a `CustomCommandsConfig` or a `BotFrameworkConfig` object.

C#

```
string resourceId = "Your Resource ID";
string aadToken = "Your Azure AD access token";
string region = "Your Speech Region";
string appId = "Your app ID";

// You need to include the "aad#" prefix and the "#" (hash) separator
// between resource ID and AAD access token.
var authorizationToken = $"aad#{resourceId}#{aadToken}";
```

```
var customCommandsConfig =  
    CustomCommandsConfig.FromAuthorizationToken(appId, authorizationToken,  
    region);
```

VoiceProfileClient

To use the `VoiceProfileClient` with Azure AD authentication, use the custom domain name created above.

C#

```
string customDomainName = "Your Custom Name";  
string hostName =  
    $"https://{{customDomainName}}.cognitiveservices.azure.com/";  
string token = "Your Azure AD access token";  
  
var config = SpeechConfig.FromHost(new Uri(hostName));  
  
// You need to include the "aad#" prefix and the "#" (hash) separator  
// between resource ID and AAD access token.  
var authorizationToken = $"aad#{resourceId}#{aadToken}";  
config.AuthorizationToken = authorizationToken;
```

ⓘ Note

The `ConversationTranslator` doesn't support Azure AD authentication.

Additional resources

📘 Documentation

[azure.cognitiveservices.speech.speech module](#)

Classes related to recognizing text from speech, synthesizing speech from text, and general classes used in the various recognizers.

[azure.cognitiveservices.speech.RecognitionResult class](#)

Detailed information about the result of a recognition operation.

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[azure.cognitiveservices.speech.Recognizer class](#)

Base class for different recognizers

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Deploy a Custom Speech model - Speech service - Azure Cognitive Services](#)

Learn how to deploy Custom Speech models.

[azure.cognitiveservices.speech.audio.AudioInputStream class](#)

Base class for Input Streams

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

Learning certificate

[Microsoft Certified: Identity and Access Administrator Associate - Certifications](#)

The Microsoft identity and access administrator designs, implements, and operates an organization's identity and access management systems by using Microsoft Azure Active Directory (Azure AD), part of Microsoft Entra. They configure and manage authentication and authorization of identities for...

Use Speech service through a private endpoint

Article • 11/30/2022 • 24 minutes to read

Azure Private Link lets you connect to services in Azure by using a [private endpoint](#). A private endpoint is a private IP address that's accessible only within a specific [virtual network](#) and subnet.

This article explains how to set up and use Private Link and private endpoints with Speech Services in Azure Cognitive Services. This article then describes how to remove private endpoints later, but still use the Speech resource.

Note

Before you proceed, review [how to use virtual networks with Cognitive Services](#).

Setting up a Speech resource for the private endpoint scenarios requires performing the following tasks:

1. [Create a custom domain name](#)
2. [Turn on private endpoints](#)
3. [Adjust existing applications and solutions](#)

Private endpoints and Virtual Network service endpoints

Azure provides private endpoints and Virtual Network service endpoints for traffic that tunnels via the [private Azure backbone network](#). The purpose and underlying technologies of these endpoint types are similar. But there are differences between the two technologies. We recommend that you learn about the pros and cons of both before you design your network.

There are a few things to consider when you decide which technology to use:

- Both technologies ensure that traffic between the virtual network and the Speech resource doesn't travel over the public internet.
- A private endpoint provides a dedicated private IP address for your Speech resource. This IP address is accessible only within a specific virtual network and

subnet. You have full control of the access to this IP address within your network infrastructure.

- Virtual Network service endpoints don't provide a dedicated private IP address for the Speech resource. Instead, they encapsulate all packets sent to the Speech resource and deliver them directly over the Azure backbone network.
- Both technologies support on-premises scenarios. By default, when they use Virtual Network service endpoints, Azure service resources secured to virtual networks can't be reached from on-premises networks. But you can [change that behavior](#).
- Virtual Network service endpoints are often used to restrict the access for a Speech resource based on the virtual networks from which the traffic originates.
- For Cognitive Services, enabling the Virtual Network service endpoint forces the traffic for all Cognitive Services resources to go through the private backbone network. That requires explicit network access configuration. (For more information, see [Configure virtual networks and the Speech resource networking settings](#).) Private endpoints don't have this limitation and provide more flexibility for your network configuration. You can access one resource through the private backbone and another through the public internet by using the same subnet of the same virtual network.
- Private endpoints incur [extra costs](#). Virtual Network service endpoints are free.
- Private endpoints require [extra DNS configuration](#).
- One Speech resource can work simultaneously with both private endpoints and Virtual Network service endpoints.

We recommend that you try both endpoint types before you make a decision about your production design.

For more information, see these resources:

- [Azure Private Link and private endpoint documentation](#)
- [Virtual Network service endpoints documentation](#)

This article describes the usage of the private endpoints with Speech service. Usage of the VNet service endpoints is described [here](#).

Create a custom domain name

Caution

A Speech resource with a custom domain name enabled uses a different way to interact with Speech service. You might have to adjust your application code for

both of these scenarios: **with private endpoint** and **without private endpoint**.

Follow these steps to create a [custom subdomain name for Cognitive Services](#) for your Speech resource.

✖ Caution

When you turn on a custom domain name, the operation is **not reversible**. The only way to go back to the **regional name** is to create a new Speech resource.

If your Speech resource has a lot of associated custom models and projects created via [Speech Studio](#), we strongly recommend trying the configuration with a test resource before you modify the resource used in production.

Azure portal

To create a custom domain name using the Azure portal, follow these steps:

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the required Speech resource.
3. In the **Resource Management** group on the left pane, select **Networking**.
4. On the **Firewalls and virtual networks** tab, select **Generate Custom Domain Name**. A new right panel appears with instructions to create a unique custom subdomain for your resource.
5. In the **Generate Custom Domain Name** panel, enter a custom domain name. Your full custom domain will look like: `https://{{your custom name}}.cognitiveservices.azure.com`.

Remember that after you create a custom domain name, it *cannot* be changed.

After you've entered your custom domain name, select **Save**.

6. After the operation finishes, in the **Resource management** group, select **Keys and Endpoint**. Confirm that the new endpoint name of your resource starts this way: `https://{{your custom name}}.cognitiveservices.azure.com`.

Turn on private endpoints

We recommend using the [private DNS zone](#) attached to the virtual network with the necessary updates for the private endpoints. You can create a private DNS zone during the provisioning process. If you're using your own DNS server, you might also need to change your DNS configuration.

Decide on a DNS strategy *before* you provision private endpoints for a production Speech resource. And test your DNS changes, especially if you use your own DNS server.

Use one of the following articles to create private endpoints. These articles use a web app as a sample resource to make available through private endpoints.

- [Create a private endpoint by using the Azure portal](#)
- [Create a private endpoint by using Azure PowerShell](#)
- [Create a private endpoint by using Azure CLI](#)

Use these parameters instead of the parameters in the article that you chose:

Setting	Value
Resource type	<code>Microsoft.CognitiveServices/accounts</code>
Resource	<code><your-speech-resource-name></code>
Target sub-resource	<code>account</code>

DNS for private endpoints: Review the general principles of [DNS for private endpoints in Cognitive Services resources](#). Then confirm that your DNS configuration is working correctly by performing the checks described in the following sections.

Resolve DNS from the virtual network

This check is *required*.

Follow these steps to test the custom DNS entry from your virtual network:

1. Log in to a virtual machine located in the virtual network to which you've attached your private endpoint.
2. Open a Windows command prompt or a Bash shell, run `nslookup`, and confirm that it successfully resolves your resource's custom domain name.

`dos`

```
C:\>nslookup my-private-link-speech.cognitiveservices.azure.com
Server: UnKnown
Address: 168.63.129.16

Non-authoritative answer:
Name: my-private-link-speech.privatelink.cognitiveservices.azure.com
Address: 172.28.0.10
Aliases: my-private-link-speech.cognitiveservices.azure.com
```

3. Confirm that the IP address matches the IP address of your private endpoint.

Resolve DNS from other networks

Perform this check only if you've turned on either the **All networks** option or the **Selected Networks and Private Endpoints** access option in the **Networking** section of your resource.

If you plan to access the resource by using only a private endpoint, you can skip this section.

1. Log in to a computer attached to a network that's allowed to access the resource.
2. Open a Windows command prompt or Bash shell, run `nslookup`, and confirm that it successfully resolves your resource's custom domain name.

```
dos

C:\>nslookup my-private-link-speech.cognitiveservices.azure.com
Server: UnKnown
Address: fe80::1

Non-authoritative answer:
Name: vnetproxyv1-weu-prod.westeurope.cloudapp.azure.com
Address: 13.69.67.71
Aliases: my-private-link-speech.cognitiveservices.azure.com
         my-private-link-
         speech.privatelink.cognitiveservices.azure.com
         westeurope.prod.vnet.cog.trafficmanager.net
```

ⓘ Note

The resolved IP address points to a virtual network proxy endpoint, which dispatches the network traffic to the private endpoint for the Cognitive Services resource. The behavior will be different for a resource with a custom domain name but *without* private endpoints. See [this section](#) for details.

Adjust an application to use a Speech resource with a private endpoint

A Speech resource with a custom domain interacts with Speech Services in a different way. This is true for a custom-domain-enabled Speech resource both with and without private endpoints. Information in this section applies to both scenarios.

Follow instructions in this section to adjust existing applications and solutions to use a Speech resource with a custom domain name and a private endpoint turned on.

A Speech resource with a custom domain name and a private endpoint turned on uses a different way to interact with Speech Services. This section explains how to use such a resource with the Speech Services REST APIs and the [Speech SDK](#).

Note

A Speech resource without private endpoints that uses a custom domain name also has a special way of interacting with Speech Services. This way differs from the scenario of a Speech resource that uses a private endpoint. This is important to consider because you may decide to remove private endpoints later. See [Adjust an application to use a Speech resource without private endpoints](#) later in this article.

Speech resource with a custom domain name and a private endpoint: Usage with the REST APIs

We'll use `my-private-link-speech.cognitiveservices.azure.com` as a sample Speech resource DNS name (custom domain) for this section.

Speech service has REST APIs for [Speech-to-text](#) and [Text-to-speech](#). Consider the following information for the private-endpoint-enabled scenario.

Speech-to-text has two REST APIs. Each API serves a different purpose, uses different endpoints, and requires a different approach when you're using it in the private-endpoint-enabled scenario.

The Speech-to-text REST APIs are:

- [Speech-to-text REST API](#), which is used for [Batch transcription](#) and [Custom Speech](#).
- [Speech-to-text REST API for short audio](#), which is used for online transcription

Usage of the Speech-to-text REST API for short audio and the Text-to-speech REST API in the private endpoint scenario is the same. It's equivalent to the [Speech SDK case](#) described later in this article.

Speech-to-text REST API uses a different set of endpoints, so it requires a different approach for the private-endpoint-enabled scenario.

The next subsections describe both cases.

Speech-to-text REST API

Usually, Speech resources use [Cognitive Services regional endpoints](#) for communicating with the [Speech-to-text REST API](#). These resources have the following naming format:

```
{region}.api.cognitive.microsoft.com.
```

This is a sample request URL:

HTTP

```
https://westeurope.api.cognitive.microsoft.com/speechtotext/v3.1/transcriptions
```

ⓘ Note

See [this article](#) for Azure Government and Azure China endpoints.

After you turn on a custom domain for a Speech resource (which is necessary for private endpoints), that resource will use the following DNS name pattern for the basic REST API endpoint:

```
{your custom name}.cognitiveservices.azure.com
```

That means that in our example, the REST API endpoint name will be:

```
my-private-link-speech.cognitiveservices.azure.com
```

And the sample request URL needs to be converted to:

HTTP

```
https://my-private-link-speech.cognitiveservices.azure.com/speechtotext/v3.1/transcriptions
```

This URL should be reachable from the virtual network with the private endpoint attached (provided the [correct DNS resolution](#)).

After you turn on a custom domain name for a Speech resource, you typically replace the host name in all request URLs with the new custom domain host name. All other parts of the request (like the path `/speechtotext/v3.1/transcriptions` in the earlier example) remain the same.

💡 Tip

Some customers develop applications that use the region part of the regional endpoint's DNS name (for example, to send the request to the Speech resource deployed in the particular Azure region).

A custom domain for a Speech resource contains *no* information about the region where the resource is deployed. So the application logic described earlier will *not* work and needs to be altered.

Speech-to-text REST API for short audio and Text-to-speech REST API

The [Speech-to-text REST API for short audio](#) and the [Text-to-speech REST API](#) use two types of endpoints:

- [Cognitive Services regional endpoints](#) for communicating with the Cognitive Services REST API to obtain an authorization token
- Special endpoints for all other operations

⚠ Note

See [this article](#) for Azure Government and Azure China endpoints.

The detailed description of the special endpoints and how their URL should be transformed for a private-endpoint-enabled Speech resource is provided in [this subsection](#) about usage with the Speech SDK. The same principle described for the SDK applies for the Speech-to-text REST API for short audio and the Text-to-speech REST API.

Get familiar with the material in the subsection mentioned in the previous paragraph and see the following example. The example describes the Text-to-speech REST API. Usage of the Speech-to-text REST API for short audio is fully equivalent.

Note

When you're using the Speech-to-text REST API for short audio and Text-to-speech REST API in private endpoint scenarios, use a resource key passed through the `Ocp-Apim-Subscription-Key` header. (See details for [Speech-to-text REST API for short audio](#) and [Text-to-speech REST API](#))

Using an authorization token and passing it to the special endpoint via the `Authorization` header will work *only* if you've turned on the **All networks** access option in the **Networking** section of your Speech resource. In other cases you will get either `Forbidden` or `BadRequest` error when trying to obtain an authorization token.

Text-to-speech REST API usage example

We'll use West Europe as a sample Azure region and `my-private-link-speech.cognitiveservices.azure.com` as a sample Speech resource DNS name (custom domain). The custom domain name `my-private-link-speech.cognitiveservices.azure.com` in our example belongs to the Speech resource created in the West Europe region.

To get the list of the voices supported in the region, perform the following request:

HTTP

```
https://westeurope.tts.speech.microsoft.com/cognitiveservices/voices/list
```

See more details in the [Text-to-speech REST API documentation](#).

For the private-endpoint-enabled Speech resource, the endpoint URL for the same operation needs to be modified. The same request will look like this:

HTTP

```
https://my-private-link-speech.cognitiveservices.azure.com/tts/cognitiveservices/voices/list
```

See a detailed explanation in the [Construct endpoint URL](#) subsection for the Speech SDK.

Speech resource with a custom domain name and a private endpoint: Usage with the Speech SDK

Using the Speech SDK with a custom domain name and private-endpoint-enabled Speech resources requires you to review and likely change your application code.

We'll use `my-private-link-speech.cognitiveservices.azure.com` as a sample Speech resource DNS name (custom domain) for this section.

Construct endpoint URL

Usually in SDK scenarios (as well as in the Speech-to-text REST API for short audio and Text-to-speech REST API scenarios), Speech resources use the dedicated regional endpoints for different service offerings. The DNS name format for these endpoints is:

```
{region}.{speech service offering}.speech.microsoft.com
```

An example DNS name is:

```
westeurope.stt.speech.microsoft.com
```

All possible values for the region (first element of the DNS name) are listed in [Speech service supported regions](#). (See [this article](#) for Azure Government and Azure China endpoints.) The following table presents the possible values for the Speech service offering (second element of the DNS name):

DNS name value	Speech service offering
commands	Custom Commands
convai	Conversation Transcription
s2s	Speech Translation
stt	Speech-to-text
tts	Text-to-speech
voice	Custom Voice

So the earlier example (`westeurope.stt.speech.microsoft.com`) stands for a Speech-to-text endpoint in West Europe.

Private-endpoint-enabled endpoints communicate with Speech service via a special proxy. Because of that, *you must change the endpoint connection URLs*.

A "standard" endpoint URL looks like:

```
{region}.{speech service offering}.speech.microsoft.com/{URL path}
```

A private endpoint URL looks like:

```
{your custom name}.cognitiveservices.azure.com/{speech service offering}/{URL path}
```

Example 1. An application is communicating by using the following URL (speech recognition using the base model for US English in West Europe):

```
wss://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
```

To use it in the private-endpoint-enabled scenario when the custom domain name of the Speech resource is `my-private-link-speech.cognitiveservices.azure.com`, you must modify the URL like this:

```
wss://my-private-link-speech.cognitiveservices.azure.com/stt/speech/recognition/conversation/cognitiveservices/v1?language=en-US
```

Notice the details:

- The host name `westeurope.stt.speech.microsoft.com` is replaced by the custom domain host name `my-private-link-speech.cognitiveservices.azure.com`.
- The second element of the original DNS name (`stt`) becomes the first element of the URL path and precedes the original path. So the original URL `/speech/recognition/conversation/cognitiveservices/v1?language=en-US` becomes `/stt/speech/recognition/conversation/cognitiveservices/v1?language=en-US`.

Example 2. An application uses the following URL to synthesize speech in West Europe:

```
wss://westeurope.tts.speech.microsoft.com/cognitiveservices/websocket/v1
```

The following equivalent URL uses a private endpoint, where the custom domain name of the Speech resource is `my-private-link-speech.cognitiveservices.azure.com`:

```
wss://my-private-link-  
speech.cognitiveservices.azure.com/tts/cognitiveservices/websocket/v1
```

The same principle in Example 1 is applied, but the key element this time is `tts`.

Modifying applications

Follow these steps to modify your code:

1. Determine the application endpoint URL:

- [Turn on logging for your application](#) and run it to log activity.
- In the log file, search for `SPEECH-ConnectionString`. In matching lines, the `value` parameter contains the full URL that your application used to reach Speech Services.

Example:

```
(114917): 41ms SPX_DBG_TRACE_VERBOSE: property_bag_impl.cpp:138  
ISpxPropertyBagImpl::LogPropertyAndValue: this=0x0000028FE4809D78;  
name='SPEECH-ConnectionString';  
value='wss://westeurope.speech.microsoft.com/speech/recognition/con  
versation/cognitiveservices/v1?trafficType=spx&language=en-US'
```

So the URL that the application used in this example is:

```
wss://westeurope.speech.microsoft.com/speech/recognition/conversati  
on/cognitiveservices/v1?language=en-US
```

2. Create a `SpeechConfig` instance by using a full endpoint URL:

- a. Modify the endpoint that you just determined, as described in the earlier [Construct endpoint URL](#) section.
- b. Modify how you create the instance of `SpeechConfig`. Most likely, your application is using something like this:

```
C#
```

```
var config = SpeechConfig.FromSubscription(speechKey, azureRegion);
```

This won't work for a private-endpoint-enabled Speech resource because of the host name and URL changes that we described in the previous sections. If you try to run your existing application without any modifications by using the key of a private-endpoint-enabled resource, you'll get an authentication error (401).

To make it work, modify how you instantiate the `SpeechConfig` class and use "from endpoint"/"with endpoint" initialization. Suppose we have the following two variables defined:

- `speechKey` contains the key of the private-endpoint-enabled Speech resource.
- `endPoint` contains the full *modified* endpoint URL (using the type required by the corresponding programming language). In our example, this variable should contain:

```
wss://my-private-link-
speech.cognitiveservices.azure.com/stt/speech/recognition/conver-
sation/cognitiveservices/v1?language=en-US
```

Create a `SpeechConfig` instance:

C#

```
var config = SpeechConfig.FromEndpoint(endPoint, speechKey);
```

C++

```
auto config = SpeechConfig::FromEndpoint(endPoint, speechKey);
```

Java

```
SpeechConfig config = SpeechConfig.fromEndpoint(endPoint, speechKey);
```

Python

```
import azure.cognitiveservices.speech as speechsdk
speech_config = speechsdk.SpeechConfig(endpoint=endPoint,
subscription=speechKey)
```

objectivec

```
SPXSpeechConfiguration *speechConfig = [[SPXSpeechConfiguration alloc] initWithEndpoint:endPoint subscription:speechKey];
```

💡 Tip

The query parameters specified in the endpoint URI are not changed, even if they're set by other APIs. For example, if the recognition language is defined in the URI as query parameter `language=en-US`, and is also set to `ru-RU` via the corresponding property, the language setting in the URI is used. The effective language is then `en-US`.

Parameters set in the endpoint URI always take precedence. Other APIs can override only parameters that are not specified in the endpoint URI.

After this modification, your application should work with the private-endpoint-enabled Speech resources. We're working on more seamless support of private endpoint scenarios.

Use of Speech Studio

[Speech Studio](#) is a web portal with tools for building and integrating Azure Cognitive Services Speech service in your application. When you work in Speech Studio projects, network connections and API calls to the corresponding Speech resource are made on your behalf. Working with [private endpoints](#), [virtual network service endpoints](#), and other network security options can limit the availability of Speech Studio features. You normally use Speech Studio when working with features, like [Custom Speech](#), [Custom Neural Voice](#) and [Audio Content Creation](#).

Reaching Speech Studio web portal from a Virtual network

To use Speech Studio from a virtual machine within an Azure Virtual network, you must allow outgoing connections to the required set of [service tags](#) for this virtual network. See details [here](#).

Access to the Speech resource endpoint is *not* equal to access to Speech Studio web portal. Access to Speech Studio web portal via private or Virtual Network service endpoints is not supported.

Working with Speech Studio projects

This section describes working with the different kind of Speech Studio projects for the different network security options of the Speech resource. It's expected that the web browser connection to Speech Studio is established. Speech resource network security settings are set in Azure portal.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Speech resource.
3. In the **Resource Management** group in the left pane, select **Networking > Firewalls and virtual networks**.
4. Select one option from **All networks**, **Selected Networks** and **Private Endpoints**, or **Disabled**.

Custom Speech

The following table describes Custom Speech project accessibility per Speech resource **Networking > Firewalls and virtual networks** security setting.

ⓘ Note

If you allow only private endpoints via the **Networking > Private endpoint connections** tab, then you can't use Speech Studio with the Speech resource. You can still use the Speech resource outside of Speech Studio.

Speech resource network security setting	Speech Studio project accessibility
All networks	No restrictions
Selected Networks and Private Endpoints	Accessible from allowed public IP addresses
Disabled	Not accessible

If you select **Selected Networks and private endpoints**, then you will see a tab with **Virtual networks** and **Firewall** access configuration options. In the **Firewall** section, you must allow at least one public IP address and use this address for the browser connection with Speech Studio.

If you allow only access via **Virtual network**, then in effect you don't allow access to the Speech resource through Speech Studio. You can still use the Speech resource outside of Speech Studio.

To use custom speech without relaxing network access restrictions on your production Speech resource, consider one of these workarounds.

- Create another Speech resource for development that can be used on a public network. Prepare your custom model in Speech Studio on the development resource, and then copy the model to your production resource. See the [Models_CopyTo](#) REST request with [Speech-to-text REST API](#).
- You have the option to not use Speech Studio for custom speech. Use the [Speech-to-text REST API](#) for all custom speech operations.

Custom Voice and Audio Content Creation

You can use Custom Voice and Audio Content Creation Speech Studio projects only when the Speech resource network security setting is **All networks**.

Adjust an application to use a Speech resource without private endpoints

In this article, we've pointed out several times that enabling a custom domain for a Speech resource is *irreversible*. Such a resource will use a different way of communicating with Speech service, compared to the ones that are using [regional endpoint names](#).

This section explains how to use a Speech resource with a custom domain name but *without* any private endpoints with the Speech Services REST APIs and [Speech SDK](#). This might be a resource that was once used in a private endpoint scenario, but then had its private endpoints deleted.

DNS configuration

Remember how a custom domain DNS name of the private-endpoint-enabled Speech resource is [resolved from public networks](#). In this case, the IP address resolved points to a proxy endpoint for a virtual network. That endpoint is used for dispatching the network traffic to the private-endpoint-enabled Cognitive Services resource.

However, when *all* resource private endpoints are removed (or right after the enabling of the custom domain name), the CNAME record of the Speech resource is reprovisioned. It now points to the IP address of the corresponding [Cognitive Services regional endpoint](#).

So the output of the `nslookup` command will look like this:

dos

```
C:\>nslookup my-private-link-speech.cognitiveservices.azure.com
Server: UnKnown
Address: fe80::1

Non-authoritative answer:
Name: apimgmthskquihpkz6d90kmhvnbrx3ms3pdubscpdfk1tsx3a.cloudapp.net
Address: 13.93.122.1
Aliases: my-private-link-speech.cognitiveservices.azure.com
westeurope.api.cognitive.microsoft.com
cognitiveweprod.trafficmanager.net
cognitiveweprod.azure-api.net

apimgmttmdjylckcx6clmh2isu2wr38uqzm63s8n4ub2y3e6xs.trafficmanager.net
cognitiveweprod-westeurope-01.region.azure-api.net
```

Compare it with the output from [this section](#).

Speech resource with a custom domain name and without private endpoints: Usage with the REST APIs

Speech-to-text REST API

Speech-to-text REST API usage is fully equivalent to the case of [private-endpoint-enabled Speech resources](#).

Speech-to-text REST API for short audio and Text-to-speech REST API

In this case, usage of the Speech-to-text REST API for short audio and usage of the Text-to-speech REST API have no differences from the general case, with one exception. (See the following note.) You should use both APIs as described in the [speech-to-text REST API for short audio](#) and [Text-to-speech REST API](#) documentation.

ⓘ Note

When you're using the Speech-to-text REST API for short audio and Text-to-speech REST API in custom domain scenarios, use a Speech resource key passed through the `Ocp-Apim-Subscription-Key` header. (See details for [Speech-to-text REST API for short audio](#) and [Text-to-speech REST API](#))

Using an authorization token and passing it to the special endpoint via the `Authorization` header will work *only* if you've turned on the **All networks** access option in the **Networking** section of your Speech resource. In other cases you will get either `Forbidden` or `BadRequest` error when trying to obtain an authorization token.

Speech resource with a custom domain name and without private endpoints: Usage with the Speech SDK

Using the Speech SDK with custom-domain-enabled Speech resources *without* private endpoints is equivalent to the general case as described in the [Speech SDK documentation](#).

In case you have modified your code for using with a [private-endpoint-enabled Speech resource](#), consider the following.

In the section on [private-endpoint-enabled Speech resources](#), we explained how to determine the endpoint URL, modify it, and make it work through "from endpoint"/"with endpoint" initialization of the `SpeechConfig` class instance.

However, if you try to run the same application after having all private endpoints removed (allowing some time for the corresponding DNS record reprovisioning), you'll get an internal service error (404). The reason is that the `DNS record` now points to the regional Cognitive Services endpoint instead of the virtual network proxy, and the URL paths like `/stt/speech/recognition/conversation/cognitiveservices/v1?language=en-US` won't be found there.

You need to roll back your application to the standard instantiation of `SpeechConfig` in the style of the following code:

C#

```
var config = SpeechConfig.FromSubscription(speechKey, azureRegion);
```

Simultaneous use of private endpoints and Virtual Network service endpoints

You can use [private endpoints](#) and [Virtual Network service endpoints](#) to access to the same Speech resource simultaneously. To enable this simultaneous use, you need to use

the **Selected Networks and Private Endpoints** option in the networking settings of the Speech resource in the Azure portal. Other options aren't supported for this scenario.

Pricing

For pricing details, see [Azure Private Link pricing](#).

Learn more

- Use Speech service through a Virtual Network service endpoint
- Azure Private Link
- Azure VNet service endpoint
- Speech SDK
- Speech-to-text REST API
- Text-to-speech REST API

Use Speech service through a Virtual Network service endpoint

Article • 02/23/2022 • 9 minutes to read

Azure Virtual Network service endpoints help to provide secure and direct connectivity to Azure services over an optimized route on the Azure backbone network. Endpoints help you secure your critical Azure service resources to only your virtual networks. Service endpoints enable private IP addresses in the virtual network to reach the endpoint of an Azure service without needing a public IP address on the virtual network.

This article explains how to set up and use Virtual Network service endpoints with Speech service in Azure Cognitive Services.

ⓘ Note

Before you start, review [how to use virtual networks with Cognitive Services](#).

This article also describes [how to remove Virtual Network service endpoints later but still use the Speech resource](#).

To set up a Speech resource for Virtual Network service endpoint scenarios, you need to:

1. [Create a custom domain name for the Speech resource](#).
2. [Configure virtual networks and networking settings for the Speech resource](#).
3. [Adjust existing applications and solutions](#).

ⓘ Note

Setting up and using Virtual Network service endpoints for Speech service is similar to setting up and using private endpoints. In this article, we refer to the corresponding sections of the [article on using private endpoints](#) when the procedures are the same.

Private endpoints and Virtual Network service endpoints

Azure provides private endpoints and Virtual Network service endpoints for traffic that tunnels via the [private Azure backbone network](#). The purpose and underlying

technologies of these endpoint types are similar. But there are differences between the two technologies. We recommend that you learn about the pros and cons of both before you design your network.

There are a few things to consider when you decide which technology to use:

- Both technologies ensure that traffic between the virtual network and the Speech resource doesn't travel over the public internet.
- A private endpoint provides a dedicated private IP address for your Speech resource. This IP address is accessible only within a specific virtual network and subnet. You have full control of the access to this IP address within your network infrastructure.
- Virtual Network service endpoints don't provide a dedicated private IP address for the Speech resource. Instead, they encapsulate all packets sent to the Speech resource and deliver them directly over the Azure backbone network.
- Both technologies support on-premises scenarios. By default, when they use Virtual Network service endpoints, Azure service resources secured to virtual networks can't be reached from on-premises networks. But you can [change that behavior](#).
- Virtual Network service endpoints are often used to restrict the access for a Speech resource based on the virtual networks from which the traffic originates.
- For Cognitive Services, enabling the Virtual Network service endpoint forces the traffic for all Cognitive Services resources to go through the private backbone network. That requires explicit network access configuration. (For more information, see [Configure virtual networks and the Speech resource networking settings](#).) Private endpoints don't have this limitation and provide more flexibility for your network configuration. You can access one resource through the private backbone and another through the public internet by using the same subnet of the same virtual network.
- Private endpoints incur [extra costs](#). Virtual Network service endpoints are free.
- Private endpoints require [extra DNS configuration](#).
- One Speech resource can work simultaneously with both private endpoints and Virtual Network service endpoints.

We recommend that you try both endpoint types before you make a decision about your production design.

For more information, see these resources:

- [Azure Private Link and private endpoint documentation](#)
- [Virtual Network service endpoints documentation](#)

This article describes how to use Virtual Network service endpoints with Speech service. For information about private endpoints, see [Use Speech service through a private endpoint](#).

Create a custom domain name

Virtual Network service endpoints require a [custom subdomain name for Cognitive Services](#). Create a custom domain by following the [guidance](#) in the private endpoint article. All warnings in the section also apply to Virtual Network service endpoints.

Configure virtual networks and the Speech resource networking settings

You need to add all virtual networks that are allowed access via the service endpoint to the Speech resource networking properties.

Note

To access a Speech resource via the Virtual Network service endpoint, you need to enable the `Microsoft.CognitiveServices` service endpoint type for the required subnets of your virtual network. Doing so will route all subnet traffic related to Cognitive Services through the private backbone network. If you intend to access any other Cognitive Services resources from the same subnet, make sure these resources are configured to allow your virtual network.

If a virtual network isn't added as *allowed* in the Speech resource networking properties, it won't have access to the Speech resource via the service endpoint, even if the `Microsoft.CognitiveServices` service endpoint is enabled for the virtual network. And if the service endpoint is enabled but the virtual network isn't allowed, the Speech resource won't be accessible for the virtual network through a public IP address, no matter what the Speech resource's other network security settings are. That's because enabling the `Microsoft.CognitiveServices` endpoint routes all traffic related to Cognitive Services through the private backbone network, and in this case the virtual network should be explicitly allowed to access the resource. This guidance applies for all Cognitive Services resources, not just for Speech resources.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Speech resource.

3. In the Resource Management group in the left pane, select **Networking**.
4. On the **Firewalls and virtual networks** tab, select **Selected Networks and Private Endpoints**.

 **Note**

To use Virtual Network service endpoints, you need to select the **Selected Networks and Private Endpoints** network security option. No other options are supported. If your scenario requires the **All networks** option, consider using **private endpoints**, which support all three network security options.

5. Select **Add existing virtual network** or **Add new virtual network** and provide the required parameters. Select **Add** for an existing virtual network or **Create** for a new one. If you add an existing virtual network, the `Microsoft.CognitiveServices` service endpoint will automatically be enabled for the selected subnets. This operation can take up to 15 minutes. Also, see the note at the beginning of this section.

Enabling service endpoint for an existing virtual network

As described in the previous section, when you configure a virtual network as *allowed* for the Speech resource, the `Microsoft.CognitiveServices` service endpoint is automatically enabled. If you later disable it, you need to re-enable it manually to restore the service endpoint access to the Speech resource (and to other Cognitive Services resources):

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the virtual network.
3. In the **Settings** group in the left pane, select **Subnets**.
4. Select the required subnet.
5. A new panel appears on the right side of the window. In this panel, in the **Service Endpoints** section, select `Microsoft.CognitiveServices` in the **Services** list.
6. Select **Save**.

Adjust existing applications and solutions

A Speech resource that has a custom domain enabled interacts with the Speech service in a different way. This is true for a custom-domain-enabled Speech resource regardless

of whether service endpoints are configured. Information in this section applies to both scenarios.

Use a Speech resource that has a custom domain name and allowed virtual networks

In this scenario, the **Selected Networks and Private Endpoints** option is selected in the networking settings of the Speech resource and at least one virtual network is allowed. This scenario is equivalent to [using a Speech resource that has a custom domain name and a private endpoint enabled](#).

Use a Speech resource that has a custom domain name but that doesn't have allowed virtual networks

In this scenario, private endpoints aren't enabled and one of these statements is true:

- The **Selected Networks and Private Endpoints** option is selected in the networking settings of the Speech resource, but no allowed virtual networks are configured.
- The **All networks** option is selected in the networking settings of the Speech resource.

This scenario is equivalent to [using a Speech resource that has a custom domain name and that doesn't have private endpoints](#).

Use of Speech Studio

[Speech Studio](#) is a web portal with tools for building and integrating Azure Cognitive Services Speech service in your application. When you work in Speech Studio projects, network connections and API calls to the corresponding Speech resource are made on your behalf. Working with [private endpoints](#), [virtual network service endpoints](#), and other network security options can limit the availability of Speech Studio features. You normally use Speech Studio when working with features, like [Custom Speech](#), [Custom Neural Voice](#) and [Audio Content Creation](#).

Reaching Speech Studio web portal from a Virtual network

To use Speech Studio from a virtual machine within an Azure Virtual network, you must allow outgoing connections to the required set of [service tags](#) for this virtual network. See details [here](#).

Access to the Speech resource endpoint is *not* equal to access to Speech Studio web portal. Access to Speech Studio web portal via private or Virtual Network service endpoints is not supported.

Working with Speech Studio projects

This section describes working with the different kind of Speech Studio projects for the different network security options of the Speech resource. It's expected that the web browser connection to Speech Studio is established. Speech resource network security settings are set in Azure portal.

1. Go to the [Azure portal](#) and sign in to your Azure account.
2. Select the Speech resource.
3. In the **Resource Management** group in the left pane, select **Networking > Firewalls and virtual networks**.
4. Select one option from **All networks**, **Selected Networks and Private Endpoints**, or **Disabled**.

Custom Speech

The following table describes Custom Speech project accessibility per Speech resource **Networking > Firewalls and virtual networks** security setting.

ⓘ Note

If you allow only private endpoints via the **Networking > Private endpoint connections** tab, then you can't use Speech Studio with the Speech resource. You can still use the Speech resource outside of Speech Studio.

Speech resource network security setting	Speech Studio project accessibility
All networks	No restrictions
Selected Networks and Private Endpoints	Accessible from allowed public IP addresses
Disabled	Not accessible

If you select **Selected Networks and private endpoints**, then you will see a tab with **Virtual networks** and **Firewall** access configuration options. In the **Firewall** section, you must allow at least one public IP address and use this address for the browser connection with Speech Studio.

If you allow only access via **Virtual network**, then in effect you don't allow access to the Speech resource through Speech Studio. You can still use the Speech resource outside of Speech Studio.

To use custom speech without relaxing network access restrictions on your production Speech resource, consider one of these workarounds.

- Create another Speech resource for development that can be used on a public network. Prepare your custom model in Speech Studio on the development resource, and then copy the model to your production resource. See the [CopyModelToSubscriptionToSubscription](#) REST request with [Speech-to-text REST API](#).
- You have the option to not use Speech Studio for custom speech. Use the [Speech-to-text REST API](#) for all custom speech operations.

Custom Voice and Audio Content Creation

You can use Custom Voice and Audio Content Creation Speech Studio projects only when the Speech resource network security setting is **All networks**.

Simultaneous use of private endpoints and Virtual Network service endpoints

You can use [private endpoints](#) and [Virtual Network service endpoints](#) to access to the same Speech resource simultaneously. To enable this simultaneous use, you need to use the **Selected Networks and Private Endpoints** option in the networking settings of the Speech resource in the Azure portal. Other options aren't supported for this scenario.

Learn more

- [Use Speech service through a private endpoint](#)
- [Azure Virtual Network service endpoints](#)
- [Azure Private Link](#)
- [Speech SDK](#)
- [Speech-to-text REST API](#)
- [Text-to-speech REST API](#)

How to monitor and control service connections with the Speech SDK

Article • 05/12/2022 • 2 minutes to read

`SpeechRecognizer` and other objects in the Speech SDK automatically connect to the Speech Service when it's appropriate. Sometimes, you may either want additional control over when connections begin and end or want more information about when the Speech SDK establishes or loses its connection. The supporting `Connection` class provides this capability.

Retrieve a Connection object

A `Connection` can be obtained from most top-level Speech SDK objects via a static `From...` factory method, e.g. `Connection::FromRecognizer(recognizer)` for `SpeechRecognizer`.

C#

```
var connection = Connection.FromRecognizer(recognizer);
```

Monitor for connections and disconnections

A `Connection` raises `Connected` and `Disconnected` events when the corresponding status change happens in the Speech SDK's connection to the Speech Service. You can listen to these events to know the latest connection state.

C#

```
connection.Connected += (sender, connectedEventArgs) =>
{
    Console.WriteLine($"Successfully connected, sessionId:
{connectedEventArgs.SessionId}");
};

connection.Disconnected += (sender, disconnectedEventArgs) =>
{
    Console.WriteLine($"Disconnected, sessionId:
{disconnectedEventArgs.SessionId}");
};
```

Connect and disconnect

`Connection` has explicit methods to start or end a connection to the Speech Service.

Reasons you may want to use these include:

- "Pre-connecting" to the Speech Service to allow the first interaction to start as quickly as possible
- Establishing connection at a specific time in your application's logic to gracefully and predictably handle initial connection failures
- Disconnecting to clear an idle connection when you don't expect immediate reconnection but also don't want to destroy the object

Some important notes on the behavior when manually modifying connection state:

- Trying to connect when already connected will do nothing. It will not generate an error. Monitor the `Connected` and `Disconnected` events if you want to know the current state of the connection.
- A failure to connect that originates from a problem that has no involvement with the Speech Service -- such as attempting to do so from an invalid state -- will throw or return an error as appropriate to the programming language. Failures that require network resolution -- such as authentication failures -- will not throw or return an error but instead generate a `Canceled` event on the top-level object the `Connection` was created from.
- Manually disconnecting from the Speech Service during an ongoing interaction will result in a connection error and loss of data for that interaction. Connection errors are surfaced on the appropriate top-level object's `Canceled` event.

C#

```
try
{
    connection.Open(forContinuousRecognition: false);
}
catch (ApplicationException ex)
{
    Console.WriteLine($"Couldn't pre-connect. Details: {ex.Message}");
}
// ... Use the SpeechRecognizer
connection.Close();
```

Next steps

[Explore our samples on GitHub](#)

Configure OpenSSL for Linux

Article • 06/22/2022 • 4 minutes to read

With the Speech SDK, [OpenSSL](#) is dynamically configured to the host-system version.

ⓘ Note

This article is only applicable where the Speech SDK is **supported on Linux**.

To ensure connectivity, verify that OpenSSL certificates have been installed in your system. Run a command:

Bash

```
openssl version -d
```

The output on Ubuntu/Debian based systems should be:

```
OPENSSLDIR: "/usr/lib/ssl"
```

Check whether there's a `certs` subdirectory under OPENSSLDIR. In the example above, it would be `/usr/lib/ssl/certs`.

- If the `/usr/lib/ssl/certs` exists, and if it contains many individual certificate files (with `.crt` or `.pem` extension), there's no need for further actions.
- If OPENSSLDIR is something other than `/usr/lib/ssl` or there's a single certificate bundle file instead of multiple individual files, you need to set an appropriate SSL environment variable to indicate where the certificates can be found.

Examples

Here are some example environment variables to configure per OpenSSL directory.

- OPENSSLDIR is `/opt/ssl`. There's a `certs` subdirectory with many `.crt` or `.pem` files. Set the environment variable `SSL_CERT_DIR` to point at `/opt/ssl/certs` before using the Speech SDK. For example:

Bash

```
export SSL_CERT_DIR=/opt/ssl/certs
```

- OPENSSLDIR is `/etc/pki/tls` (like on RHEL/CentOS based systems). There's a `certs` subdirectory with a certificate bundle file, for example `ca-bundle.crt`. Set the environment variable `SSL_CERT_FILE` to point at that file before using the Speech SDK. For example:

Bash

```
export SSL_CERT_FILE=/etc/pki/tls/certs/ca-bundle.crt
```

Certificate revocation checks

When the Speech SDK connects to the Speech Service, it checks the Transport Layer Security (TLS/SSL) certificate. The Speech SDK verifies that the certificate reported by the remote endpoint is trusted and hasn't been revoked. This verification provides a layer of protection against attacks involving spoofing and other related vectors. The check is accomplished by retrieving a certificate revocation list (CRL) from a certificate authority (CA) used by Azure. A list of Azure CA download locations for updated TLS CRLs can be found in [this document](#).

If a destination posing as the Speech Service reports a certificate that's been revoked in a retrieved CRL, the SDK will terminate the connection and report an error via a `Canceled` event. The authenticity of a reported certificate can't be checked without an updated CRL. Therefore, the Speech SDK will also treat a failure to download a CRL from an Azure CA location as an error.

Large CRL files (>10 MB)

One cause of CRL-related failures is the use of large CRL files. This class of error is typically only applicable to special environments with extended CA chains. Standard public endpoints shouldn't encounter this class of issue.

The default maximum CRL size used by the Speech SDK (10 MB) can be adjusted per config object. The property key for this adjustment is `CONFIG_MAX_CRL_SIZE_KB` and the value, specified as a string, is by default "10000" (10 MB). For example, when creating a `SpeechRecognizer` object (that manages a connection to the Speech Service), you can set this property in its `SpeechConfig`. In the snippet below, the configuration is adjusted to permit a CRL file size up to 15 MB.

```
C#
```

```
config SetProperty("CONFIG_MAX_CRL_SIZE_KB", "15000");
```

Bypassing or ignoring CRL failures

If an environment can't be configured to access an Azure CA location, the Speech SDK will never be able to retrieve an updated CRL. You can configure the SDK either to continue and log download failures or to bypass all CRL checks.

⚠ Warning

CRL checks are a security measure and bypassing them increases susceptibility to attacks. They should not be bypassed without thorough consideration of the security implications and alternative mechanisms for protecting against the attack vectors that CRL checks mitigate.

To continue with the connection when a CRL can't be retrieved, set the property `"OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE"` to `"true"`. An attempt will still be made to retrieve a CRL and failures will still be emitted in logs, but connection attempts will be allowed to continue.

```
C#
```

```
config SetProperty("OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE", "true");
```

To turn off certificate revocation checks, set the property `"OPENSSL_DISABLE_CRL_CHECK"` to `"true"`. Then, while connecting to the Speech Service, there will be no attempt to check or download a CRL and no automatic verification of a reported TLS/SSL certificate.

```
C#
```

```
config SetProperty("OPENSSL_DISABLE_CRL_CHECK", "true");
```

CRL caching and performance

By default, the Speech SDK will cache a successfully downloaded CRL on disk to improve the initial latency of future connections. When no cached CRL is present or when the cached CRL is expired, a new list will be downloaded.

Some Linux distributions don't have a `TMP` or `TMPDIR` environment variable defined, so the Speech SDK won't cache downloaded CRLs. Without `TMP` or `TMPDIR` environment variable defined, the Speech SDK will download a new CRL for each connection. To improve initial connection performance in this situation, you can [create a `TMPDIR` environment variable and set it to the accessible path of a temporary directory](#). ↗.

Next steps

- [Speech SDK overview](#)
- [Install the Speech SDK](#)

Configure Azure Cognitive Services virtual networks

Article • 08/03/2022 • 16 minutes to read

Azure Cognitive Services provides a layered security model. This model enables you to secure your Cognitive Services accounts to a specific subset of networks. When network rules are configured, only applications requesting data over the specified set of networks can access the account. You can limit access to your resources with request filtering. Allowing only requests originating from specified IP addresses, IP ranges or from a list of subnets in [Azure Virtual Networks](#).

An application that accesses a Cognitive Services resource when network rules are in effect requires authorization. Authorization is supported with [Azure Active Directory](#) (Azure AD) credentials or with a valid API key.

Important

Turning on firewall rules for your Cognitive Services account blocks incoming requests for data by default. In order to allow requests through, one of the following conditions needs to be met:

- The request should originate from a service operating within an Azure Virtual Network (VNet) on the allowed subnet list of the target Cognitive Services account. The endpoint in requests originated from VNet needs to be set as the [custom subdomain](#) of your Cognitive Services account.
- Or the request should originate from an allowed list of IP addresses.

Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on.

Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See [Install Azure PowerShell](#) to get started. To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Scenarios

To secure your Cognitive Services resource, you should first configure a rule to deny access to traffic from all networks (including internet traffic) by default. Then, you should configure rules that grant access to traffic from specific VNets. This configuration enables you to build a secure network boundary for your applications. You can also configure rules to grant access to traffic from select public internet IP address ranges, enabling connections from specific internet or on-premises clients.

Network rules are enforced on all network protocols to Azure Cognitive Services, including REST and WebSocket. To access data using tools such as the Azure test consoles, explicit network rules must be configured. You can apply network rules to existing Cognitive Services resources, or when you create new Cognitive Services resources. Once network rules are applied, they're enforced for all requests.

Supported regions and service offerings

Virtual networks (VNets) are supported in [regions where Cognitive Services are available](#). Cognitive Services supports service tags for network rules configuration. The services listed below are included in the **CognitiveServicesManagement** service tag.

- ✓ Anomaly Detector
- ✓ Computer Vision
- ✓ Content Moderator
- ✓ Custom Vision
- ✓ Face
- ✓ Language Understanding (LUIS)
- ✓ Personalizer
- ✓ Speech service
- ✓ Language service
- ✓ QnA Maker
- ✓ Translator Text

Note

If you're using LUIS, Speech Services, or Language services, the **CognitiveServicesManagement** tag only enables you use the service using the SDK or REST API. To access and use LUIS portal , Speech Studio or Language Studio from a virtual network, you will need to use the following tags:

- **AzureActiveDirectory**
- **AzureFrontDoor.Frontend**
- **AzureResourceManager**

- CognitiveServicesManagement

Change the default network access rule

By default, Cognitive Services resources accept connections from clients on any network. To limit access to selected networks, you must first change the default action.

⚠ Warning

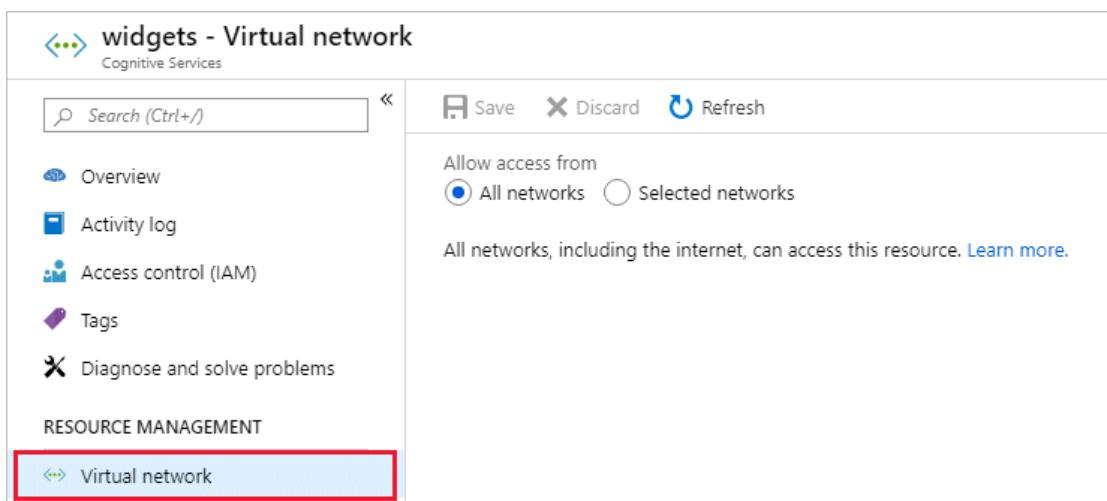
Making changes to network rules can impact your applications' ability to connect to Azure Cognitive Services. Setting the default network rule to **deny** blocks all access to the data unless specific network rules that **grant** access are also applied. Be sure to grant access to any allowed networks using network rules before you change the default rule to deny access. If you are allowing listing IP addresses for your on-premises network, be sure to add all possible outgoing public IP addresses from your on-premises network.

Managing default network access rules

You can manage default network access rules for Cognitive Services resources through the Azure portal, PowerShell, or the Azure CLI.

Azure portal

1. Go to the Cognitive Services resource you want to secure.
2. Select the **RESOURCE MANAGEMENT** menu called **Virtual network**.



3. To deny access by default, choose to allow access from **Selected networks**.

With the **Selected networks** setting alone, unaccompanied by configured **Virtual networks** or **Address ranges** - all access is effectively denied. When all access is denied, requests attempting to consume the Cognitive Services resource aren't permitted. The Azure portal, Azure PowerShell or, Azure CLI can still be used to configure the Cognitive Services resource.

4. To allow traffic from all networks, choose to allow access from **All networks**.

The screenshot shows the Azure portal interface for managing a Cognitive Service. The left sidebar has a 'Virtual network' section selected. The main content area shows a note about firewall settings. Below it, under 'Allow access from', there are two options: 'All networks' and 'Selected networks'. The 'Selected networks' option is selected and highlighted with a red box. In the 'Firewall' section, there's a note to add IP ranges and an input field for 'IP address or CIDR'.

5. Select **Save** to apply your changes.

Grant access from a virtual network

You can configure Cognitive Services resources to allow access only from specific subnets. The allowed subnets may belong to a VNet in the same subscription, or in a different subscription, including subscriptions belonging to a different Azure Active Directory tenant.

Enable a [service endpoint](#) for Azure Cognitive Services within the VNet. The service endpoint routes traffic from the VNet through an optimal path to the Azure Cognitive Services service. The identities of the subnet and the virtual network are also transmitted with each request. Administrators can then configure network rules for the Cognitive Services resource that allow requests to be received from specific subnets in a VNet. Clients granted access via these network rules must continue to meet the authorization requirements of the Cognitive Services resource to access the data.

Each Cognitive Services resource supports up to 100 virtual network rules, which may be combined with [IP network rules](#).

Required permissions

To apply a virtual network rule to a Cognitive Services resource, the user must have the appropriate permissions for the subnets being added. The required permission is the default *Contributor* role, or the *Cognitive Services Contributor* role. Required permissions can also be added to custom role definitions.

Cognitive Services resource and the virtual networks granted access may be in different subscriptions, including subscriptions that are a part of a different Azure AD tenant.

Note

Configuration of rules that grant access to subnets in virtual networks that are a part of a different Azure Active Directory tenant are currently only supported through PowerShell, CLI and REST APIs. Such rules cannot be configured through the Azure portal, though they may be viewed in the portal.

Managing virtual network rules

You can manage virtual network rules for Cognitive Services resources through the Azure portal, PowerShell, or the Azure CLI.

Azure portal

1. Go to the Cognitive Services resource you want to secure.
2. Select the **RESOURCE MANAGEMENT** menu called **Virtual network**.
3. Check that you've selected to allow access from **Selected networks**.
4. To grant access to a virtual network with an existing network rule, under **Virtual networks**, select **Add existing virtual network**.

widgets - Virtual network
Cognitive Services

Save Discard Refresh

Firewall settings allowing access to cognitive service will remain in effect for up to three minutes after saving updated settings restricting access.

Allow access from
 All networks Selected networks

Configure network security for your cognitive service. [Learn more.](#)

Virtual networks
Secure your cognitive service with virtual networks. [+ Add existing virtual network](#) [+ Add new virtual network](#)

VIRTUAL NETWORK	SUBNET	ADDRESS RANGE
No network selected.		

Firewall
Add IP ranges to allow access from the internet or your on-premises networks. [Learn more.](#)

Add your client IP address? [?](#)

ADDRESS RANGE

5. Select the **Virtual networks** and **Subnets** options, and then select **Enable**.

Add networks X

* Subscription

* Virtual networks

* Subnets

Information
The following networks don't have service endpoints enabled for 'Microsoft.CognitiveServices'. Enabling access will take up to 15 minutes to complete. After starting this operation, it is safe to leave and return later if you do not wish to wait.

VIRTUAL NETWORK	SERVICE ENDPOINT STATUS
▼ widgets-vnet...	...
default	Not enabled

Enable

6. To create a new virtual network and grant it access, select **Add new virtual network**.

The screenshot shows a Microsoft Azure Cognitive Services interface for managing virtual networks. At the top, there's a header with the title 'widgets - Virtual network' and a 'Cognitive Services' link. Below the header are standard navigation buttons: 'Save' (blue), 'Discard' (red), and 'Refresh' (green). A status message at the top states: 'Firewall settings allowing access to cognitive service will remain in effect for up to three minutes after saving updated settings restricting access.' Under the 'Allow access from' section, the 'Selected networks' radio button is selected. A note below says, 'Configure network security for your cognitive service. [Learn more.](#)' In the 'Virtual networks' section, there's a table with columns 'VIRTUAL NETWORK', 'SUBNET', and 'ADDRESS RANGE'. A sub-note says, 'Secure your cognitive service with virtual networks.' Below the table, it says 'No network selected.' In the 'Firewall' section, there's a note about adding IP ranges and a checkbox for 'Add your client IP address?'. At the bottom, there's a 'ADDRESS RANGE' input field with a placeholder 'IP address or CIDR'.

7. Provide the information necessary to create the new virtual network, and then select **Create**.

Create virtual network

★ Name
widgets-vnet ✓

★ Address space ⓘ
10.1.0.0/16
10.1.0.0 - 10.1.255.255 (65536 addresses)

★ Subscription
widgets-subscription ▾

★ Resource group
widgets-resource-group ▾
[Create new](#)

★ Location
(US) West US 2 ▾

Subnet

★ Name
default

★ Address range ⓘ
10.1.0.0/24 ✓
10.1.0.0 - 10.1.0.255 (256 addresses)

DDoS protection ⓘ
 Basic Standard

Service endpoint ⓘ
Microsoft.CognitiveServices

Firewall ⓘ

(!) Note

If a service endpoint for Azure Cognitive Services wasn't previously configured for the selected virtual network and subnets, you can configure it as part of this operation.

Presently, only virtual networks belonging to the same Azure Active Directory tenant are shown for selection during rule creation. To grant access to a subnet in a virtual network belonging to another tenant, please use PowerShell, CLI or REST APIs.

8. To remove a virtual network or subnet rule, select ... to open the context menu for the virtual network or subnet, and select Remove.

The screenshot shows the Azure portal interface for managing a virtual network. At the top, it says "widgets - Virtual network" under "Cognitive Services". Below that are "Save", "Discard", and "Refresh" buttons. A message box states: "Firewall settings allowing access to cognitive service will remain in effect for up to three minutes after saving updated settings restricting access." Under "Allow access from", the "Selected networks" radio button is selected. A note says: "Configure network security for your cognitive service. [Learn more](#)." Below this is a section for "Virtual networks" with "Secure your cognitive service with virtual networks." buttons for "Add existing virtual network" and "Add new virtual network". A table lists subnets:

VIRTUAL NETWORK	SUBNET	ADDRESS RANGE	ENDPOINT STATUS	RESOURCE GROUP	SUBSCRIPTION
widgets-vnet	1	default 10.1.0.0/24	✓ Enabled	widgets-	widgets-resource-gr... widgets-subscription ...

A context menu is open over the "Remove" button for the first subnet rule. The menu items are "Remove" (highlighted with a red box) and "...". Below the table is a "Firewall" section with a note to "Add IP ranges to allow access from the internet or your on-premises networks." and a checkbox for "Add your client IP address". There is also an "ADDRESS RANGE" input field with "IP address or CIDR" placeholder text.

9. Select Save to apply your changes.

ⓘ Important

Be sure to **set the default rule to deny**, or network rules have no effect.

Grant access from an internet IP range

You can configure Cognitive Services resources to allow access from specific public internet IP address ranges. This configuration grants access to specific services and on-premises networks, effectively blocking general internet traffic.

Provide allowed internet address ranges using [CIDR notation](#) in the form `16.17.18.0/24` or as individual IP addresses like `16.17.18.19`.

💡 Tip

Small address ranges using "/31" or "/32" prefix sizes are not supported. These ranges should be configured using individual IP address rules.

IP network rules are only allowed for **public internet** IP addresses. IP address ranges reserved for private networks (as defined in [RFC 1918](#)) aren't allowed in IP rules. Private networks include addresses that start with `10.*`, `172.16.* - 172.31.*`, and `192.168.*`.

Only IPV4 addresses are supported at this time. Each Cognitive Services resource supports up to 100 IP network rules, which may be combined with [Virtual network rules](#).

Configuring access from on-premises networks

To grant access from your on-premises networks to your Cognitive Services resource with an IP network rule, you must identify the internet facing IP addresses used by your network. Contact your network administrator for help.

If you're using [ExpressRoute](#) on-premises for public peering or Microsoft peering, you'll need to identify the NAT IP addresses. For public peering, each ExpressRoute circuit by default uses two NAT IP addresses. Each is applied to Azure service traffic when the traffic enters the Microsoft Azure network backbone. For Microsoft peering, the NAT IP addresses that are used are either customer provided or are provided by the service provider. To allow access to your service resources, you must allow these public IP addresses in the resource IP firewall setting. To find your public peering ExpressRoute circuit IP addresses, open a support ticket with [ExpressRoute](#) via the Azure portal. Learn more about [NAT for ExpressRoute public and Microsoft peering](#).

Managing IP network rules

You can manage IP network rules for Cognitive Services resources through the Azure portal, PowerShell, or the Azure CLI.

Azure portal

1. Go to the Cognitive Services resource you want to secure.
2. Select the **RESOURCE MANAGEMENT** menu called **Virtual network**.
3. Check that you've selected to allow access from **Selected networks**.
4. To grant access to an internet IP range, enter the IP address or address range (in [CIDR format](#)) under **Firewall > Address Range**. Only valid public IP (non-reserved) addresses are accepted.

widgets - Virtual network
Cognitive Services

» Save Discard Refresh

Firewall settings allowing access to cognitive service will remain in effect for up to three minutes after saving updated settings restricting access.

Allow access from
 All networks Selected networks

Configure network security for your cognitive service. [Learn more](#).

Virtual networks
Secure your cognitive service with virtual networks. [+ Add existing virtual network](#) [+ Add new virtual network](#)

VIRTUAL NETWORK	SUBNET	ADDRESS RANGE
No network selected.		

Firewall
Add IP ranges to allow access from the internet or your on-premises networks. [Learn more](#).

Add your client IP address?

ADDRESS RANGE

IP address or CIDR
173.0.0.0/16

5. To remove an IP network rule, select the trash can icon next to the address range.

widgets - Virtual network
Cognitive Services

» Save Discard Refresh

Firewall settings allowing access to cognitive service will remain in effect for up to three minutes after saving updated settings restricting access.

Allow access from
 All networks Selected networks

Configure network security for your cognitive service. [Learn more](#).

Virtual networks
Secure your cognitive service with virtual networks. [+ Add existing virtual network](#) [+ Add new virtual network](#)

VIRTUAL NETWORK	SUBNET	ADDRESS RANGE
No network selected.		

Firewall
Add IP ranges to allow access from the internet or your on-premises networks. [Learn more](#).

Add your client IP address?

ADDRESS RANGE

IP address or CIDR
173.0.0.0/16

6. Select **Save** to apply your changes.

Important

Be sure to **set the default rule to deny**, or network rules have no effect.

Use private endpoints

You can use [private endpoints](#) for your Cognitive Services resources to allow clients on a virtual network (VNet) to securely access data over a [Private Link](#). The private endpoint uses an IP address from the VNet address space for your Cognitive Services resource. Network traffic between the clients on the VNet and the resource traverses the VNet and a private link on the Microsoft backbone network, eliminating exposure from the public internet.

Private endpoints for Cognitive Services resources let you:

- Secure your Cognitive Services resource by configuring the firewall to block all connections on the public endpoint for the Cognitive Services service.
- Increase security for the VNet, by enabling you to block exfiltration of data from the VNet.
- Securely connect to Cognitive Services resources from on-premises networks that connect to the VNet using [VPN](#) or [ExpressRoutes](#) with private-peering.

Conceptual overview

A private endpoint is a special network interface for an Azure resource in your [VNet](#). Creating a private endpoint for your Cognitive Services resource provides secure connectivity between clients in your VNet and your resource. The private endpoint is assigned an IP address from the IP address range of your VNet. The connection between the private endpoint and the Cognitive Services service uses a secure private link.

Applications in the VNet can connect to the service over the private endpoint seamlessly, using the same connection strings and authorization mechanisms that they would use otherwise. The exception is the Speech Services, which require a separate endpoint. See the section on [Private endpoints with the Speech Services](#). Private endpoints can be used with all protocols supported by the Cognitive Services resource, including REST.

Private endpoints can be created in subnets that use [Service Endpoints](#). Clients in a subnet can connect to one Cognitive Services resource using private endpoint, while using service endpoints to access others.

When you create a private endpoint for a Cognitive Services resource in your VNet, a consent request is sent for approval to the Cognitive Services resource owner. If the user requesting the creation of the private endpoint is also an owner of the resource, this consent request is automatically approved.

Cognitive Services resource owners can manage consent requests and the private endpoints, through the '*Private endpoints*' tab for the Cognitive Services resource in the [Azure portal](#).

Private endpoints

When creating the private endpoint, you must specify the Cognitive Services resource it connects to. For more information on creating a private endpoint, see:

- [Create a private endpoint using the Private Link Center in the Azure portal](#)
- [Create a private endpoint using Azure CLI](#)
- [Create a private endpoint using Azure PowerShell](#)

Connecting to private endpoints

Clients on a VNet using the private endpoint should use the same connection string for the Cognitive Services resource as clients connecting to the public endpoint. The exception is the Speech Services, which require a separate endpoint. See the section on [Private endpoints with the Speech Services](#). We rely upon DNS resolution to automatically route the connections from the VNet to the Cognitive Services resource over a private link.

We create a [private DNS zone](#) attached to the VNet with the necessary updates for the private endpoints, by default. However, if you're using your own DNS server, you may need to make additional changes to your DNS configuration. The section on [DNS changes](#) below describes the updates required for private endpoints.

Private endpoints with the Speech Services

See [Using Speech Services with private endpoints provided by Azure Private Link](#).

DNS changes for private endpoints

When you create a private endpoint, the DNS CNAME resource record for the Cognitive Services resource is updated to an alias in a subdomain with the prefix '*privatelink*'. By default, we also create a [private DNS zone](#), corresponding to the '*privatelink*' subdomain, with the DNS A resource records for the private endpoints.

When you resolve the endpoint URL from outside the VNet with the private endpoint, it resolves to the public endpoint of the Cognitive Services resource. When resolved from

the VNet hosting the private endpoint, the endpoint URL resolves to the private endpoint's IP address.

This approach enables access to the Cognitive Services resource using the same connection string for clients in the VNet hosting the private endpoints and clients outside the VNet.

If you are using a custom DNS server on your network, clients must be able to resolve the fully qualified domain name (FQDN) for the Cognitive Services resource endpoint to the private endpoint IP address. Configure your DNS server to delegate your private link subdomain to the private DNS zone for the VNet.

💡 Tip

When using a custom or on-premises DNS server, you should configure your DNS server to resolve the Cognitive Services resource name in the 'privatelink' subdomain to the private endpoint IP address. You can do this by delegating the 'privatelink' subdomain to the private DNS zone of the VNet, or configuring the DNS zone on your DNS server and adding the DNS A records.

For more information on configuring your own DNS server to support private endpoints, refer to the following articles:

- [Name resolution for resources in Azure virtual networks](#)
- [DNS configuration for private endpoints](#)

Pricing

For pricing details, see [Azure Private Link pricing](#).

Next steps

- Explore the various [Azure Cognitive Services](#)
- Learn more about [Azure Virtual Network Service Endpoints](#)

Speech Services in sovereign clouds

Article • 09/20/2022 • 3 minutes to read

Azure Government (United States)

Available to US government entities and their partners only. See more information about Azure Government [here](#) and [here](#).

- **Azure portal:**
 - <https://portal.azure.us/>
- **Regions:**
 - US Gov Arizona
 - US Gov Virginia
- **Available pricing tiers:**
 - Free (F0) and Standard (S0). See more details [here](#)
- **Supported features:**
 - Speech-to-text
 - Custom speech (Acoustic Model (AM) and Language Model (LM) adaptation)
 - [Speech Studio](#)
 - Text-to-speech
 - Standard voice
 - Neural voice
 - Speech translation
- **Unsupported features:**
 - Custom Voice
- **Supported languages:**
 - See the list of supported languages [here](#)

Endpoint information

This section contains Speech Services endpoint information for the usage with [Speech SDK](#), [Speech-to-text REST API](#), and [Text-to-speech REST API](#).

Speech Services REST API

Speech Services REST API endpoints in Azure Government have the following format:

REST API type	Endpoint format
/ operation	

REST API type	Endpoint format
Access token	<code>https://<REGION_IDENTIFIER>.api.cognitive.microsoft.us/sts/v1.0/issueToken</code>
Speech-to-text REST API	<code>https://<REGION_IDENTIFIER>.api.cognitive.microsoft.us/<URL_PATH></code>
Speech-to-text REST API for short audio	<code>https://<REGION_IDENTIFIER>.stt.speech.azure.us/<URL_PATH></code>
Text-to-speech REST API	<code>https://<REGION_IDENTIFIER>.tts.speech.azure.us/<URL_PATH></code>

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

Region identifier	
US Gov Arizona	<code>usgovarizona</code>
US Gov Virginia	<code>usgovvirginia</code>

Speech SDK

For [Speech SDK](#) in sovereign clouds you need to use "from host / with host" instantiation of `SpeechConfig` class or `--host` option of [Speech CLI](#). (You may also use "from endpoint / with endpoint" instantiation and `--endpoint` Speech CLI option).

`SpeechConfig` class should be instantiated like this:

```
C#
```

```
C#
```

```
var config = SpeechConfig.FromHost(usGovHost, subscriptionKey);
```

Speech CLI should be used like this (note the `--host` option):

```
dos
```

```
spx recognize --host "usGovHost" --file myaudio.wav
```

Replace `subscriptionKey` with your Speech resource key. Replace `usGovHost` with the expression matching the required service offering and the region of your subscription from this table:

Region / Service offering	Host expression
US Gov Arizona	
Speech-to-text	<code>wss://usgovarizona.stt.speech.azure.us</code>
Text-to-Speech	<code>https://usgovarizona.tts.speech.azure.us</code>
US Gov Virginia	
Speech-to-text	<code>wss://usgovvirginia.stt.speech.azure.us</code>
Text-to-Speech	<code>https://usgovvirginia.tts.speech.azure.us</code>

Azure China

Available to organizations with a business presence in China. See more information about Azure China [here](#).

- **Azure portal:**
 - <https://portal.azure.cn/>
- **Regions:**
 - China East 2
 - China North 2
- **Available pricing tiers:**
 - Free (F0) and Standard (S0). See more details [here](#)
- **Supported features:**
 - Speech-to-text
 - Custom speech (Acoustic Model (AM) and Language Model (LM) adaptation)
 - [Speech Studio](#)
 - [Pronunciation assessment](#)
 - Text-to-speech
 - Standard voice
 - Neural voice
 - Speech translator
- **Unsupported features:**
 - Custom Voice
- **Supported languages:**
 - See the list of supported languages [here](#)

Endpoint information

This section contains Speech Services endpoint information for the usage with [Speech SDK](#), [Speech-to-text REST API](#), and [Text-to-speech REST API](#).

Speech Services REST API

Speech Services REST API endpoints in Azure China have the following format:

REST API type / operation	Endpoint format
Access token	<code>https://<REGION_IDENTIFIER>.api.cognitive.azure.cn/sts/v1.0/issueToken</code>
Speech-to-text REST API	<code>https://<REGION_IDENTIFIER>.api.cognitive.azure.cn/<URL_PATH></code>
Speech-to-text REST API for short audio	<code>https://<REGION_IDENTIFIER>.stt.speech.azure.cn/<URL_PATH></code>
Text-to-speech REST API	<code>https://<REGION_IDENTIFIER>.tts.speech.azure.cn/<URL_PATH></code>

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

Region identifier	
China East 2	<code>chinaeast2</code>
China North 2	<code>chinanorth2</code>

Speech SDK

For [Speech SDK](#) in sovereign clouds you need to use "from host / with host" instantiation of `SpeechConfig` class or `--host` option of [Speech CLI](#). (You may also use "from endpoint / with endpoint" instantiation and `--endpoint` Speech CLI option).

`SpeechConfig` class should be instantiated like this:

```
C#  
  
C#  
  
var config = SpeechConfig.FromHost("azCnHost", subscriptionKey);
```

Speech CLI should be used like this (note the `--host` option):

```
dos  
spx recognize --host "azCnHost" --file myaudio.wav
```

Replace `subscriptionKey` with your Speech resource key. Replace `azCnHost` with the expression matching the required service offering and the region of your subscription from this table:

Region / Service offering	Host expression
China East 2	
Speech-to-text	<code>wss://chinaeast2.stt.speech.azure.cn</code>
Text-to-Speech	<code>https://chinaeast2.tts.speech.azure.cn</code>
China North 2	
Speech-to-text	<code>wss://chinanorth2.stt.speech.azure.cn</code>
Text-to-Speech	<code>https://chinanorth2.tts.speech.azure.cn</code>

Speech service encryption of data at rest

Article • 02/20/2022 • 2 minutes to read

Speech Service automatically encrypts your data when it is persisted to the cloud. Speech service encryption protects your data and to help you to meet your organizational security and compliance commitments.

About Cognitive Services encryption

Data is encrypted and decrypted using [FIPS 140-2](#) compliant [256-bit AES](#) encryption. Encryption and decryption are transparent, meaning encryption and access are managed for you. Your data is secure by default and you don't need to modify your code or applications to take advantage of encryption.

About encryption key management

When you use Custom Speech and Custom Voice, Speech service may store following data in the cloud:

- Speech trace data - only if you turn the trace on for your custom endpoint
- Uploaded training and test data

By default, your data are stored in Microsoft's storage and your subscription uses Microsoft-managed encryption keys. You also have an option to prepare your own storage account. Access to the store is managed by the Managed Identity, and Speech service cannot directly access to your own data, such as speech trace data, customization training data and custom models.

For more information about Managed Identity, see [What are managed identities](#).

In the meantime, when you use Custom Command, you can manage your subscription with your own encryption keys. Customer-managed keys (CMK), also known as bring your own key (BYOK), offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data. For more information about Custom Command and CMK, see [Custom Commands encryption of data at rest](#).

Bring your own storage (BYOS) for customization and logging

To request access to bring your own storage, fill out and submit the [Speech service - bring your own storage \(BYOS\) request form](#). Once approved, you'll need to create your own storage account to store the data required for customization and logging. When adding a storage account, the Speech service resource will enable a system assigned managed identity.

Important

The user account you use to create a Speech resource with BYOS functionality enabled should be assigned the [Owner role at the Azure subscription scope](#). Otherwise you will get an authorization error during the resource provisioning.

After the system assigned managed identity is enabled, this resource will be registered with Azure Active Directory (AAD). After being registered, the managed identity will be given access to the storage account. For more about managed identities, see [What are managed identities](#).

Important

If you disable system assigned managed identities, access to the storage account will be removed. This will cause the parts of the Speech service that require access to the storage account to stop working.

The Speech service doesn't currently support Customer Lockbox. However, customer data can be stored using BYOS, allowing you to achieve similar data controls to [Customer Lockbox](#). Keep in mind that Speech service data stays and is processed in the region where the Speech resource was created. This applies to any data at rest and data in transit. When using customization features, like Custom Speech and Custom Voice, all customer data is transferred, stored, and processed in the same region where your BYOS (if used) and Speech service resource reside.

Important

Microsoft **does not** use customer data to improve its Speech models. Additionally, if endpoint logging is disabled and no customizations are used, then no customer data is stored.

Back up and recover speech customer resources

Article • 01/11/2023 • 5 minutes to read

The Speech service is [available in various regions](#). Speech resource keys are tied to a single region. When you acquire a key, you select a specific region, where your data, model and deployments reside.

Datasets for customer-created data assets, such as customized speech models, custom voice fonts and speaker recognition voice profiles, are also **available only within the service-deployed region**. Such assets are:

Custom Speech

- Training audio/text data
- Test audio/text data
- Customized speech models
- Log data

Custom Voice

- Training audio/text data
- Test audio/text data
- Custom voice fonts

Speaker Recognition

- Speaker enrollment audio
- Speaker voice signature

While some customers use our default endpoints to transcribe audio or standard voices for speech synthesis, other customers create assets for customization.

These assets are backed up regularly and automatically by the repositories themselves, so **no data loss will occur** if a region becomes unavailable. However, you must take steps to ensure service continuity if there's a region outage.

How to monitor service availability

If you use the default endpoints, you should configure your client code to monitor for errors. If errors persist, be prepared to redirect to another region where you have a Speech resource.

Follow these steps to configure your client to monitor for errors:

1. Find the [list of regionally available endpoints in our documentation](#).
2. Select a primary and one or more secondary/backup regions from the list.
3. From Azure portal, create Speech Service resources for each region.
 - If you have set a specific quota, you may also consider setting the same quota in the backup regions. See details in [Speech service Quotas and Limits](#).
4. Each region has its own STS token service. For the primary region and any backup regions your client configuration file needs to know the:
 - Regional Speech service endpoints
 - [Regional key and the region code](#)
5. Configure your code to monitor for connectivity errors (typically connection timeouts and service unavailability errors). Here's sample code in C#: [GitHub: Adding Sample for showing a possible candidate for switching regions](#).
 - a. Since networks experience transient errors, for single connectivity issue occurrences, the suggestion is to retry.
 - b. For persistence redirect traffic to the new STS token service and Speech service endpoint. (For Text-to-Speech, reference sample code: [GitHub: TTS public voice switching region](#)).

The recovery from regional failures for this usage type can be instantaneous and at a low cost. All that is required is the development of this functionality on the client side. The data loss that will incur assuming no backup of the audio stream will be minimal.

Custom endpoint recovery

Data assets, models or deployments in one region can't be made visible or accessible in any other region.

You should create Speech Service resources in both a main and a secondary region by following the same steps as used for default endpoints.

Custom Speech

Custom Speech Service doesn't support automatic failover. We suggest the following steps to prepare for manual or automatic failover implemented in your client code. In

these steps, you replicate custom models in a secondary region. With this preparation, your client code can switch to a secondary region when the primary region fails.

1. Create your custom model in one main region (Primary).
2. Run the [Models_CopyTo](#) operation to replicate the custom model to all prepared regions (Secondary).
3. Go to Speech Studio to load the copied model and create a new endpoint in the secondary region. See how to deploy a new model in [Deploy a Custom Speech model](#).
 - If you have set a specific quota, also consider setting the same quota in the backup regions. See details in [Speech service Quotas and Limits](#).
4. Configure your client to fail over on persistent errors as with the default endpoints usage.

Your client code can monitor availability of your deployed models in your primary region, and redirect their audio traffic to the secondary region when the primary fails. If you don't require real-time failover, you can still follow these steps to prepare for a manual failover.

Offline failover

If you don't require real-time failover you can decide to import your data, create and deploy your models in the secondary region at a later time with the understanding that these tasks will take time to complete.

Failover time requirements

This section provides general guidance about timing. The times were recorded to estimate offline failover using a [representative test data set](#).

- Data upload to new region: **15mins**
- Acoustic/language model creation: **6 hours (depending on the data volume)**
- Model evaluation: **30 mins**
- Endpoint deployment: **10 mins**
- Model copy API call: **10 mins**
- Client code reconfiguration and deployment: **Depending on the client system**

It's nonetheless advisable to create keys for a primary and secondary region for production models with real-time requirements.

Custom Voice

Custom Voice doesn't support automatic failover. Handle real-time synthesis failures with these two options.

Option 1: Fail over to public voice in the same region.

When custom voice real-time synthesis fails, fail over to a public voice (client sample code: [GitHub: custom voice failover to public voice ↗](#)).

Check the [public voices available](#). You can also change the sample code above if you would like to fail over to a different voice or in a different region.

Option 2: Fail over to custom voice on another region.

1. Create and deploy your custom voice in one main region (primary).
2. Copy your custom voice model to another region (the secondary region) in [Speech Studio ↗](#).
3. Go to Speech Studio and switch to the Speech resource in the secondary region.
Load the copied model and create a new endpoint.
 - Voice model deployment usually finishes **in 3 minutes**.
 - Each endpoint is subject to extra charges. [Check the pricing for model hosting here ↗](#).
4. Configure your client to fail over to the secondary region. See sample code in C#:
[GitHub: custom voice failover to secondary region ↗](#).

Speaker Recognition

Speaker Recognition uses [Azure paired regions](#) to automatically fail over operations. Speaker enrollments and voice signatures are backed up regularly to prevent data loss and to be used if there's an outage.

During an outage, Speaker Recognition service will automatically fail over to a paired region and use the backed-up data to continue processing requests until the main region is back online.

Enable logging in the Speech SDK

Article • 10/21/2022 • 4 minutes to read

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 is handled by a static class in Speech SDK's native library. You can turn on logging for any Speech SDK recognizer or synthesizer instance. All instances in the same process write log entries to the same log file.

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`:

C#

```
config SetProperty(PropertyId.Speech_LogFilename, "LogfilePathAndName");
```

Java

```
config.setProperty(PropertyId.Speech_LogFilename, "LogfilePathAndName");
```

C++

```
config->SetProperty(PropertyId::Speech_LogFilename, "LogfilePathAndName");
```

Python

```
config.set_property(speechsdk.PropertyId.Speech_LogFilename,  
"LogfilePathAndName")
```

Objective-C

```
[config SetPropertyTo:@"LogfilePathAndName" byId:SPXSpeechLogFilename];
```

Go

```
import ("github.com/Microsoft/cognitive-services-speech-sdk-go/common")
```

```
config SetProperty(common.SpeechLogFilename, "LogfilePathAndName")
```

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 be places log files in one of the application data locations (local, roaming, or temporary). A log file can be created in the local application folder:

C#

```
StorageFolder storageFolder = ApplicationData.Current.LocalFolder;
StorageFile logFile = await storageFolder.CreateFileAsync("logfile.txt",
CreationCollisionOption.ReplaceExisting);
config SetProperty(PropertyId.Speech_LogFilename, logFile.Path);
```

Within a Unity UWP application, a log file can be created using the application persistent data path folder as follows:

C#

```
#if ENABLE_WINMD_SUPPORT
    string logFile = Application.persistentDataPath + "/logfile.txt";
    config SetProperty(PropertyId.Speech_LogFilename, logFile);
#endif
```

For more about file access permissions in UWP applications, see [File access permissions](#).

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.

Java

```
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:

XML

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="...">
...
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
...
</manifest>
```

Within a Unity Android application, the log file can be created using the application persistent data path folder as follows:

C#

```
string logfile = Application.persistentDataPath + "/logfile.txt";
config SetProperty(PropertyId.Speech_LogFilename, logfile);
```

In addition, you need to also set write permission in your Unity Player settings for Android to "External (SDCard)". The log will be written to a directory you can get using a tool such as AndroidStudio Device File Explorer. The exact directory path may vary between Android devices, location is typically the `sdcard/Android/data/your-app-pagename/files` directory.

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.

If you are using Objective-C on iOS, use the following code snippet to create a log file in the application document directory:

Objective-C

```
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:

XML

```
<key>UIFileSharingEnabled</key>
<true/>
<key>LSSupportsOpeningDocumentsInPlace</key>
<true/>
```

If you are using Swift on iOS, please use the following code snippet to enable logs:

Swift

```
let documentsDirectoryPathString =
NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask,
true).first!
let documentsDirectoryPath = NSURL(string: documentsDirectoryPathString)!
let filePath = documentsDirectoryPath.appendingPathComponent("swift.log")
self.speechConfig!.setPropertyTo(filePath!.absoluteString, by:
SPXPropertyId.speechLogFilename)
```

More about iOS File System is available [here ↗](#).

Logging with multiple recognizers

Although a log file output path is specified as a configuration property into a `SpeechRecognizer` or other SDK object, SDK logging is a singleton, *process-wide* facility

with no concept of individual instances. You can think of this as the `SpeechRecognizer` constructor (or similar) implicitly calling a static and internal "Configure Global Logging" routine with the property data available in the corresponding `SpeechConfig`.

This means that you cannot, as an example, configure six parallel recognizers to output simultaneously to six separate files. Instead, the latest recognizer created will configure the global logging instance to output to the file specified in its configuration properties and all SDK logging will be emitted to that file.

This also means that the lifetime of the object that configured logging is not tied to the duration of logging. Logging will not stop in response to the release of an SDK object and will continue as long as no new logging configuration is provided. Once started, process-wide logging may be stopped by setting the log file path to an empty string when creating a new object.

To reduce potential confusion when configuring logging for multiple instances, it may be useful to abstract control of logging from objects doing real work. An example pair of helper routines:

C++

```
void EnableSpeechSdkLogging(const char* relativePath)
{
    auto configForLogging = SpeechConfig::FromSubscription("unused_key",
"unused_region");
    configForLogging->SetProperty(PropertyId::Speech_LogFilename,
relativePath);
    auto emptyAudioConfig =
AudioConfig::FromStreamInput(AudioInputStream::CreatePushStream());
    auto temporaryRecognizer =
SpeechRecognizer::FromConfig(configForLogging, emptyAudioConfig);
}

void DisableSpeechSdkLogging()
{
    EnableSpeechSdkLogging("");
}
```

Next steps

[Explore our samples on GitHub](#)

Troubleshoot the Speech SDK

Article • 12/09/2022 • 3 minutes to read

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

Authentication failed

You might observe one of several authentication errors, depending on the programming environment, API, or SDK. Here are some example errors:

- Did you set the speech resource key and region values?
- AuthenticationFailure
- HTTP 403 Forbidden or HTTP 401 Unauthorized. Connection requests without a valid `Ocp-Apim-Subscription-Key` or `Authorization` header are rejected with a status of 403 or 401.
- ValueError: cannot construct `SpeechConfig` with the given arguments (or a variation of this message). This error could be observed, for example, when you run one of the Speech SDK for Python quickstarts without setting environment variables. You might also see it when you set the environment variables to something invalid such as your key or region.
- Exception with an error code: 0x5. This access denied error could be observed, for example, when you run one of the Speech SDK for C# quickstarts without setting environment variables.

For baseline authentication troubleshooting tips, see [validate your resource key](#) and [validate an authorization token](#). For more information about confirming credentials, see [get the keys for your resource](#).

Validate your resource key

You can verify that you have a valid resource key by running one of the following commands.

Note

Replace `YOUR_RESOURCE_KEY` and `YOUR_REGION` with your own resource key and associated region.

```
$FetchTokenHeader = @{
    'Content-type'='application/x-www-form-urlencoded'
    'Content-Length'= '0'
    'Ocp-Apim-Subscription-Key' = 'YOUR_RESOURCE_KEY'
}
$OAuthToken = Invoke-RestMethod -Method POST -Uri
https://YOUR_REGION.api.cognitive.microsoft.com/sts/v1.0/issueToken -
Headers $FetchTokenHeader
$OAuthToken
```

If you entered a valid resource key, the command returns an authorization token, otherwise an error is returned.

Validate an authorization token

If you're using an authorization token for authentication, you might see an authentication error because:

- The authorization token is invalid
- The authorization token is expired

If you use an authorization token for authentication, run one of the following commands to verify that the authorization token is still valid. Tokens are valid for 10 minutes.

ⓘ Note

Replace `YOUR_AUDIO_FILE` with the path to your prerecorded audio file. Replace `YOUR_ACCESS_TOKEN` with the authorization token returned in the preceding step.

Replace `YOUR_REGION` with the correct region.

```
$SpeechServiceURI =
'https://YOUR_REGION.stt.speech.microsoft.com/speech/recognition/interactive/cognitiveservices/v1?language=en-US'

# $OAuthToken is the authorization token returned by the token service.
$RecoRequestHeader = @{
    'Authorization' = 'Bearer ' + $OAuthToken
    'Transfer-Encoding' = 'chunked'
    'Content-type' = 'audio/wav; codec=audio/pcm; samplerate=16000'
```

```
}

# Read audio into byte array.
$audioBytes = [System.IO.File]::ReadAllBytes("YOUR_AUDIO_FILE")

$RecoResponse = Invoke-RestMethod -Method POST -Uri $SpeechServiceURI -
Headers $RecoRequestHeader -Body $audioBytes

# Show the result.
$RecoResponse
```

If you entered a valid authorization token, the command returns the transcription for your audio file, otherwise an error is returned.

InitialSilenceTimeout via RecognitionStatus

This issue usually is observed with [single-shot recognition](#) of a single utterance. For example, the error can be returned under the following circumstances:

- The audio begins with a long stretch of silence. In that case, the service stops the recognition after a few seconds and returns `InitialSilenceTimeout`.
- The audio uses an unsupported codec format, which causes the audio data to be treated as silence.

It's OK to have silence at the beginning of audio, but only when you use [continuous recognition](#).

SPXERR_AUDIO_SYS_LIBRARY_NOT_FOUND

This can be returned, for example, when multiple versions of Python have been installed, or if you're not using a supported version of Python. You can try using a different python interpreter or uninstall all python versions and re-install the latest version of python and the Speech SDK.

HTTP 400 Bad Request

This error usually occurs when the request body contains invalid audio data. Only WAV format is supported. Also, check the request's headers to make sure you specify appropriate values for `Content-Type` and `Content-Length`.

HTTP 408 Request Timeout

The error most likely occurs because no audio data is being sent to the service. This error also might be caused by network issues.

Connection closed or timeout

There is a known issue on Windows 11 that might affect some types of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) connections. These connections might have handshake failures. For developers, the affected connections are likely to send multiple frames followed by a partial frame with a size of less than 5 bytes within a single input buffer. If the connection fails, your app will receive the error such as, "USP error", "Connection closed", "ServiceTimeout", or "SEC_E_ILLEGAL_MESSAGE".

There is an out of band update available for Windows 11 that fixes these issues. The update may be manually installed by following the instructions here:

- [Windows 11 21H2 ↗](#)
- [Windows 11 22H2 ↗](#)

The issue started October 12th, 2022 and should be resolved via Windows update in November, 2022.

Next steps

- [Review the release notes](#)
-

Additional resources

Documentation

[Create a custom keyword quickstart - Speech service - Azure Cognitive Services](#)

When a user speaks the keyword, your device sends their dictation to the cloud, until the user stops speaking. Customizing your keyword is an effective way to differentiate your device and strengthen your branding.

[Language identification - Speech service - Azure Cognitive Services](#)

Language identification is used to determine the language being spoken in audio when compared against a list of provided languages.

[Release notes - Speech Service - Azure Cognitive Services](#)

A running log of Speech Service feature releases, improvements, bug fixes, and known issues.

[Speech SDK audio input stream concepts - Azure Cognitive Services](#)

An overview of the capabilities of the Speech SDK audio input stream API.

[Speech-to-text REST API for short audio - Speech service - Azure Cognitive Services](#)

Learn how to use Speech-to-text REST API for short audio to convert speech to text.

[microsoft-cognitiveservices-speech-sdk package](#)

[Train a Custom Speech model - Speech service - Azure Cognitive Services](#)

Learn how to train Custom Speech models. Training a speech-to-text model can improve recognition accuracy for the Microsoft base model or a custom model.

[How to use compressed input audio - Speech service - Azure Cognitive Services](#)

Learn how to use compressed input audio the Speech SDK and CLI.

[Show 5 more](#)

How to get Speech-to-text Session ID and Transcription ID

Article • 01/02/2023 • 2 minutes to read

If you use [Speech-to-text](#) and need to open a support case, you are often asked to provide a *Session ID* or *Transcription ID* of the problematic transcriptions to debug the issue. This article explains how to get these IDs.

ⓘ Note

- *Session ID* is used in [Online transcription](#) and [Translation](#).
- *Transcription ID* is used in [Batch transcription](#).

Getting Session ID for Online transcription and Translation. (Speech SDK and REST API for short audio).

[Online transcription](#) and [Translation](#) use either the [Speech SDK](#) or the [REST API](#) for short audio.

To get the Session ID, when using SDK you need to:

1. Enable application logging.
2. Find the Session ID inside the log.

If you use [Speech CLI](#), you can also get the Session ID interactively. See details [below](#).

In case of [Speech-to-text REST API for short audio](#) you need to "inject" the session information in the requests. See details [below](#).

Enable logging in the Speech SDK

Enable logging for your application as described in [this article](#).

Get Session ID from the log

Open the log file your application produced and look for `sessionId:`. The number, that would follow is the Session ID you need. In the log excerpt example below

`0b734c41faf8430380d493127bd44631` is the Session ID.

```
[874193]: 218ms SPX_DBG_TRACE_VERBOSE:  audio_stream_session.cpp:1238
[0000023981752A40]CSpxAudioStreamSession::FireSessionStartedEvent: Firing
SessionStarted event: SessionId: 0b734c41faf8430380d493127bd44631
```

Get Session ID using Speech CLI

If you use [Speech CLI](#), then you will see the Session ID in `SESSION STARTED` and `SESSION STOPPED` console messages.

You can also enable logging for your sessions and get the Session ID from the log file as described above. Run the appropriate Speech CLI command to get the information on using logs:

Console

```
spx help recognize log
```

Console

```
spx help translate log
```

Provide Session ID using REST API for short audio

Unlike Speech SDK, [Speech-to-text REST API for short audio](#) does not automatically generate a Session ID. You need to generate it yourself and provide it within the REST request.

Generate a GUID inside your code or using any standard tool. Use the GUID value *without dashes or other dividers*. As an example we will use

`9f4ffa5113a846eba289aa98b28e766f`.

As a part of your REST request use `X-ConnectionId=<GUID>` expression. For our example, a sample request will look like this:

HTTP

```
https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/
cognitiveservices/v1?language=en-US&X-
ConnectionId=9f4ffa5113a846eba289aa98b28e766f
```

9f4ffa5113a846eba289aa98b28e766f will be your Session ID.

Getting Transcription ID for Batch transcription

Batch transcription uses [Speech-to-text REST API](#).

The required Transcription ID is the GUID value contained in the main `self` element of the Response body returned by requests, like [Transcriptions_Create ↗](#).

The example below is the Response body of a `Create Transcription` request. GUID value `537216f8-0620-4a10-ae2d-00bdb423b36f` found in the first `self` element is the Transcription ID.

JSON

```
{  
  "self":  
    "https://japaneast.api.cognitive.microsoft.com/speechtotext/v3.1/transcrip  
tions/537216f8-0620-4a10-ae2d-00bdb423b36f",  
  "model": {  
    "self":  
      "https://japaneast.api.cognitive.microsoft.com/speechtotext/v3.1/models/base  
/824bd685-2d45-424d-bb65-c3fe99e32927"  
  },  
  "links": {  
    "files":  
      "https://japaneast.api.cognitive.microsoft.com/speechtotext/v3.1/transcrip  
tions/537216f8-0620-4a10-ae2d-00bdb423b36f/files"  
  },  
  "properties": {  
    "diarizationEnabled": false,  
    "wordLevelTimestampsEnabled": false,  
    "channels": [  
      0,  
      1  
    ],  
    "punctuationMode": "DictatedAndAutomatic",  
    "profanityFilterMode": "Masked"  
  },  
  "lastActionDateTime": "2021-11-19T14:09:51Z",  
  "status": "NotStarted",  
  "createdDateTime": "2021-11-19T14:09:51Z",  
  "locale": "ru-RU",  
  "displayName": "transcriptiontest"  
}
```

ⓘ Note

Use the same technique to determine different IDs required for debugging issues related to **Custom Speech**, like uploading a dataset using [Datasets_Create ↗](#) request.

ⓘ Note

You can also see all existing transcriptions and their Transcription IDs for a given Speech resource by using [Transcriptions_Get ↗](#) request.

Additional resources

📖 Documentation

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Speech service containers frequently asked questions \(FAQ\) - Azure Cognitive Services](#)

Install and run containers for speech-to-text and text-to-speech.

[Configure Speech containers - Azure Cognitive Services](#)

Speech service provides each container with a common configuration framework, so that you can easily configure and manage storage, logging and telemetry, and security settings for your containers.

[Model lifecycle of Custom Speech - Speech service - Azure Cognitive Services](#)

Custom Speech provides base models for training and lets you create custom models from your data. This article describes the timelines for models and for endpoints that use these models.

[How to back up and recover speech customer resources - Azure Cognitive Services](#)

Learn how to prepare for service outages with Custom Speech and Custom Voice.

[azure-cognitiveservices-speech package](#)

[azure.cognitiveservices.speech.SpeechRecognizer class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, language, source_language_config or auto_detect_source_language_config.

[CI/CD for Custom Speech - Speech service - Azure Cognitive Services](#)

Apply DevOps with Custom Speech and CI/CD workflows. Implement an existing DevOps solution for your own project.

[Show 5 more](#)

How to track Speech SDK memory usage

Article • 02/20/2022 • 2 minutes to read

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:

```
terminal

(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`:

```
C#
```

```
config SetProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

💡 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:

terminal

```
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.  
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`:

C#

```
config SetProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

💡 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

- Learn more about the Speech SDK

Configure RHEL/CentOS 7

Article • 06/21/2022 • 2 minutes to read

To use the Speech SDK on Red Hat Enterprise Linux (RHEL) 7 x64 and CentOS 7 x64, update the C++ compiler (for C++ development) and the shared C++ runtime library on your system.

Install dependencies

First install all general dependencies:

Bash

```
sudo rpm -Uvh https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm

# Install development tools and libraries
sudo yum update -y
sudo yum groupinstall -y "Development tools"
sudo yum install -y alsa-lib dotnet-sdk-2.1 java-1.8.0-openjdk-devel openssl
sudo yum install -y gstreamer1 gstreamer1-plugins-base gstreamer1-plugins-good gstreamer1-plugins-bad-free gstreamer1-plugins-ugly-free
```

C/C++ compiler and runtime libraries

Install the prerequisite packages with this command:

Bash

```
sudo yum install -y gmp-devel mpfr-devel libmpc-devel
```

Next update the compiler and runtime libraries:

Bash

```
# Build GCC 7.5.0 and runtimes and install them under /usr/local
curl https://ftp.gnu.org/gnu/gcc/gcc-7.5.0/gcc-7.5.0.tar.gz -O
tar -xf gcc-7.5.0.tar.gz
mkdir gcc-7.5.0-build && cd gcc-7.5.0-build
../gcc-7.5.0/configure --enable-languages=c,c++ --disable-bootstrap --
--disable-multilib --prefix=/usr/local
make -j$(nproc)
sudo make install-strip
```

If the updated compiler and libraries need to be deployed on several machines, you can simply copy them from under `/usr/local` to other machines. If only the runtime libraries are needed then the files in `/usr/local/lib64` will be enough.

Environment settings

Run the following commands to complete the configuration:

Bash

```
# Add updated C/C++ runtimes to the library path
# (this is required for any development/testing with Speech SDK)
export LD_LIBRARY_PATH=/usr/local/lib64:$LD_LIBRARY_PATH

# For C++ development only:
# - add the updated compiler to PATH
#   (note, /usr/local/bin should be already first in PATH on vanilla
#   systems)
# - add Speech SDK libraries from the Linux tar package to LD_LIBRARY_PATH
#   (note, use the actual path to extracted files!)
export PATH=/usr/local/bin:$PATH
hash -r # reset cached paths in the current shell session just in case
export LD_LIBRARY_PATH=/path/to/extracted/SpeechSDK-Linux-
<version>/lib/centos7-x64:$LD_LIBRARY_PATH
```

Note

The Linux .tar package contains specific libraries for RHEL/CentOS 7. These are in `lib/centos7-x64` as shown in the environment setting example for `LD_LIBRARY_PATH` above. Speech SDK libraries in `lib/x64` are for all the other supported Linux x64 distributions (including RHEL/CentOS 8) and don't work on RHEL/CentOS 7.

Next steps

[About the Speech SDK](#)

Generate a REST API client library for the Speech-to-text REST API

Article • 11/30/2022 • 2 minutes to read

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 area of the Speech Studio](#) can be completed programmatically using these APIs.

ⓘ Note

Speech service has several REST APIs for **Speech-to-text** and **Text-to-speech**.

However only **Speech-to-text REST API** is documented in the Swagger specification. See the documents referenced in the previous paragraph for the information on all other Speech Services REST APIs.

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 region of your Speech resource. You can confirm the region in the **Overview** part of your Speech resource settings in Azure portal. The complete list of supported regions is available [here](#).

1. In a browser, go to the Swagger specification for your [region](#):
`https://<your-region>.dev.cognitive.microsoft.com/docs/services/speech-to-text-api-v3-1`
2. On that page, click **API definition**, and click **Swagger**. Copy the URL of the page that appears.
3. In a new browser, go to <https://editor.swagger.io>
4. Click **File**, click **Import URL**, paste the URL, and click **OK**.

5. Click **Generate Client** and select **python**. The client library downloads to your computer in a `.zip` file.

6. Extract everything from the download. You might use `tar -xf` to extract everything.

7. Install the extracted module into your Python environment:

```
pip install path/to/package/python-client
```

8. The installed package is named `swagger_client`. Check that the installation has worked:

```
python -c "import swagger_client"
```

You can use the Python library that you generated with the [Speech service samples on GitHub](#).

Next steps

- [Speech service samples on GitHub](#).
- [Speech-to-text REST API](#)

Embedded Speech (preview)

Article • 01/20/2023 • 4 minutes to read

Embedded Speech is designed for on-device [speech-to-text](#) and [text-to-speech](#) scenarios where cloud connectivity is intermittent or unavailable. For example, you can use embedded speech in industrial equipment, a voice enabled air conditioning unit, or a car that might travel out of range. You can also develop hybrid cloud and offline solutions. For scenarios where your devices must be in a secure environment like a bank or government entity, you should first consider [disconnected containers](#).

ⓘ Important

Microsoft limits access to embedded speech. You can apply for access through the Azure Cognitive Services [embedded speech limited access review](#). For more information, see [Limited access for embedded speech](#).

Platform requirements

Embedded speech is included with the Speech SDK (version 1.24.1 and higher) for C#, C++, and Java. Refer to the general [Speech SDK installation requirements](#) for programming language and target platform specific details.

Choose your target environment

Windows

Requires Windows 10 or newer on x64 or ARM64 hardware.

The latest [Microsoft Visual C++ Redistributable for Visual Studio 2015-2022](#) must be installed regardless of the programming language used with the Speech SDK.

The Speech SDK for Java doesn't support Windows on ARM64.

Limitations

Embedded speech is only available with C#, C++, and Java SDKs. The other Speech SDKs, Speech CLI, and REST APIs don't support embedded speech.

Embedded speech recognition only supports mono 16 bit, 16-kHz PCM-encoded WAV audio.

Embedded neural voices only support 24-kHz sample rate.

Models and voices

For embedded speech, you'll need to download the speech recognition models for [speech-to-text](#) and voices for [text-to-speech](#). Instructions will be provided upon successful completion of the [limited access review](#) process.

The following [speech-to-text](#) models are available: de-DE, en-AU, en-CA, en-GB, en-IE, en-IN, en-NZ, en-US, es-ES, es-MX, fr-CA, fr-FR, hi-IN, it-IT, ja-JP, ko-KR, nl-NL, pt-BR, ru-RU, sv-SE, tr-TR, zh-CN, zh-HK, and zh-TW.

The following [text-to-speech](#) locales and voices are available:

Locale (BCP-47)	Language	Text-to-speech voices
de-DE	German (Germany)	de-DE-KatjaNeural (Female) de-DE-ConradNeural (Male)
en-AU	English (Australia)	en-AU-AnnetteNeural (Female) en-AU-WilliamNeural (Male)
en-CA	English (Canada)	en-CA-ClaraNeural (Female) en-CA-LiamNeural (Male)
en-GB	English (United Kingdom)	en-GB-LibbyNeural (Female) en-GB-RyanNeural (Male)
en-US	English (United States)	en-US-AriaNeural (Female) en-US-GuyNeural (Male) en-US-JennyNeural (Female)
es-ES	Spanish (Spain)	es-ES-ElviraNeural (Female) es-ES-AlvaroNeural (Male)
es-MX	Spanish (Mexico)	es-MX-DaliaNeural (Female) es-MX-JorgeNeural (Male)
fr-CA	French (Canada)	fr-CA-SylvieNeural (Female) fr-CA-JeanNeural (Male)
fr-FR	French (France)	fr-FR-DeniseNeural (Female) fr-FR-HenriNeural (Male)

Locale (BCP-47)	Language	Text-to-speech voices
it-IT	Italian (Italy)	it-IT-IsabellaNeural (Female) it-IT-DiegoNeural (Male)
ja-JP	Japanese (Japan)	ja-JP-NanamiNeural (Female) ja-JP-KeitaNeural (Male)
ko-KR	Korean (Korea)	ko-KR-SunHiNeural (Female) ko-KR-InJoonNeural (Male)
pt-BR	Portuguese (Brazil)	pt-BR-FranciscaNeural (Female) pt-BR-AntonioNeural (Male)
zh-CN	Chinese (Mandarin, Simplified)	zh-CN-XiaoxiaoNeural (Female) zh-CN-YunxiNeural (Male)

Embedded speech configuration

For cloud connected applications, as shown in most Speech SDK samples, you use the `SpeechConfig` object with a Speech resource key and region. For embedded speech, you don't use a Speech resource. Instead of a cloud resource, you use the [models and voices](#) that you downloaded to your local device.

Use the `EmbeddedSpeechConfig` object to set the location of the models or voices. If your application is used for both speech-to-text and text-to-speech, you can use the same `EmbeddedSpeechConfig` object to set the location of the models and voices.

C#

```
// Provide the location of the models and voices.
List<string> paths = new List<string>();
paths.Add("C:\\dev\\embedded-speech\\stt-models");
paths.Add("C:\\dev\\embedded-speech\\tts-voices");
var embeddedSpeechConfig = EmbeddedSpeechConfig.FromPaths(paths.ToArray());

// For speech-to-text
embeddedSpeechConfig.SetSpeechRecognitionModel(
    "Microsoft Speech Recognizer en-US FP Model V8",
    Environment.GetEnvironmentVariable("MODEL_KEY"));

// For text-to-speech
embeddedSpeechConfig.SetSpeechSynthesisVoice(
    "Microsoft Server Speech Text to Speech Voice (en-US, JennyNeural)",
    Environment.GetEnvironmentVariable("VOICE_KEY"));
embeddedSpeechConfig.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm);
```

You can find ready to use embedded speech samples at [GitHub](#).

- [C# \(.NET 6.0\)](#)
- [C# for Unity](#)

Hybrid speech

Hybrid speech with the `HybridSpeechConfig` object uses the cloud speech service by default and embedded speech as a fallback in case cloud connectivity is limited or slow.

With hybrid speech configuration for [speech-to-text](#) (recognition models), embedded speech is used when connection to the cloud service fails after repeated attempts. Recognition may continue using the cloud service again if the connection is later resumed.

With hybrid speech configuration for [text-to-speech](#) (voices), embedded and cloud synthesis are run in parallel and the result is selected based on which one gives a faster response. The best result is evaluated on each synthesis request.

Cloud speech

For cloud speech, you use the `SpeechConfig` object, as shown in the [speech-to-text quickstart](#) and [text-to-speech quickstart](#). To run the quickstarts for embedded speech, you can replace `SpeechConfig` with `EmbeddedSpeechConfig` or `HybridSpeechConfig`. Most of the other speech recognition and synthesis code are the same, whether using cloud, embedded, or hybrid configuration.

Next steps

- [Quickstart: Recognize and convert speech to text](#)
- [Quickstart: Convert text to speech](#)

Additional resources

Documentation

[Quickstart: The Speech CLI - Speech service - Azure Cognitive Services](#)

In this Azure Speech CLI quickstart, you interact with speech-to-text, text-to-speech, and speech translation without having to write code.

[azure.cognitiveservices.speech.SpeechSynthesizer class](#)

A speech synthesizer.

[Troubleshoot the Speech SDK - Speech service - Azure Cognitive Services](#)

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[Custom Neural Voice Lite - Speech service - Azure Cognitive Services](#)

Use Custom Neural Voice Lite to demo and evaluate Custom Neural Voice before investing in professional recordings to create a higher-quality voice.

[Run batch operations with the Speech CLI - Speech service - Azure Cognitive Services](#)

Learn how to do batch speech to text (speech recognition), batch text to speech (speech synthesis) with the Speech CLI.

[Create a project for Custom Neural Voice - Speech service - Azure Cognitive Services](#)

Learn how to create a Custom Neural Voice project that contains data, models, tests, and endpoints in Speech Studio.

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without having to write any code.

[Show 5 more](#)

Azure Cognitive Services containers

Azure Cognitive Services offers several containers to use AI on premises. Containers give you the flexibility to bring Cognitive Services closer to your data for compliance, security or other operational reasons.

About Cognitive Services containers



[What are Cognitive Services containers?](#)

[Frequently Asked Questions](#)

Container recipes



[Container reuse recipe](#)

[Deploy on Azure Container Instances](#)

[Deploy on Kubernetes](#)

[Deploy with Docker compose](#)

Vision containers



[Read OCR](#)

[Spatial analysis](#)

[Face](#)

Language containers



[Sentiment Analysis](#)

[Key phrase extraction](#)
[Text language detection](#)
[Text Analytics for health](#)
[LUIS container](#)
[Translator container](#)

Speech containers

 [HOW-TO GUIDE](#)
[Speech to text](#)
[Custom Speech to text](#)
[Neural text to speech](#)
[Speech language identification](#)

Decision container

 [HOW-TO GUIDE](#)
[Anomaly Detector](#)

Disconnected containers

 [HOW-TO GUIDE](#)
[Run containers disconnected from the internet](#)
[Frequently Asked Questions about disconnected containers](#)

Install and run Docker containers for the Speech service APIs

Article • 02/01/2023 • 24 minutes to read

By using containers, you can run *some* of the Azure Cognitive Services 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.

With Speech containers, you can build a speech application architecture that's optimized for both robust cloud capabilities and edge locality. Several containers are available, which use the same [pricing](#) as the cloud-based Azure Speech services.

ⓘ Important

We retired the standard speech synthesis voices and text-to-speech container on August 31, 2021. Consider migrating your applications to use the neural text-to-speech container instead. For more information on updating your application, see [Migrate from standard voice to prebuilt neural voice](#).

Container	Features	Latest	Release status
Speech-to-text	Analyzes sentiment and transcribes continuous real-time speech or batch audio recordings with intermediate results.	3.10.0	Generally available
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.	3.10.0	Generally available
Speech language identification	Detects the language spoken in audio files.	1.5.0	Preview
Neural text-to-speech	Converts text to natural-sounding speech by using deep neural network technology, which allows for more natural synthesized speech.	2.9.0	Generally available

Prerequisites

ⓘ Important

- To use the Speech containers, you must submit an online request and have it approved. For more information, see the "Request approval to run the container" section.
- *Generally available* containers meet Microsoft's stability and support requirements. Containers in *preview* are still under development.

You must meet the following prerequisites before you use Speech service containers. If you don't have an Azure subscription, create a [free account](#) before you begin. You need:

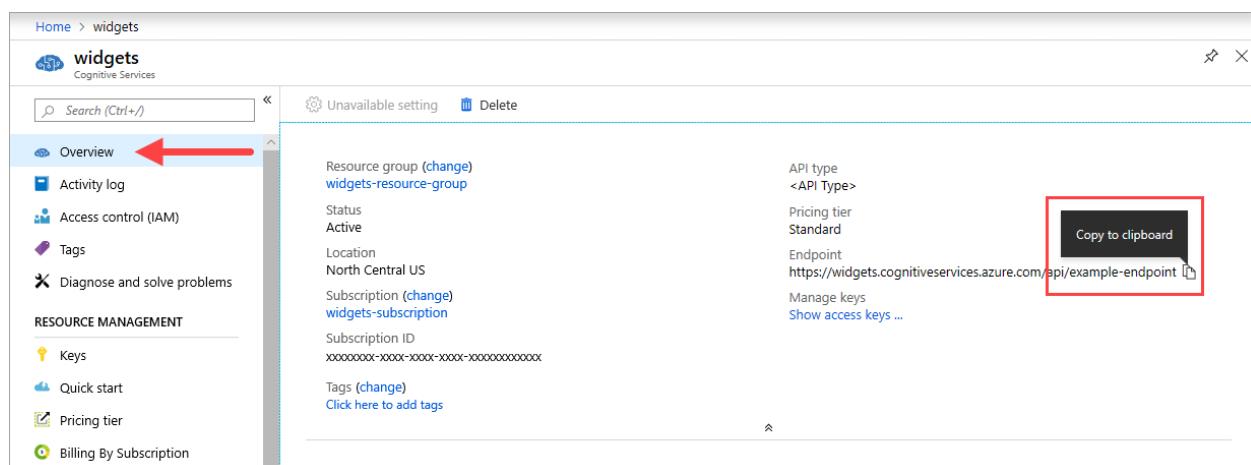
- [Docker](#) installed on a host computer. Docker must be configured to allow the containers to connect with and send billing data to Azure.
 - On Windows, Docker must also be configured to support Linux containers.
 - You should have a basic understanding of [Docker concepts](#).
- A [Speech service resource](#) with the free (F0) or standard (S) pricing tier.

Gather required parameters

Three primary parameters for all Cognitive Services containers are required. The Microsoft Software License Terms must be present with a value of **accept**. An Endpoint URI and API key are also needed.

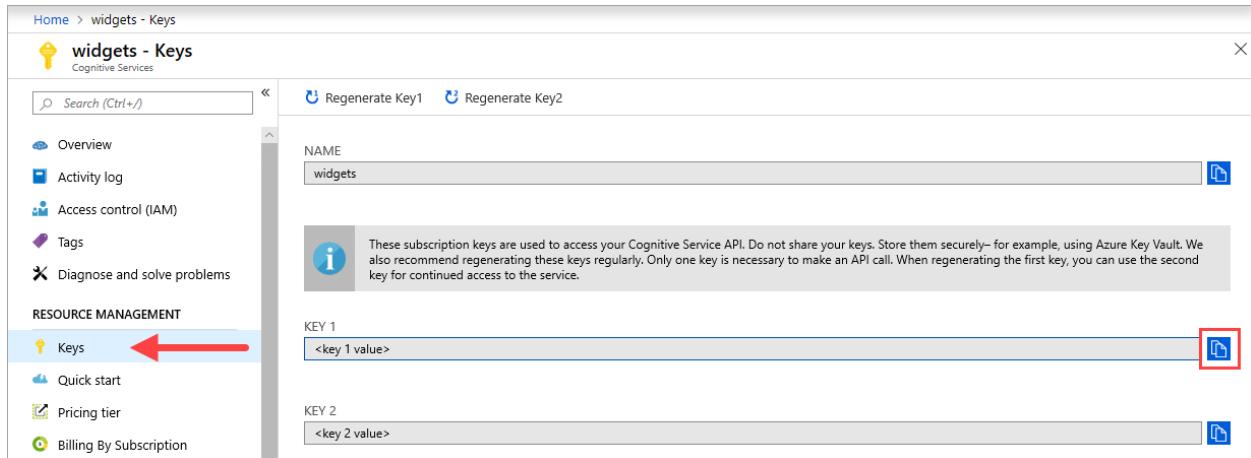
Endpoint URI

The `{ENDPOINT_URI}` value is available on the Azure portal **Overview** page of the corresponding Cognitive Services resource. Go to the **Overview** page, hover over the endpoint, and a **Copy to clipboard** icon appears. Copy and use the endpoint where needed.



Keys

The `{API_KEY}` value is used to start the container and is available on the Azure portal's **Keys** page of the corresponding Cognitive Services resource. Go to the **Keys** page, and select the **Copy to clipboard**  icon.



The screenshot shows the Azure portal interface for a Cognitive Services resource named "widgets". The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, RESOURCE MANAGEMENT (with Keys highlighted by a red arrow), Quick start, Pricing tier, and Billing By Subscription. The main content area displays the "Keys" page with a "NAME" input field set to "widgets". A note states: "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." Below this are two fields: "KEY 1" containing "<key 1 value>" and "KEY 2" containing "<key 2 value>". Each field has a "Copy to clipboard" icon (a blue square with a white document icon) which is also highlighted with a red box.

ⓘ Important

These subscription keys are used to access your Cognitive Services API. Don't share your keys. Store them securely. For example, use Azure Key Vault. We also recommend that you regenerate these keys regularly. Only one key is necessary to make an API call. When you regenerate the first key, you can use the second key for continued access to the service.

Host computer requirements and recommendations

The host is an 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](#).

Container requirements and recommendations

The following table describes the minimum and recommended allocation of resources for each Speech container:

Container	Minimum	Recommended	Speech Model
Speech-to-text	4 core, 4-GB memory	8 core, 8-GB memory	+4 to 8 GB memory
Custom speech-to-text	4 core, 4-GB memory	8 core, 8-GB memory	+4 to 8 GB memory
Speech language identification	1 core, 1-GB memory	1 core, 1-GB memory	n/a
Neural text-to-speech	6 core, 12-GB memory	8 core, 16-GB memory	n/a

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 allocations are based on Docker limits, *not* the host machine resources. For example, speech-to-text containers memory map portions of a large language model. We recommend that the entire file should fit in memory. You need to add an additional 4 to 8 GB to load the speech models (see above table). Also, the first run of either container might take longer because models are being paged into memory.

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:

Console

```
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.

Request approval to run the container

Fill out and submit the [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 submit the form, the Azure Cognitive Services team reviews it and emails you with a decision within 10 business days.

ⓘ Important

- On the form, you must use an email address associated with an Azure subscription ID.
- The Azure resource you use to run the container must have been created with the approved Azure subscription ID.
- Check your email (both inbox and junk folders) for updates on the status of your application from Microsoft.

After you're approved, you'll be able to run the container after you download it from the Microsoft Container Registry (MCR), described later in the article.

You won't be able to run the container if your Azure subscription hasn't been approved.

Speech container images

Speech-to-text

The Speech-to-text container image can be found on the [mcr.microsoft.com](#) container registry syndicate. It resides within the `azure-cognitive-services/speechservices/` repository and is named `speech-to-text`. The fully qualified container image name is, `mcr.microsoft.com/azure-cognitive-services/speechservices/speech-to-text`. You can find a full list of [tags on the MCR](#).

Container	Repository
Speech-to-text	<code>mcr.microsoft.com/azure-cognitive-services/speechservices/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>
```

Get the container image with docker pull

Speech-to-text

Docker pull for the speech-to-text container

Use the [docker pull](#) command to download a container image from Microsoft Container Registry:

```
Docker  
  
docker pull mcr.microsoft.com/azure-cognitive-  
services/speechservices/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:

```
2.6.0-amd64-en-us
```

For all the supported locales of the speech-to-text container, see [Speech-to-text image tags](#).

Use the container

After 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. For more information on how to get the `{Endpoint_URI}` and `{API_Key}` values, see [Gather required parameters](#). More [examples](#) of the `docker run` command are also available.

ⓘ Note

For general container requirements, see [Container requirements and recommendations](#).

Speech-to-text

To run the standard speech-to-text container, execute the following `docker run` command:

Bash

```
docker run --rm -it -p 5000:5000 --memory 8g --cpus 4 \
mcr.microsoft.com/azure-cognitive-services/speechservices/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 8 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.

① Note

Containers support compressed audio input to the Speech SDK by using GStreamer. To install GStreamer in a container, follow Linux instructions for GStreamer in [Use codec compressed audio input with the Speech SDK](#).

Diarization on the speech-to-text output

Diarization is enabled by default. To get diarization in your response, use

```
diarize_speech_config.set_service_property.
```

1. Set the phrase output format to `Detailed`.
2. Set the mode of diarization. The supported modes are `Identity` and `Anonymous`.

Python

```
diarize_speech_config.set_service_property(
    name='speechcontext-PhraseOutput.Format',
    value='Detailed',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)

diarize_speech_config.set_service_property(
    name='speechcontext-phraseDetection.speakerDiarization.mode',
    value='Identity',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)
```

① Note

"Identity" mode returns "SpeakerId": "Customer" or "SpeakerId": "Agent". "Anonymous" mode returns "SpeakerId": "Speaker 1" or "SpeakerId": "Speaker 2".

Analyze sentiment on the speech-to-text output

Starting in v2.6.0 of the speech-to-text container, you should use Language service 3.0 API endpoint instead of the preview one. For example:

- <https://eastus.api.cognitive.microsoft.com/text/analytics/v3.0/sentiment>
- <https://localhost:5000/text/analytics/v3.0/sentiment>

➊ Note

The Language service v3.0 API isn't backward compatible with v3.0-preview.1. To get the latest sentiment feature support, use v2.6.0 of the speech-to-text container image and Language service v3.0.

Starting in v2.2.0 of the speech-to-text container, you can call the [sentiment analysis v3 API](#) on the output. To call sentiment analysis, you'll need a Language service API resource endpoint. For example:

- <https://eastus.api.cognitive.microsoft.com/text/analytics/v3.0-preview.1/sentiment>
- <https://localhost:5000/text/analytics/v3.0-preview.1/sentiment>

If you're accessing a Language service endpoint in the cloud, you'll need a key. If you're running Language service features locally, you might not need to provide this.

The key and endpoint are passed to the Speech container as arguments, as in the following example:

Bash

```
docker run -it --rm -p 5000:5000 \
mcr.microsoft.com/azure-cognitive-services/speechservices/speech-to-
text:latest \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY} \
CloudAI:SentimentAnalysisSettings:TextAnalyticsHost=
{TEXT_ANALYTICS_HOST} \
```

```
CloudAI:SentimentAnalysisSettings:SentimentAnalysisApiKey=
{SENTIMENT_APIKEY}
```

This command:

- Performs the same steps as the preceding command.
- Stores a Language service API endpoint and key, for sending sentiment analysis requests.

Phraselist v2 on the speech-to-text output

Starting in v2.6.0 of the speech-to-text container, you can get the output with your own phrases, either the whole sentence or phrases in the middle. For example, *the tall man* in the following sentence:

- "This is a sentence **the tall man** this is another sentence."

To configure a phrase list, you need to add your own phrases when you make the call. For example:

Python

```
phrase="the tall man"
recognizer = speechsdk.SpeechRecognizer(
    speech_config=dict_speech_config,
    audio_config=audio_config)
phrase_list_grammar =
speechsdk.PhraseListGrammar.from_recognizer(recognizer)
phrase_list_grammar.addPhrase(phrase)

dict_speech_config.set_service_property(
    name='setflight',
    value='xonlineinterp',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)
```

If you have multiple phrases to add, call `.addPhrase()` for each phrase to add it to the phrase list.

ⓘ 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](#).

Run the container in disconnected environments

You must request access to use containers disconnected from the internet. For more information, see [Request access to use containers in disconnected environments](#).

For Speech Service container configuration, see [Disconnected containers](#).

Query the container's prediction endpoint

Note

Use a unique port number if you're running multiple containers.

Containers	SDK Host URL	Protocol
Standard speech-to-text and custom speech-to-text	<code>ws://localhost:5000</code>	WS
Neural Text-to-speech, Speech language identification	<code>http://localhost:5000</code>	HTTP

For more information on using WSS and HTTPS protocols, see [Container security](#).

Speech-to-text (standard and custom)

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 you use the Speech SDK with containers, you don't need to provide the Azure Speech resource **subscription key or an authentication bearer token**.

See the following examples.

C#

Change from using this Azure-cloud initialization call:

C#

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

To use this call with the container [host](#):

C#

```
var config = SpeechConfig.FromHost(
    new Uri("ws://localhost:5000"));
```

Analyze sentiment

If you provided your Language service API credentials [to the container](#), you can use the Speech SDK to send speech recognition requests with sentiment analysis. You can configure the API responses to use either a *simple* or *detailed* format.

ⓘ Note

v1.13 of the Speech Service Python SDK has an identified issue with sentiment analysis. Use v1.12.x or earlier if you're using sentiment analysis in the Speech Service Python SDK.

Simple format

To configure the Speech client to use a simple format, add `"Sentiment"` as a value for `Simple.Extensions`. If you want to choose a specific Language service model version, replace `'latest'` in the `speechcontext-phraseDetection.sentimentAnalysis.modelversion` property configuration.

Python

```
speech_config.set_service_property(
    name='speechcontext-PhraseOutput.Simple.Extensions',
    value='["Sentiment"]',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)
speech_config.set_service_property(
    name='speechcontext-phraseDetection.sentimentAnalysis.modelversion',
    value='latest',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)
```

`Simple.Extensions` returns the sentiment result in the root layer of the response.

JSON

```
{  
    "DisplayText": "What's the weather like?",  
    "Duration": 13000000,  
    "Id": "6098574b79434bd4849fee7e0a50f22e",  
    "Offset": 4700000,  
    "RecognitionStatus": "Success",  
    "Sentiment": {  
        "Negative": 0.03,  
        "Neutral": 0.79,  
        "Positive": 0.18  
    }  
}
```

If you want to completely disable sentiment analysis, add a `false` value to `sentimentanalysis.enabled`.

Python

```
speech_config.set_service_property(  
    name='speechcontext-phraseDetection.sentimentanalysis.enabled',  
    value='false',  
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter  
)
```

Neural Text-to-Speech

The container provides [REST-based endpoint APIs](#). Many [sample source code projects](#) for platform, framework, and language variations are available.

With the neural Text-to-Speech containers, 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 `AriaNeural` voice. The `{VOICE_NAME}` argument would then be `en-US-AriaNeural`. See the following example SSML:

XML

```
<speak version="1.0" xml:lang="en-US">  
    <voice name="en-US-AriaNeural">  
        This text will get converted into synthesized speech.  
    </voice>  
</speak>
```

Run multiple containers on the same host

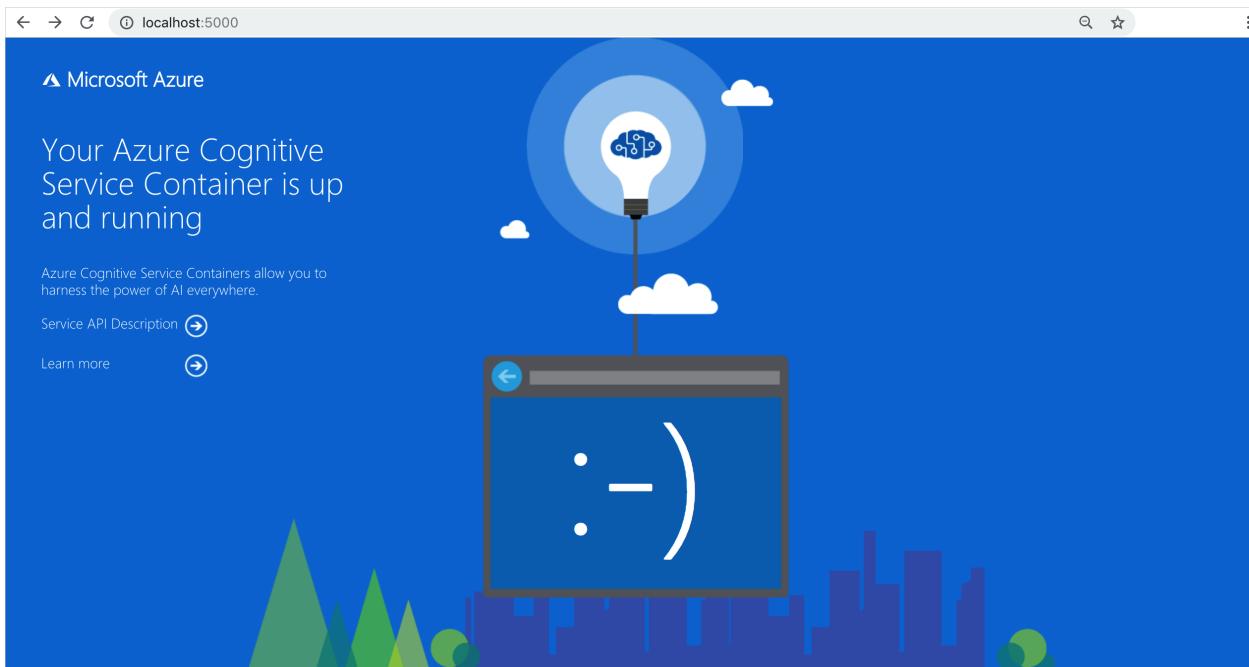
If you intend to run multiple containers with exposed ports, make sure to run each container with a different exposed port. For example, run the first container on port 5000 and the second container on port 5001.

You can have this container and a different 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 that follow to validate the container is running. The example request URLs listed here are `http://localhost:5000`, but your specific container might vary. Make sure 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/ready</code>	Requested with GET, this URL provides a verification that the container is ready to accept a query against the model. This request can be used for Kubernetes liveness and readiness probes .
<code>http://localhost:5000/status</code>	Also requested with GET, this URL verifies if the api-key used to start the container is valid 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 you start or run the container, you might experience issues. Use an output `mount` and enable logging. Doing so allows the container to generate log files that are helpful when you troubleshoot issues.

Tip

For more troubleshooting information and guidance, see [Cognitive Services containers frequently asked questions \(FAQ\)](#).

If you're having trouble running a Cognitive Services container, you can try using the Microsoft diagnostics container. Use this container to diagnose common errors in your deployment environment that might prevent Cognitive Services containers from functioning as expected.

To get the container, use the following `docker pull` command:

Bash

```
docker pull mcr.microsoft.com/azure-cognitive-services/diagnostic
```

Then run the container. Replace `{ENDPOINT_URI}` with your endpoint, and replace `{API_KEY}` with your key to your resource:

Bash

```
docker run --rm mcr.microsoft.com/azure-cognitive-services/diagnostic \
eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

The container will test for network connectivity to the billing endpoint.

Billing

The Speech containers send billing information to Azure by 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` parameter.

Azure Cognitive Services containers aren't licensed to run without being connected to the metering or billing endpoint. 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 serving requests. See the [Cognitive Services container FAQ](#) for an example of the information sent to Microsoft for billing.

Billing arguments

The [docker run ↗](#) command will start the container when all three of the following options are provided with valid values:

Option	Description
--------	-------------

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 Billing.
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 accept.

For more information about these options, see [Configure containers](#).

Summary

In this article, you learned concepts and workflow for how to download, install, and run Speech containers. In summary:

- Speech provides four Linux containers for Docker that have various capabilities:
 - Speech-to-text
 - Custom speech-to-text
 - Neural text-to-speech
 - Speech language identification
- Container images are downloaded from the container registry in Azure.
- Container images run in Docker.
- Whether you use the REST API (text-to-speech only) or the SDK (speech-to-text or text-to-speech), you specify the host URI of the container.
- You're required to provide billing information when you instantiate a container.

Important

Cognitive Services containers aren't 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 don't send customer data (for example, the image or text that's 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.
-

Additional resources

Documentation

[Configure Speech containers - Azure Cognitive Services](#)

Speech service provides each container with a common configuration framework, so that you can easily configure and manage storage, logging and telemetry, and security settings for your containers.

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Training and testing datasets - Speech service - Azure Cognitive Services](#)

Learn about types of training and testing data for a Custom Speech project, along with how to use and manage that data.

[Speech service quotas and limits - Azure Cognitive Services](#)

Quick reference, detailed description, and best practices on the quotas and limits for the Speech service in Azure Cognitive Services.

[Speaker Recognition quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you use speaker recognition to confirm who is speaking. Learn about common design patterns for working with speaker verification and identification.

[azure.cognitiveservices.speech.SpeechRecognizer class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, language, source_language_config or auto_detect_source_language_config.

[azure.cognitiveservices.speech.Recognizer class](#)

Base class for different recognizers

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[Show 5 more](#)

Training

Learning paths and modules

[Process and Translate Speech with Azure Cognitive Speech Services - Training](#)

Process and Translate Speech with Azure Cognitive Speech Services

Configure Speech service containers

Article • 02/01/2023 • 9 minutes to read

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 supported speech containers are **speech-to-text**, **Custom speech-to-text**, **speech language identification** and **Neural 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	ApiKey	Tracks billing information.
No	ApplicationInsights	Enables adding Azure Application Insights telemetry support to your container.
Yes	Billing	Specifies the endpoint URI of the service resource on Azure.
Yes	Eula	Indicates that you've accepted the license for the container.
No	Fluentd	Writes log and, optionally, metric data to a Fluentd server.
No	HTTP Proxy	Configures an HTTP proxy for making outbound requests.
No	Logging	Provides ASP.NET Core logging support for your container.
No	Mounts	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 Application Insights for ASP.NET Core.</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	Billing	String	Billing endpoint URI. For more information on obtaining the billing URI, see gather required parameters . For more information and a complete list of regional endpoints, see Custom subdomain names for Cognitive Services .

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	Eula	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
Host	String	The IP address or DNS host name of the Fluentd server.

Name	Data type	Description
Port	Integer	The port of the Fluentd server. The default value is 24224.
HeartbeatMs	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).
SendBufferSize	Integer	The network buffer space, in bytes, allocated for send operations. The default value is 32768 bytes (32 kilobytes).
TlsConnectionEstablishmentTimeoutMs	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.
UseTLS	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
HTTP_PROXY	string	The proxy to use, for example, <code><proxy-url></code>
HTTP_PROXY_CREDS	string	Any credentials needed to authenticate against the proxy, for example, <code>username:password</code> . This value must be in lower-case .
<code><proxy-user></code>	string	The user for the proxy.
<code><proxy-password></code>	string	The password associated with <code><proxy-user></code> for the proxy.

Bash

```
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:

Bash

```
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 \
Mounts:Output=/output
```

This container command shows debugging information, prefixed with `dbug`, while the container is running:

Bash

```
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
<code>Format</code>	String	<p>The output format for log files.</p> <p>Note: 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.</p>
<code>MaxFileSize</code>	Integer	<p>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.</p>

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 volume mounts .
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=/output`

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

Bash

```
-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** containers. The **Speech-to-text**, **Neural Text-to-speech** and **Speech language**

identification 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 Keys</code> page.	<code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code>
<code>{ENDPOINT_URI}</code>	The billing endpoint value is available on the Azure <code>Speech</code> Overview page.	See gather required parameters 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.

Basic example for Speech-to-text

Docker

```
docker run --rm -it -p 5000:5000 --memory 8g --cpus 4 \
mcr.microsoft.com/azure-cognitive-services/speechservices/speech-to-text \
\
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

Logging example for Speech-to-text

Docker

```
docker run --rm -it -p 5000:5000 --memory 8g --cpus 4 \
mcr.microsoft.com/azure-cognitive-services/speechservices/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](#)

Additional resources

Documentation

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Upload training and testing datasets for Custom Speech - Speech service - Azure Cognitive Services](#)

Learn about how to upload data to test or train a Custom Speech model.

[azure.cognitiveservices.speech.SpeechRecognizer class](#)

A speech recognizer. If you need to specify source language information, please only specify one of these three parameters, language, source_language_config or auto_detect_source_language_config.

[Training and testing datasets - Speech service - Azure Cognitive Services](#)

Learn about types of training and testing data for a Custom Speech project, along with how to use and manage that data.

[CI/CD for Custom Speech - Speech service - Azure Cognitive Services](#)

Apply DevOps with Custom Speech and CI/CD workflows. Implement an existing DevOps solution for your own project.

[Install and run Docker containers for the Speech service APIs - Azure Cognitive Services](#)

Use the Docker containers for the Speech service to perform speech recognition, transcription, generation, and more on-premises.

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Model lifecycle of Custom Speech - Speech service - Azure Cognitive Services](#)

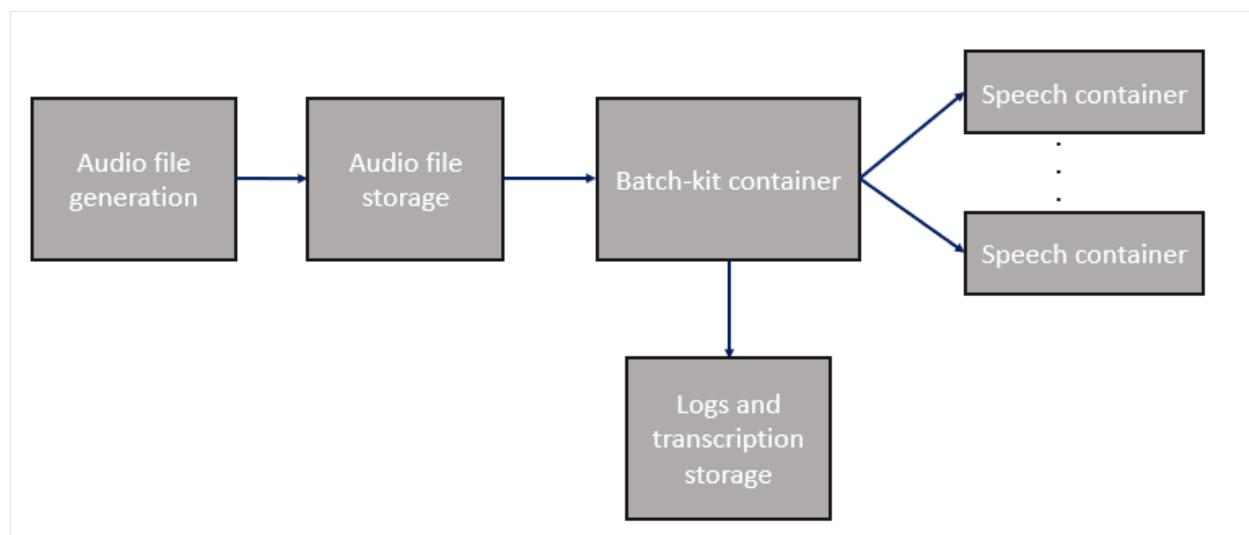
Custom Speech provides base models for training and lets you create custom models from your data. This article describes the timelines for models and for endpoints that use these models.

[Show 5 more](#)

Batch processing kit for Speech containers

Article • 01/19/2023 • 6 minutes to read

Use the batch processing kit to complement and scale out workloads on Speech containers. Available as a container, this open-source utility helps facilitate batch transcription for large numbers of audio files, across any number of on-premises and cloud-based speech container endpoints.



The batch kit container is available for free on [GitHub](#) and [Docker hub](#). You are only [billed](#) for the Speech containers you use.

Feature	Description
Batch audio file distribution	Automatically dispatch large numbers of files to on-premises or cloud-based Speech container endpoints. Files can be on any POSIX-compliant volume, including network filesystems.
Speech SDK integration	Pass common flags to the Speech SDK, including: n-best hypotheses, diarization, language, profanity masking.
Run modes	Run the batch client once, continuously in the background, or create HTTP endpoints for audio files.
Fault tolerance	Automatically retry and continue transcription without losing progress, and differentiate between which errors can, and can't be retried on.
Endpoint availability detection	If an endpoint becomes unavailable, the batch client will continue transcribing, using other container endpoints. After becoming available again, the client will automatically begin using the endpoint.

Feature	Description
Endpoint hot-swapping	Add, remove, or modify Speech container endpoints during runtime without interrupting the batch progress. Updates are immediate.
Real-time logging	Real-time logging of attempted requests, timestamps, and failure reasons, with Speech SDK log files for each audio file.

Get the container image with `docker pull`

Use the [docker pull](#) command to download the latest batch kit container.

ⓘ Note

The following example pulls a public container image from Docker Hub. We recommend that you authenticate with your Docker Hub account (`docker login`) first instead of making an anonymous pull request. To improve reliability when using public content, import and manage the image in a private Azure container registry. [Learn more about working with public images](#).

Bash

```
docker pull docker.io/batchkit/speech-batch-kit:latest
```

Endpoint configuration

The batch client takes a YAML configuration file that specifies the on-premises container endpoints. The following example can be written to `/mnt/my_nfs/config.yaml`, which is used in the examples below.

YAML

```
MyContainer1:
  concurrency: 5
  host: 192.168.0.100
  port: 5000
  rtf: 3
MyContainer2:
  concurrency: 5
  host: BatchVM0.corp.redmond.microsoft.com
  port: 5000
  rtf: 2
MyContainer3:
```

```
concurrency: 10
host: localhost
port: 6001
rtf: 4
```

This yaml example specifies three speech containers on three hosts. The first host is specified by a IPv4 address, the second is running on the same VM as the batch-client, and the third container is specified by the DNS hostname of another VM. The `concurrency` value specifies the maximum concurrent file transcriptions that can run on the same container. The `rtf` (Real-Time Factor) value is optional and can be used to tune performance.

The batch client can dynamically detect if an endpoint becomes unavailable (for example, due to a container restart or networking issue), and when it becomes available again. Transcription requests will not be sent to containers that are unavailable, and the client will continue using other available containers. You can add, remove, or edit endpoints at any time without interrupting the progress of your batch.

Run the batch processing container

ⓘ Note

- This example uses the same directory (`/my_nfs`) for the configuration file and the inputs, outputs, and logs directories. You can use hosted or NFS-mounted directories for these folders.
- Running the client with `-h` will list the available command-line parameters, and their default values.
- The batch processing container is only supported on Linux.

Use the Docker `run` command to start the container. This will start an interactive shell inside the container.

Docker

```
docker run --network host --rm -ti -v /mnt/my_nfs:/my_nfs --entrypoint
/bin/bash /mnt/my_nfs:/my_nfs docker.io/batchkit/speech-batch-kit:latest
```

To run the batch client:

Docker

```
run-batch-client -config /my_nfs/config.yaml -input_folder  
/my_nfs/audio_files -output_folder /my_nfs/transcriptions -log_folder  
/my_nfs/logs -file_log_level DEBUG -nbest 1 -m ONESHOT -diarization None -  
language en-US -strict_config
```

To run the batch client and container in a single command:

Docker

```
docker run --network host --rm -ti -v /mnt/my_nfs:/my_nfs  
docker.io/batchkit/speech-batch-kit:latest -config /my_nfs/config.yaml -  
input_folder /my_nfs/audio_files -output_folder /my_nfs/transcriptions -  
log_folder /my_nfs/logs
```

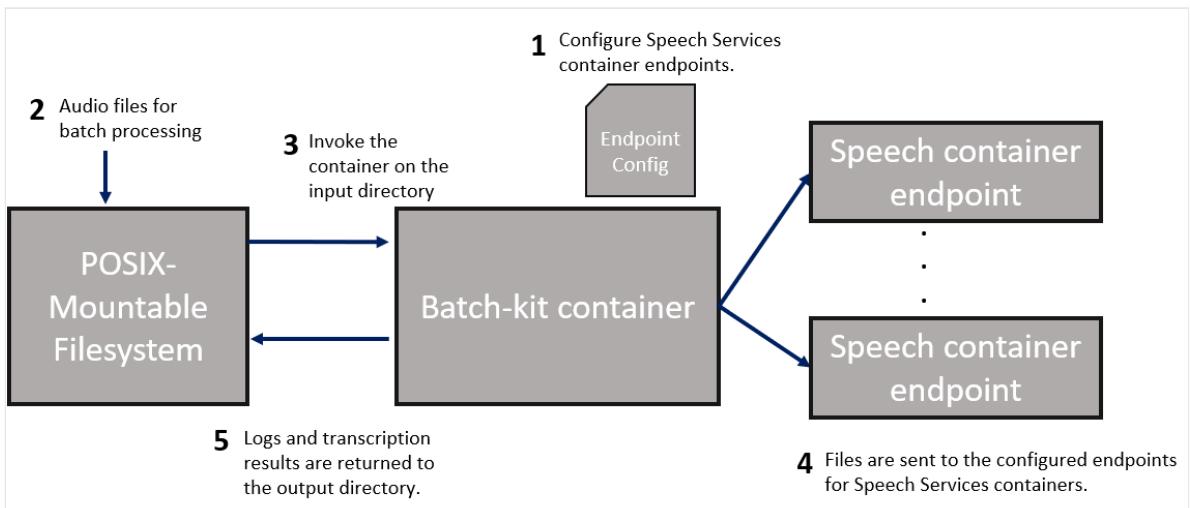
The client will start running. If an audio file has already been transcribed in a previous run, the client will automatically skip the file. Files are sent with an automatic retry if transient errors occur, and you can differentiate between which errors you want the client to retry on. On a transcription error, the client will continue transcription, and can retry without losing progress.

Run modes

The batch processing kit offers three modes, using the `--run-mode` parameter.

Oneshot

`ONESHOT` mode transcribes a single batch of audio files (from an input directory and optional file list) to an output folder.



1. Define the Speech container endpoints that the batch client will use in the `config.yaml` file.
2. Place audio files for transcription in an input directory.
3. Invoke the container on the directory, which will begin processing the files. If the audio file has already been transcribed in a previous run with the same output directory (same file name and checksum), the client will skip the file.
4. The files are dispatched to the container endpoints from step 1.
5. Logs and the Speech container output are returned to the specified output directory.

Logging

Note

The batch client may overwrite the `run.log` file periodically if it gets too large.

The client creates a `run.log` file in the directory specified by the `-log_folder` argument in the docker `run` command. Logs are captured at the DEBUG level by default. The same logs are sent to the `stdout/stderr`, and filtered depending on the `-file_log_level` or `console_log_level` arguments. This log is only necessary for debugging, or if you need to send a trace for support. The logging folder also contains the Speech SDK logs for each audio file.

The output directory specified by `-output_folder` will contain a `run_summary.json` file, which is periodically rewritten every 30 seconds or whenever new transcriptions are finished. You can use this file to check on progress as the batch proceeds. It will also contain the final run statistics and final status of every file when the batch is completed. The batch is completed when the process has a clean exit.

Next steps

- [How to install and run containers](#)

Additional resources

Documentation

[Configure Speech containers - Azure Cognitive Services](#)

Speech service provides each container with a common configuration framework, so that you can easily configure and manage storage, logging and telemetry, and security settings for your containers.

[How to get Speech-to-text Session ID and Transcription ID - Azure Cognitive Services](#)

Learn how to get Speech service Speech-to-text Session ID and Transcription ID

[Test recognition quality of a Custom Speech model - Speech service - Azure Cognitive Services](#)

Custom Speech lets you qualitatively inspect the recognition quality of a model. You can play back uploaded audio and determine if the provided recognition result is correct.

[Create captions with speech to text quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you convert speech to text as captions.

[Model lifecycle of Custom Speech - Speech service - Azure Cognitive Services](#)

Custom Speech provides base models for training and lets you create custom models from your data. This article describes the timelines for models and for endpoints that use these models.

[Speech service containers frequently asked questions \(FAQ\) - Azure Cognitive Services](#)

Install and run containers for speech-to-text and text-to-speech.

[Create a Custom Speech project - Speech service - Azure Cognitive Services](#)

Learn about how to create a project for Custom Speech.

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Show 5 more](#)

Use Speech service containers with Kubernetes and Helm

Article • 12/01/2022 • 12 minutes to read

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 free account 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 Kubernetes CLI 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	Install the Helm CLI , which is used to install a helm chart (container package definition).
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 Speech Overview and Keys pages and are required to start the container. {API_KEY}: resource key {ENDPOINT_URI}: endpoint URI example is: https://eastus.api.cognitive.microsoft.com/sts/v1.0

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
Speech-to-Text	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
Text-to-Speech	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.

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.

Console

```
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.

YAML

```
# 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/speechservices/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/speechservices/speech-to-text
    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](#).

Console

```
helm install onprem-speech microsoft/cognitive-services-speech-onpremise \
--version 0.1.1 \
--values <config-values.yaml>
```

Here is an example output you might expect to see from a successful install execution:

Console

```
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:

Console

```
kubectl get all
```

You should expect to see something similar to the following output:

Console

NAME	READY	STATUS	RESTARTS	AGE
pod/speech-to-text-7664f5f465-87w2d	1/1	Running	0	34m
pod/speech-to-text-7664f5f465-k1br8	1/1	Running	0	34m
pod/text-to-speech-56f8fb685b-4jtzh	1/1	Running	0	34m
pod/text-to-speech-56f8fb685b-frwxr	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 2 10 2 34m	Deployment/speech-to-text	1%/50%
horizontalpodautoscaler.autoscaling/text-to-speech-autoscaler 2 10 2 34m	Deployment/text-to-speech	0%/50%

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.

Console

```
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:

Console

```
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 speech-to-text service is enabled.	<code>true</code>
<code>speechToText.verification.enabled</code>	Whether the <code>helm test</code> capability for speech-to-text service is enabled.	<code>true</code>
<code>speechToText.verification.image.registry</code>	The docker image repository that <code>helm test</code> uses to test speech-to-text 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 speech-to-text 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 speech-to-text 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 text-to-speech service is enabled.	<code>true</code>
<code>textToSpeech.verification.enabled</code>	Whether the <code>helm test</code> capability for speech-to-text service is enabled.	<code>true</code>

Parameter	Description	Default
<code>textToSpeech.verification.image.registry</code>	The docker image repository that <code>helm test</code> uses to test speech-to-text service. Helm creates separate pod inside the cluster for testing and pulls the <code>test-use</code> image from this registry.	<code>docker.io</code>
<code>textToSpeech.verification.image.repository</code>	The docker image repository that <code>helm test</code> uses to test speech-to-text service. Helm test pod uses this repository to pull <code>test-use</code> image.	<code>antsu/on-prem-client</code>
<code>textToSpeech.verification.image.tag</code>	The docker image tag used with <code>helm test</code> for speech-to-text service. Helm test pod uses this tag to pull <code>test-use</code> image.	<code>latest</code>
<code>textToSpeech.verification.image.pullByHash</code>	Whether the <code>test-use</code> 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 <code>test-use</code> docker image. The helm test pod passes these arguments to container when running <code>helm test</code> .	<code>./text-to-speech-client "--input='What's the weather like'" "--host=\$(TEXT_TO_SPEECH_HOST)" "--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 speech-to-text service is enabled.	<code>false</code>
<code>numberOfConcurrentRequest</code>	The number of concurrent requests for the speech-to-text 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 speech-to-text docker image registry.	<code>containerpreview.azurecr.io</code>
<code>image.repository</code>	The speech-to-text docker image repository.	<code>microsoft/cognitive-services-speech-to-text</code>
<code>image.tag</code>	The speech-to-text docker image tag.	<code>latest</code>
<code>image.pullSecrets</code>	The image secrets for pulling the speech-to-text docker image.	
<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 speech-to-text docker image hash. Only used when <code>image.pullByHash: true</code> .	

Parameter	Description	Default
<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 speech-to-text service. See the Kubernetes service types instructions for more details and verify cloud provider support.	<code>LoadBalancer</code>
<code>service.port</code>	The port of the speech-to-text service.	<code>80</code>
<code>service.annotations</code>	The speech-to-text 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 Horizontal Pod Autoscaler 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 Pod Disruption Budget is enabled. If <code>true</code> , the <code>speech-to-text-poddisruptionbudget</code> will be deployed in the Kubernetes cluster.	<code>true</code>

Sentiment analysis (sub-chart: charts/speechToText)

Starting with v2.2.0 of the speech-to-text container and v0.2.0 of the Helm chart, the following parameters are used for sentiment analysis using the Language service API.

Parameter	Description	Values	Default
<code>textanalytics.enabled</code>	Whether the text-analytics service is enabled	true/false	<code>false</code>
<code>textanalytics.image.registry</code>	The text-analytics docker image registry	valid docker image registry	
<code>textanalytics.image.repository</code>	The text-analytics docker image repository	valid docker image repository	
<code>textanalytics.image.tag</code>	The text-analytics docker image tag	valid docker image tag	
<code>textanalytics.image.pullSecrets</code>	The image secrets for pulling text-analytics docker image	valid secrets name	
<code>textanalytics.image.pullByHash</code>	Specifies if you are pulling docker image by hash. If <code>yes</code> , <code>image.hash</code> is required to have as well. If <code>no</code> , set it as 'false'. Default is <code>false</code> .	true/false	<code>false</code>
<code>textanalytics.image.hash</code>	The text-analytics docker image hash. Only use it with <code>image.pullByHash:true</code> .	valid docker image hash	

Parameter	Description	Values	Default
<code>textanalytics.image.args.eula</code>	One of the required arguments by <code>text-analytics</code> container, which indicates you've accepted the license. The value of this option must be: <code>accept</code> .	accept, if you want to use the container	
<code>textanalytics.image.args.billing</code>	One of the required arguments by <code>text-analytics</code> container, which specifies the billing endpoint URL. The billing endpoint URI value is available on the Azure portal's Speech Overview page.	valid billing endpoint URI	
<code>textanalytics.image.args.apikey</code>	One of the required arguments by <code>text-analytics</code> container, which is used to track billing information.	valid apikey	
<code>textanalytics.cpuRequest</code>	The requested CPU for <code>text-analytics</code> container	int	<code>3000m</code>
<code>textanalytics.cpuLimit</code>	The limited CPU for <code>text-analytics</code> container		<code>8000m</code>
<code>textanalytics.memoryRequest</code>	The requested memory for <code>text-analytics</code> container		<code>3Gi</code>
<code>textanalytics.memoryLimit</code>	The limited memory for <code>text-analytics</code> container		<code>8Gi</code>
<code>textanalytics.service.sentimentURISuffix</code>	The sentiment analysis URI suffix, the whole URI is in format "http://<service>:<port>/<sentimentURISuffix>".		<code>text/analytics/v3.0-preview/sentiment</code>
<code>textanalytics.service.type</code>	The type of <code>text-analytics</code> service in Kubernetes. See Kubernetes service types	valid Kubernetes service type	<code>LoadBalancer</code>
<code>textanalytics.service.port</code>	The port of the <code>text-analytics</code> service	int	<code>50085</code>
<code>textanalytics.service.annotations</code>	The annotations users can add to <code>text-analytics</code> service metadata. For instance: <code>annotations:</code> <code>some/annotation1: value1</code> <code>some/annotation2: value2</code>	annotations, one per each line	
<code>textanalytics.servce.autoScaler.enabled</code>	Whether Horizontal Pod Autoscaler is enabled. If enabled, <code>text-analytics-autoscaler</code> will be deployed in the Kubernetes cluster	true/false	<code>true</code>
<code>textanalytics.service.podDisruption.enabled</code>	Whether Pod Disruption Budget is enabled. If enabled, <code>text-analytics-poddisruptionbudget</code> will be deployed in the Kubernetes cluster	true/false	<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 <code>text-to-speech</code> service is enabled.	<code>false</code>
<code>numberOfConcurrentRequest</code>	The number of concurrent requests for the <code>text-to-speech</code> service. This chart automatically calculates CPU and memory resources, based on this value.	<code>2</code>

Parameter	Description	Default
<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.	<code>false</code>
<code>image.registry</code>	The text-to-speech docker image registry.	<code>containerpreview.azurecr.io</code>
<code>image.repository</code>	The text-to-speech docker image repository.	<code>microsoft/cognitive-services-text-to-speech</code>
<code>image.tag</code>	The text-to-speech docker image tag.	<code>latest</code>
<code>image.pullSecrets</code>	The image secrets for pulling the text-to-speech docker image.	
<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 text-to-speech 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 text-to-speech service. See the Kubernetes service types instructions for more details and verify cloud provider support.	<code>LoadBalancer</code>
<code>service.port</code>	The port of the text-to-speech service.	<code>80</code>
<code>service.annotations</code>	The text-to-speech 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 Horizontal Pod Autoscaler 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 Pod Disruption Budget 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 and run container on Azure Container Instance

Article • 02/11/2022 • 7 minutes to read

With the following steps, scale Azure Cognitive Services applications in the cloud easily with Azure [Container Instances](#). Containerization helps you focus on building your applications instead of managing the infrastructure. For more information on using containers, see [features and benefits](#).

Prerequisites

The recipe works with any Cognitive Services container. The Cognitive Service resource must be created before using the recipe. Each Cognitive Service that supports containers has a "How to install" article for installing and configuring the service for a container. Some services require a file or set of files as input for the container, it is important that you understand and have used the container successfully before using this solution.

- An Azure resource for the Azure Cognitive Service you're using.
- Cognitive Service **endpoint URL** - review your specific service's "How to install" for the container, to find where the endpoint URL is from within the Azure portal, and what a correct example of the URL looks like. The exact format can change from service to service.
- Cognitive Service **key** - the keys are on the **Keys** page for the Azure resource. You only need one of the two keys. The key is a string of 32 alpha-numeric characters.
- A single Cognitive Services Container on your local host (your computer). Make sure you can:
 - Pull down the image with a `docker pull` command.
 - Run the local container successfully with all required configuration settings with a `docker run` command.
 - Call the container's endpoint, getting a response of HTTP 2xx and a JSON response back.

All variables in angle brackets, `<>`, need to be replaced with your own values. This replacement includes the angle brackets.

 **Important**

The LUIS container requires a `.gz` model file that is pulled in at runtime. The container must be able to access this model file via a volume mount from the container instance. To upload a model file, follow these steps:

1. [Create an Azure file share](#). Take note of the Azure Storage account name, key, and file share name as you'll need them later.
2. [export your LUIS model \(packaged app\) from the LUIS portal](#).
3. In the Azure portal, navigate to the [Overview](#) page of your storage account resource, and select **File shares**.
4. Select the file share name that you recently created, then select **Upload**. Then upload your packaged app.

Azure portal

Create an Azure Container Instance resource using the Azure portal

1. Go to the [Create](#) page for Container Instances.
2. On the **Basics** tab, enter the following details:

Setting	Value
Subscription	Select your subscription.
Resource group	Select the available resource group or create a new one such as <code>cognitive-services</code> .
Container name	Enter a name such as <code>cognitive-container-instance</code> . The name must be in lower caps.
Location	Select a region for deployment.
Image type	If your container image is stored in a container registry that doesn't require credentials, choose <code>Public</code> . If accessing your container image requires credentials, choose <code>Private</code> . Refer to container repositories and images for details on whether or not the container image is <code>Public</code> or <code>Private</code> ("Public Preview").

Setting	Value
Image name	<p>Enter the Cognitive Services container location. The location is what's used as an argument to the <code>docker pull</code> command. Refer to the container repositories and images for the available image names and their corresponding repository.</p> <p>The image name must be fully qualified specifying three parts. First, the container registry, then the repository, finally the image name: <code><container-registry>/<repository>/<image-name></code>.</p> <p>Here is an example, <code>mcr.microsoft.com/azure-cognitive-services/keyphrase</code> would represent the Key Phrase Extraction image in the Microsoft Container Registry under the Azure Cognitive Services repository. Another example is, <code>containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text</code> which would represent the Speech to Text image in the Microsoft repository of the Container Preview container registry.</p>
OS type	<code>Linux</code>
Size	<p>Change size to the suggested recommendations for your specific Cognitive Service container:</p> <ul style="list-style-type: none"> 2 CPU cores 4 GB

3. On the **Networking** tab, enter the following details:

Setting	Value
Ports	Set the TCP port to <code>5000</code> . Exposes the container on port 5000.

4. On the **Advanced** tab, enter the required **Environment Variables** for the container billing settings of the Azure Container Instance resource:

Key	Value
<code>ApiKey</code>	Copied from the Keys and endpoint page of the resource. It is a 32 alphanumeric-character string with no spaces or dashes, <code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code> .
<code>Billing</code>	Your endpoint URL copied from the Keys and endpoint page of the resource.
<code>Eula</code>	<code>accept</code>

5. Click **Review and Create**

6. After validation passes, click **Create** to finish the creation process

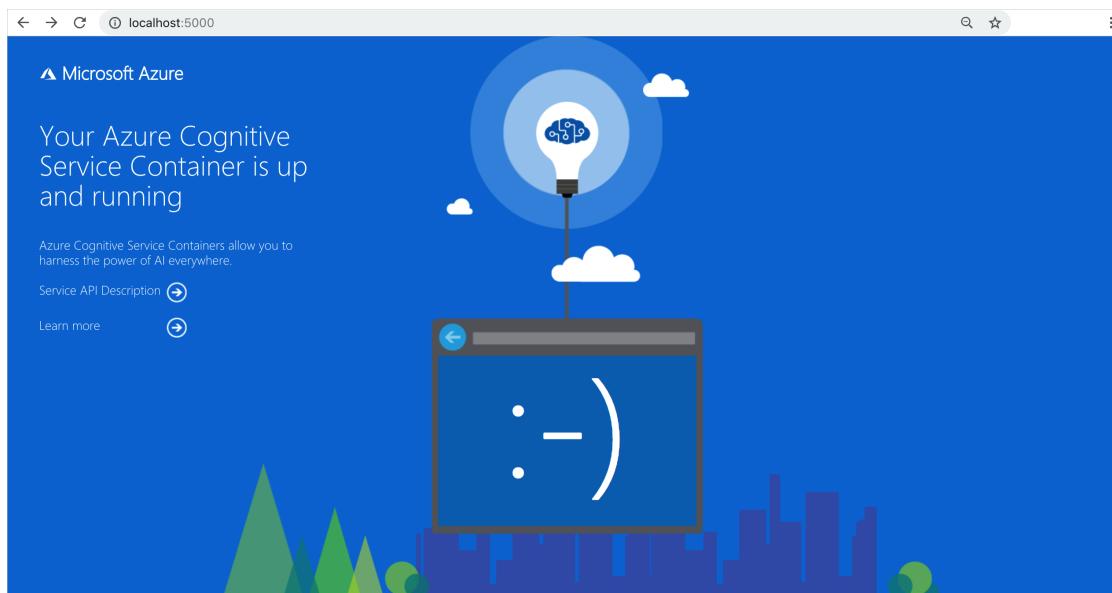
7. When the resource is successfully deployed, it's ready

Use the Container Instance

Azure portal

1. Select the **Overview** and copy the IP address. It will be a numeric IP address such as `55.55.55.55`.

2. Open a new browser tab and use the IP address, for example, `http://<IP-address>:5000` (`http://55.55.55.55:5000`). You will see the container's home page, letting you know the container is running.



3. Select **Service API Description** to view the swagger page for the container.

4. Select any of the **POST** APIs and select **Try it out**. The parameters are displayed including the input. Fill in the parameters.

5. Select **Execute** to send the request to your Container Instance.

You have successfully created and used Cognitive Services containers in Azure Container Instance.

Use Docker containers in disconnected environments

Article • 02/01/2023 • 10 minutes to read

Containers enable you to run Cognitive Services APIs in your own environment, and are great for your specific security and data governance requirements. Disconnected containers enable you to use several of these APIs disconnected from the internet.

Currently, the following containers can be run in this manner:

- [Speech-to-Text](#)
- [Custom Speech-to-Text](#)
- [Neural Text-to-Speech](#)
- [Text Translation \(Standard\)](#)
- [Language Understanding \(LUIS\)](#)
- Azure Cognitive Service for Language
 - [Sentiment Analysis](#)
 - [Key Phrase Extraction](#)
 - [Language Detection](#)
- Computer Vision - Read

Disconnected container usage is also available for the following Applied AI service:

- [Form Recognizer](#)

Before attempting to run a Docker container in an offline environment, make sure you know the steps to successfully download and use the container. For example:

- Host computer requirements and recommendations.
- The Docker `pull` command you'll use to download the container.
- How to validate that a container is running.
- How to send queries to the container's endpoint, once it's running.

Request access to use containers in disconnected environments

Fill out and submit the [request form](#) to request access to the containers disconnected from the internet.

The form requests information about you, your company, and the user scenario for which you'll use the container. After you submit the form, the Azure Cognitive Services

team reviews it and emails you with a decision within 10 business days.

ⓘ Important

- On the form, you must use an email address associated with an Azure subscription ID.
- The Azure resource you use to run the container must have been created with the approved Azure subscription ID.
- Check your email (both inbox and junk folders) for updates on the status of your application from Microsoft.

After you're approved, you'll be able to run the container after you download it from the Microsoft Container Registry (MCR), described later in the article.

You won't be able to run the container if your Azure subscription hasn't been approved.

Access is limited to customers that meet the following requirements:

- Your organization should be identified as strategic customer or partner with Microsoft.
- Disconnected containers are expected to run fully offline, hence your use cases must meet one of below or similar requirements:
 - Environment or device(s) with zero connectivity to internet.
 - Remote location that occasionally has internet access.
 - Organization under strict regulation of not sending any kind of data back to cloud.
- Application completed as instructed - Please pay close attention to guidance provided throughout the application to ensure you provide all the necessary information required for approval.

Purchase a commitment plan to use containers in disconnected environments

Create a new resource

1. Sign into the [Azure portal](#) and select **Create a new resource** for one of the applicable Cognitive Services or Applied AI services listed above.
2. Enter the applicable information to create your resource. Be sure to select **Commitment tier disconnected containers** as your pricing tier.

 **Note**

- You will only see the option to purchase a commitment tier if you have been approved by Microsoft.
- Pricing details are for example only.

Create Form Recognizer ...

Basics Network Identity Tags Review + create

Accelerate your business processes by automating information extraction. Form Recognizer applies advanced machine learning to accurately extract text, key/value pairs, and tables from documents. With just a few samples, Form Recognizer tailors its understanding to your documents, both on-premises and in the cloud. Turn forms into usable data at a fraction of the time and cost, so you can focus more time acting on the information rather than compiling it. [Learn more.](#)

Project Details

Subscription * ⓘ my-subscription

Resource group * ⓘ my-resource-group

[Create new](#)

Instance Details

Region ⓘ West US 2

Name * ⓘ my-resource

Pricing tier * ⓘ Commitment tier disconnected containers DCO

[View full pricing details](#)

Commitment tier pricing (charged annually)

Select a commitment tier pricing for the capabilities you wish to use. With commitment tier pricing, you will be charged a fixed amount every year for a given quota. If you need additional quota, you could either increase the number of units. [Learn more](#)

Custom API/Invoice * ⓘ Tier 2 (100,000 pages/month for 31,200 USD per unit)

Unit count * ⓘ 1 units

Annual cost (in USD): \$31200

Annual quota (pages/month): 100000

 You will be charged for the full year as soon as the resource is created

I agree to getting charged for the full year as per the commitment plan tier price at the time of resource creation



[Review + create](#)

[< Previous](#)

[Next : Network >](#)



3. Select **Review + Create** at the bottom of the page. Review the information, and select **Create**.

Gather 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 when you first run the container, to configure it for disconnected usage.

You can find the key and endpoint on the **Key and endpoint** page for your resource.

ⓘ Important

You will only use your key and endpoint to configure the container to be run in a disconnected environment. After you configure the container, you won't need them to send API requests. Store them securely, for example, using Azure Key Vault. Only one key is necessary for this process.

Download a Docker container with `docker pull`

After you have a license file, download the Docker container you have approval to run in a disconnected environment. For example:

Docker

```
docker pull mcr.microsoft.com/azure-cognitive-services/form-  
recognizer/invoice:latest
```

Configure the container to be run in a disconnected environment

Now that you've downloaded your container, you'll need to run the container with the `DownloadLicense=True` parameter in your `docker run` command. This parameter will download a license file that will enable your Docker container to run when it isn't connected to the internet. It also contains an expiration date, after which the license file will be invalid to run the container. You can only use a license file with the appropriate container that you've been approved for. For example, you can't use a license file for a speech-to-text container with a form recognizer container. Please do not rename or modify the license file as this will prevent the container from running successfully.

ⓘ Important

- **Translator container only:**

- You must include a parameter to download model files for the [languages](#) you want to translate. For example: `-e Languages=en,es`
- The container will generate a `docker run` template that you can use to run the container, containing parameters you will need for the downloaded models and configuration file. Make sure you save this template.

The following example shows the formatting of the `docker run` command you'll use, with placeholder values. Replace these placeholder values with your own values.

Placeholder	Value	Format or example
{IMAGE}	The container image you want to use.	<code>mcr.microsoft.com/azure-cognitive-services/form-recognizer/invoice</code>
{LICENSE_MOUNT}	The path where the license will be downloaded, and mounted.	<code>/host/license:/path/to/license/directory</code>
{ENDPOINT_URI}	The endpoint for authenticating your service request. You can find it on your resource's Key and endpoint page, on the Azure portal.	<code>https://<your-custom-subdomain>.cognitiveservices.azure.com</code>
{API_KEY}	The key for your Text Analytics resource. You can find it on your resource's Key and endpoint page, on the Azure portal.	<code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code>
{CONTAINER_LICENSE_DIRECTORY}	Location of the license folder on the container's local filesystem.	<code>/path/to/license/directory</code>

Bash

```
docker run --rm -it -p 5000:5000 \
-v {LICENSE_MOUNT} \
{IMAGE} \
eula=accept \
billing={ENDPOINT_URI} \
apikey={API_KEY} \
DownloadLicense=True \
Mounts:License={CONTAINER_LICENSE_DIRECTORY}
```

After you've configured the container, use the next section to run the container in your environment with the license, and appropriate memory and CPU allocations.

Run the container in a disconnected environment

Important

If you're using the Translator, Neural text-to-speech, or Speech-to-text containers, read the **Additional parameters** section below for information on commands or additional parameters you will need to use.

Once the license file has been downloaded, you can run the container in a disconnected environment. The following example shows the formatting of the `docker run` command you'll use, with placeholder values. Replace these placeholder values with your own values.

Wherever the container is run, the license file must be mounted to the container and the location of the license folder on the container's local filesystem must be specified with `Mounts:License=`. An output mount must also be specified so that billing usage records can be written.

Placeholder	Value	Format or example
{IMAGE}	The container image you want to use.	<code>mcr.microsoft.com/azure-cognitive-services/form-recognizer/invoice</code>
{MEMORY_SIZE}	The appropriate size of memory to allocate for your container.	<code>4g</code>

Placeholder	Value	Format or example
{NUMBER_CPUS}	The appropriate number of CPUs to allocate for your container.	4
{LICENSE_MOUNT}	The path where the license will be located and mounted.	/host/license:/path/to/license/directory
{OUTPUT_PATH}	The output path for logging usage records.	/host/output:/path/to/output/directory
{CONTAINER_LICENSE_DIRECTORY}	Location of the license folder on the container's local filesystem.	/path/to/license/directory
{CONTAINER_OUTPUT_DIRECTORY}	Location of the output folder on the container's local filesystem.	/path/to/output/directory

Bash

```
docker run --rm -it -p 5000:5000 --memory {MEMORY_SIZE} --cpus {NUMBER_CPUS}
 \
-v {LICENSE_MOUNT} \
-v {OUTPUT_PATH} \
{IMAGE} \
eula=accept \
Mounts:License={CONTAINER_LICENSE_DIRECTORY}
Mounts:Output={CONTAINER_OUTPUT_DIRECTORY}
```

Additional parameters and commands

See the following sections for additional parameters and commands you may need to run the container.

Translator container

If you're using the [Translator container](#), you'll need to add parameters for the downloaded translation models and container configuration. These values are generated

and displayed in the container output when you [configure the container](#) as described above. For example:

```
Bash
```

```
-e MODELS= /path/to/model1/, /path/to/model2/
-e TRANSLATORSYSTEMCONFIG=/path/to/model/config/translatorsystemconfig.json
```

Speech containers

Speech-to-text

The [Speech-to-Text](#) container provides a default directory for writing the license file and billing log at runtime. The default directories are `/license` and `/output` respectively.

When you're mounting these directories to the container with the `docker run -v` command, make sure the local machine directory is set ownership to `user:group nonroot:nonroot` before running the container.

Below is a sample command to set file/directory ownership.

```
Bash
```

```
sudo chown -R nonroot:nonroot <YOUR_LOCAL_MACHINE_PATH_1>
<YOUR_LOCAL_MACHINE_PATH_2> ...
```

Additional resources

Documentation

[Cognitive Services containers FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions.

[Use Azure Cognitive Services Containers on-premises - Azure Cognitive Services](#)

Learn how to use Docker containers to use Cognitive Services on-premises.

[Azure Container Instance recipe - Azure Cognitive Services](#)

Learn how to deploy Cognitive Services Containers on Azure Container Instance

[Running Cognitive Services containers in disconnected environments FAQ - Azure Cognitive Services](#)

Get answers to frequently asked questions about running containers disconnected from the internet.

[Sign up for Azure Video Indexer and upload your first video - Azure - Azure Video Indexer](#)

Learn how to sign up and upload your first video using the Azure Video Indexer website.

[Configure Read OCR containers - Computer Vision - Azure Cognitive Services](#)

This article shows you how to configure both required and optional settings for Read OCR containers in Computer Vision.

[Frequently asked questions - Computer Vision - Azure Cognitive Services](#)

Get answers to frequently asked questions about the Computer Vision Service in Azure Cognitive Services.

[Computer Vision 3.2 GA Read OCR container - Azure Cognitive Services](#)

Use the Read 3.2 OCR containers from Computer Vision to extract text from images and documents, on-premises.

[Show 5 more](#)

Speech service containers frequently asked questions (FAQ)

FAQ

When using the Speech service with containers, you can refer to these frequently asked questions before contacting support. This article captures questions of varying degree, from general to technical.

General questions

How do Speech containers work and how do I set them up?

For an overview, see [Install and run Speech containers](#). When setting up the production cluster, there are several things to consider. First, setting up single language, multiple containers, on the same machine, should not be a large issue. If you are experiencing problems, it may be a hardware-related issue - so we would first look at resource, that is; CPU and memory specifications.

Consider for a moment, the `ja-JP` container and latest model. The acoustic model is the most demanding piece CPU-wise, while the language model demands the most memory. When we benchmarked the use, it takes about 0.6 CPU cores to process a single speech-to-text request when audio is flowing in at real-time, for example, from a microphone. If you are feeding audio faster than real-time, for example, from a file, that usage can double (1.2x cores). Meanwhile, the memory in this context is operating memory for decoding speech. It does *not* take into account the actual full size of the language model, which will reside in file cache. It's an additional 2 GB for `ja-JP`; for `en-US`, it may be more (6-7 GB).

If you have a machine where memory is scarce, and you are trying to deploy multiple languages on it, it is possible that file cache is full, and the OS is forced to page models in and out. For a running transcription, that could be disastrous, and may lead to slowdowns and other performance implications.

Furthermore, we pre-package executables for machines with the [advanced vector extension \(AVX2\)](#) instruction set. A machine with the AVX512 instruction set will require code generation for that target, and starting 10 containers for 10 languages may temporarily exhaust CPU. A message like this one will appear in the docker logs:

```
Console  
2020-01-16 16:46:54.981118943  
[W:onnxruntime:Default, tvm_utils.cc:276 LoadTVMFuncFromCache]
```

```
Cannot find Scan4_11vm__mcpu_skylake_avx512 in cache, using JIT...
```

You can set the number of decoders you want inside a *single* container using `DECODER_MAX_COUNT` variable. Start with your CPU and memory SKU and then refer to the recommended host machine resource specifications.

Could you help with capacity planning and cost estimation of on-prem Speech-to-text containers?

For container capacity in batch processing mode, each decoder could handle 2-3x in real-time, with two CPU cores, for a single recognition. We do not recommend keeping more than two concurrent recognitions per container instance, but recommend running more instances of containers for reliability and availability reasons, behind a load balancer.

Though we could have each container instance running with more decoders. For example, we may be able to set up seven decoders per container instance on an eight-core machine at more than 2x each, yielding 15x throughput. There is a param `DECODER_MAX_COUNT` to be aware of. For the extreme case, reliability and latency issues arise, with throughput increased significantly. For a microphone, it will be at 1x real-time. The overall usage should be at about one core for a single recognition.

For scenario of processing 1-K hours per day in batch processing mode, in an extreme case, 3 VMs could handle it within 24 hours but not guaranteed. To handle spike days, failover, update, and to provide minimum backup/BCP, we recommend 4-5 machines instead of 3 per cluster, and with 2+ clusters.

For hardware, we use standard Azure VM `DS13_v2` as a reference (each core must be 2.6 GHz or better, with AVX2 instruction set enabled).

Instance	vCPU(s)	RAM	Temp storage	Pay-as-you-go with AHB	1-year reserve with AHB (% Savings)	3-year reserved with AHB (% Savings)
<code>DS13_v2</code>	8	56 GiB	112 GiB	\$0.598/hour	\$0.3528/hour (~41%)	\$0.2333/hour (~61%)

Based on the design reference (two clusters of 5 VMs to handle 1 K hours/day audio batch processing), one-year hardware cost will be:

$$2 \text{ (clusters)} * 5 \text{ (VMs per cluster)} * \$0.3528/\text{hour} * 365 \text{ (days)} * 24 \text{ (hours)} = \$31 \text{ K / year}$$

When mapping to physical machine, a general estimation is 1 vCPU = 1 Physical CPU Core. In reality, 1vCPU is more powerful than a single core.

For on-prem, all of these additional factors come into play:

- On what type the physical CPU is and how many cores on it
- How many CPUs running together on the same box/machine
- How VMs are set up
- How hyper-threading / multi-threading is used
- How memory is shared
- The OS, etc.

Normally it is not as well tuned as Azure the environment. Considering other overhead, one estimation is 10 physical CPU cores = 8 Azure vCPU. Though popular CPUs only have eight cores. With on-prem deployment, the cost will be higher than using Azure VMs. Also, consider the depreciation rate.

Service cost is the same as the online service: \$1/hour for speech-to-text. The Speech service cost is:

$$\$1 * 1000 * 365 = \$365 \text{ K}$$

Maintenance cost paid to Microsoft depends on the service level and content of the service. People cost is not included. Other infrastructure costs such as storage, networks, and load balancers are not included.

Can I use a custom acoustic model and language model with Speech container?

One model ID can be used, either a custom language model or custom acoustic model. Passing acoustic and language models concurrently is not supported.

Could you explain these errors from the custom speech-to-text container?

Error 1:

```
cmd
Failed to fetch manifest: Status: 400 Bad Request Body:
{
    "code": "InvalidModel",
    "message": "The specified model is not supported for endpoint manifests."
}
```

If you're training with the latest custom model, we currently don't support that. If you train with an older version, it should be possible to use. We are still working on supporting the latest versions.

Essentially, the custom containers do not support Halide or ONNX-based acoustic models (which is the default in the custom training portal). This is due to custom models not being encrypted and we don't want to expose ONNX models, however; language models are fine. The model size may be larger by 100 MB.

Error 2:

```
cmd

HTTPAPI result code = HTTPAPI_OK.
HTTP status code = 400.
Reason: Synthesis failed.
StatusCode: InvalidArgumentException,
Details: Voice does not match.
```

You need to provide the correct voice name in the request, which is case-sensitive. Refer to the full service name mapping.

Error 3:

```
JSON

{
  "code": "InvalidProductId",
  "message": "The subscription SKU \"CognitiveServices.S0\" is not supported in
this service instance."
}
```

You need to create a Speech resource, not a Cognitive Services resource.

What API protocols are supported, REST or WS?

For speech-to-text and custom speech-to-text containers, we currently only support the websocket based protocol. The SDK only supports calling in WS but not REST. Always refer to the official documentation, see [query prediction endpoints](#).

Why am I getting errors when attempting to call LUIS prediction endpoints?

I am using the LUIS container in an IoT Edge deployment and am attempting to call the LUIS prediction endpoint from another container. The LUIS container is listening on port 5001, and the URL I'm using is this:

```
C#
```

```
var luisEndpoint =  
    $"ws://192.168.1.91:5001/luis/prediction/v3.0/apps/{luisAppId}/slots/production/pr  
edict";  
var config = SpeechConfig.FromEndpoint(new Uri(luisEndpoint));
```

The error I'm getting is:

```
cmd
```

```
WebSocket Upgrade failed with HTTP status code: 404 SessionId:  
3cfe2509ef4e49919e594abf639ccfeb
```

I see the request in the LUIS container logs and the message says:

```
cmd
```

```
The request path "/luis//predict" does not match a supported file type.
```

The Speech SDK should not be used for a LUIS container. For using the LUIS container, the LUIS SDK or LUIS REST API should be used. Speech SDK should be used for a speech container.

A cloud is different than a container. A cloud can be composed of multiple aggregated containers. In this context there are two separate LUIS and Speech containers deployed in the cloud. The Speech container only does speech. The LUIS container only does LUIS. Performance would be slow for a remote client to go to the cloud, do speech, come back, then go to the cloud again and do LUIS. So we provide a feature that allows the client to go to Speech, stay in the cloud, go to LUIS then come back to the client. Thus even in this scenario the Speech SDK goes to Speech cloud container with audio, and then Speech cloud container talks to LUIS cloud container with text. The LUIS container has no concept of accepting audio. Since LUIS is a text-based service, a LUIS container doesn't accept streaming audio. With on-prem, we have no certainty our customer has deployed both containers, we don't presume to orchestrate between containers in our customers' premises, and if both containers are deployed on-prem, given they are more local to the client, it is not a burden to go the SR first, back to client, and have the customer then take that text and go LUIS.

How can we benchmark a rough measure of transactions/second/core?

Here are some of the rough numbers to expect from the model prior to general availability (GA):

- For files, the throttling will be in the Speech SDK, at 2x. First five seconds of audio are not throttled. Decoder is capable of doing about 3x real-time. For this, the overall CPU usage will be close to two cores for a single recognition.
- For mic, it will be at 1x real-time. The overall usage should be at about one core for a single recognition.

This can all be verified from the docker logs. We actually dump the line with session and phrase/utterance statistics, and that includes the RTF numbers.

How do I make multiple containers run on the same host?

By setting each container to listen on a different port using the -p argument. Use `-p <outside_unique_port>:5000`. For example: `-p 5001:5000`.

Technical questions

How can I get non-batch APIs to handle audio <15 seconds long?

The `RecognizeOnce()` SDK operation in interactive mode processes up to 15 seconds of audio where utterances are expected to be short. You use `StartContinuousRecognition()` for dictation or conversation longer than 15 seconds.

What are the recommended resources, CPU and RAM; for 50 concurrent requests?

How many concurrent requests will a 4-core, 4-GB RAM handle? If we have to serve for example, 50 concurrent requests, how many Core and RAM is recommended?

At real-time, 8 with our latest `en-US`, so we recommend using more docker containers beyond six concurrent requests. Beyond 16 cores it becomes non-uniform memory access (NUMA) node sensitive. The following table describes the minimum and recommended allocation of resources for each Speech container.

Speech-to-text		
Container	Minimum	Recommended
Speech-to-text	2-core, 2-GB memory	4-core, 4-GB memory

- Each core must be at least 2.6 GHz or faster.
- For files, the throttling will be in the Speech SDK, at 2x. The first 5 seconds of audio are not throttled.
- The decoder is capable of doing about 2-3x real-time. For this, the overall CPU usage will be close to two cores for a single recognition. That's why we do not recommend keeping more than two active connections, per container instance. The extreme side would be to put about 10 decoders at 2x real-time in an eight-core machine like `DS13_v2`. For the container version 1.3 and later, there's a param you could try setting `DECODER_MAX_COUNT=20`.
- For microphone, it will be at 1x real-time. The overall usage should be at about one core for a single recognition.

Consider the total number of hours of audio you have. If the number is large, to improve reliability and availability, we suggest running more instances of containers, either on a single box or on multiple boxes, behind a load balancer. Orchestration could be done using Kubernetes (K8S) and Helm, or with Docker compose.

As an example, to handle 1000 hours/24 hours, we have tried setting up 3-4 VMs, with 10 instances/decoders per VM.

Does the Speech container support punctuation?

We have capitalization (ITN) available in the on-prem container. Punctuation is language-dependent, and not supported for some languages, including Chinese and Japanese.

We *do* have implicit and basic punctuation support for the existing containers, but it is `off` by default. What that means is that you can get the `.` character in your example, but not the `,` character. To enable this implicit logic, here's an example of how to do so in Python using our Speech SDK. It would be similar in other languages:

Python

```
speech_config.set_service_property(
    name='punctuation',
    value='implicit',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)
```

Why am I getting 404 errors when attempting to POST data to speech-to-text container?

Speech-to-text containers do not support REST API. The Speech SDK uses WebSockets. Always refer to the official documentation, see [query prediction endpoints](#).

Why is the container running as a non-root user? What issues might occur because of this?

Note that the default user inside the container is a non-root user. This provides protection against processes escaping the container and obtaining escalated permissions on the host node. By default, some platforms like the OpenShift Container Platform already do this by running containers using an arbitrarily assigned user ID. For these platforms, the non-root user will need to have permissions to write to any externally mapped volume that requires writes. For example a logging folder, or a custom model download folder.

When using the speech-to-text service, why am I getting this error?

cmd

```
Error in STT call for file 9136835610040002161_413008000252496:  
{  
    "reason": "ResultReason.Canceled",  
    "error_details": "Due to service inactivity the client buffer size exceeded.  
Resetting the buffer. SessionId: xxxxx..."  
}
```

Cancellation can occur when the audio feed exceeds the Speech container buffer size. You need to control the concurrency and the Real-Time Factor (RTF) at which you send the audio. The RTF can fluctuate depending on concurrent requests, CPU, and memory allocated.

Next steps

[Cognitive Services containers](#)

Migration to neural voice

Article • 01/13/2023 • 2 minutes to read

We're retiring two features from [Text-to-Speech](#) capabilities as detailed below.

Custom voice (non-neural training)

ⓘ Important

We are retiring the standard non-neural training tier of custom voice from March 1, 2021 through February 29, 2024. If you used a non-neural custom voice with your Speech resource prior to March 1, 2021 then you can continue to do so until February 29, 2024. All other Speech resources can only use custom neural voice. After February 29, 2024, the non-neural custom voices won't be supported with any Speech resource.

Go to [this article](#) to learn how to migrate to custom neural voice.

Custom neural voice is a gated service. You need to create an Azure account and Speech service subscription (with S0 tier), and [apply](#) to use custom neural feature. After you've been granted access, you can visit the [Speech Studio portal](#) and then select Custom Voice to get started.

Even without an Azure account, you can listen to voice samples in [Speech Studio](#) and determine the right voice for your business needs.

Go to the [pricing page](#) and check the pricing details. Custom voice (retired) is referred as **Custom**, and custom neural voice is referred as **Custom Neural**.

Prebuilt standard voice

ⓘ Important

We are retiring the standard voices from September 1, 2021 through August 31, 2024. If you used a standard voice with your Speech resource prior to September 1, 2021 then you can continue to do so until August 31, 2024. All other Speech resources can only use prebuilt neural voices. You can choose from the supported [neural voice names](#). After August 31, the standard voices won't be supported with any Speech resource.

Go to [this article](#) to learn how to migrate to prebuilt neural voice.

Prebuilt neural voice is powered by deep neural networks. You need to create an Azure account and Speech service subscription. Then you can use the [Speech SDK](#) or visit the [Speech Studio portal](#), and select prebuilt neural voices to get started. Listening to the voice sample without creating an Azure account, you can visit the [Voice Gallery](#) and determine the right voice for your business needs.

Go to the [pricing page](#) and check the pricing details. Prebuilt standard voice (retired) is referred as **Standard**, and prebuilt neural voice is referred as **Neural**.

Next steps

- [Try out prebuilt neural voice](#)
- [Try out custom neural voice](#)

Migrate from custom voice to custom neural voice

Article • 10/27/2022 • 3 minutes to read

Important

We are retiring the standard non-neural training tier of custom voice from March 1, 2021 through February 29, 2024. If you used a non-neural custom voice with your Speech resource prior to March 1, 2021 then you can continue to do so until February 29, 2024. All other Speech resources can only use custom neural voice. After February 29, 2024, the non-neural custom voices won't be supported with any Speech resource.

The custom neural voice lets you build higher-quality voice models while requiring less data. You can develop more realistic, natural, and conversational voices. Your customers and end users will benefit from the latest Text-to-Speech technology, in a responsible way.

Custom voice	Custom neural voice
The standard, or "traditional," method of custom voice breaks down spoken language into phonetic snippets that can be remixed and matched using classical programming or statistical methods.	Custom neural voice synthesizes speech using deep neural networks that have "learned" the way phonetics are combined in natural human speech rather than using classical programming or statistical methods.
Custom voice ¹ requires a large volume of voice data to produce a more human-like voice model. With fewer recorded lines, a standard custom voice model will tend to sound more obviously robotic.	The custom neural voice capability enables you to create a unique brand voice in multiple languages and styles by using a small set of recordings.

¹ When creating a custom voice model, the maximum number of data files allowed to be imported per subscription is 10 .zip files for free subscription (F0) users, and 500 for standard subscription (S0) users.

Action required

Before you can migrate to custom neural voice, your [application](#) must be accepted. Access to the custom neural voice service is subject to Microsoft's sole discretion based

on our eligibility criteria. You must commit to using custom neural voice in alignment with our [Responsible AI principles](#) and the [code of conduct](#).

💡 Tip

Even without an Azure account, you can listen to voice samples in [Speech Studio](#) and determine the right voice for your business needs.

1. Learn more about our [policy on the limit access](#) and then [apply here](#).
2. Once your application is approved, you will be provided with the access to the "neural" training feature. Make sure you log in to [Speech Studio](#) using the same Azure subscription that you provide in your application.
3. Before you can [train](#) and [deploy](#) a custom voice model, you must [create a voice talent profile](#). The profile requires an audio file recorded by the voice talent consenting to the usage of their speech data to train a custom voice model.
4. Update your code in your apps if you have created a new endpoint with a new model.

Custom voice details (deprecated)

Read the following sections for details on custom voice.

Language support

Custom voice supports the following languages (locales).

Language	Locale
Chinese (Mandarin, Simplified)	zh-CN
Chinese (Mandarin, Simplified), English bilingual	zh-CN bilingual
English (India)	en-IN
English (United Kingdom)	en-GB
English (United States)	en-US
French (France)	fr-FR
German (Germany)	de-DE
Italian (Italy)	it-IT

Language	Locale
Portuguese (Brazil)	pt-BR
Spanish (Mexico)	es-MX

Regional support

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	<code>https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Brazil South	<code>https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Canada Central	<code>https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Central US	<code>https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
East Asia	<code>https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
East US	<code>https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
East US 2	<code>https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
France Central	<code>https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
India Central	<code>https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Japan East	<code>https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Japan West	<code>https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>

Region	Endpoint
Korea Central	<code>https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
North Central US	<code>https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
North Europe	<code>https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
South Central US	<code>https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Southeast Asia	<code>https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
UK South	<code>https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
West Europe	<code>https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
West Central US	<code>https://westcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
West US	<code>https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
West US 2	<code>https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>

Next steps

[Try out custom neural voice](#)

Migrate from prebuilt standard voice to prebuilt neural voice

Article • 01/13/2023 • 4 minutes to read

ⓘ Important

We are retiring the standard voices from September 1, 2021 through August 31, 2024. If you used a standard voice with your Speech resource prior to September 1, 2021 then you can continue to do so until August 31, 2024. All other Speech resources can only use prebuilt neural voices. You can choose from the supported [neural voice names](#). After August 31, the standard voices won't be supported with any Speech resource.

The prebuilt neural voice provides more natural sounding speech output, and thus, a better end-user experience.

Prebuilt standard voice	Prebuilt neural voice
Noticeably robotic	Natural sounding, closer to human-parity
Limited capabilities in voice tuning ¹	Advanced capabilities in voice tuning
No new investment in future voice fonts	On-going investment in future voice fonts

¹ For voice tuning, volume and pitch changes can be applied to standard voices at the word or sentence-level, whereas they can only be applied to neural voices at the sentence level. Duration supports standard voices only. To learn more about details on prosody elements, see [Improve synthesis with SSML](#).

Action required

ⓘ Tip

Even without an Azure account, you can listen to voice samples at the [Voice Gallery](#) and determine the right voice for your business needs.

1. Review the [price](#) structure.
2. To make the change, [follow the sample code](#) to update the voice name in your speech synthesis request to the supported neural voice names in chosen

languages. Use neural voices for your speech synthesis request, on cloud or on-prem. For on-premises container, use the [neural voice containers](#) and follow the [instructions](#).

Standard voice details (deprecated)

Read the following sections for details on standard voice.

Language support

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

ⓘ Note

With two exceptions, standard voices are created from samples that use a 16 khz sample rate. The **en-US-AriaRUS** and **en-US-GuyRUS** voices are also created from samples that use a 24 khz sample rate. All voices can upsample or downsample to other sample rates when synthesizing.

Language	Locale (BCP-47)	Gender	Voice name
Arabic (Arabic)	ar-EG	Female	ar-EG-Hoda
Arabic (Saudi Arabia)	ar-SA	Male	ar-SA-Naayf
Bulgarian (Bulgaria)	bg-BG	Male	bg-BG-Ivan
Catalan (Spain)	ca-ES	Female	ca-ES-HerenaRUS
Chinese (Cantonese, Traditional)	zh-HK	Male	zh-HK-Danny
Chinese (Cantonese, Traditional)	zh-HK	Female	zh-HK-TracyRUS
Chinese (Mandarin, Simplified)	zh-CN	Female	zh-CN-HuihuiRUS
Chinese (Mandarin, Simplified)	zh-CN	Male	zh-CN-Kangkang
Chinese (Mandarin, Simplified)	zh-CN	Female	zh-CN-Yaoyao
Chinese (Taiwanese Mandarin)	zh-TW	Female	zh-TW-HanHanRUS
Chinese (Taiwanese Mandarin)	zh-TW	Female	zh-TW-Yating
Chinese (Taiwanese Mandarin)	zh-TW	Male	zh-TW-Zhiwei

Language	Locale (BCP-47)	Gender	Voice name
Croatian (Croatia)	hr-HR	Male	hr-HR-Matej
Czech (Czech Republic)	cs-CZ	Male	cs-CZ-Jakub
Danish (Denmark)	da-DK	Female	da-DK-HelleRUS
Dutch (Netherlands)	nl-NL	Female	nl-NL-HannaRUS
English (Australia)	en-AU	Female	en-AU-Catherine
English (Australia)	en-AU	Female	en-AU-HayleyRUS
English (Canada)	en-CA	Female	en-CA-HeatherRUS
English (Canada)	en-CA	Female	en-CA-Linda
English (India)	en-IN	Female	en-IN-Heera
English (India)	en-IN	Female	en-IN-PriyaRUS
English (India)	en-IN	Male	en-IN-Ravi
English (Ireland)	en-IE	Male	en-IE-Sean
English (United Kingdom)	en-GB	Male	en-GB-George
English (United Kingdom)	en-GB	Female	en-GB-HazelRUS
English (United Kingdom)	en-GB	Female	en-GB-Susan
English (United States)	en-US	Male	en-US-BenjaminRUS
English (United States)	en-US	Male	en-US-GuyRUS
English (United States)	en-US	Female	en-US-AriaRUS
English (United States)	en-US	Female	en-US-ZiraRUS
Finnish (Finland)	fi-FI	Female	fi-FI-HeidiRUS
French (Canada)	fr-CA	Female	fr-CA-Caroline
French (Canada)	fr-CA	Female	fr-CA-HarmonieRUS
French (France)	fr-FR	Female	fr-FR-HortenseRUS
French (France)	fr-FR	Female	fr-FR-Julie
French (France)	fr-FR	Male	fr-FR-Paul

Language	Locale (BCP-47)	Gender	Voice name
French (Switzerland)	fr-CH	Male	fr-CH-Guillaume
German (Austria)	de-AT	Male	de-AT-Michael
German (Germany)	de-DE	Female	de-DE-HeddaRUS
German (Germany)	de-DE	Male	de-DE-Stefan
German (Switzerland)	de-CH	Male	de-CH-Karsten
Greek (Greece)	el-GR	Male	el-GR-Stefanos
Hebrew (Israel)	he-IL	Male	he-IL-Asaf
Hindi (India)	hi-IN	Male	hi-IN-Hemant
Hindi (India)	hi-IN	Female	hi-IN-Kalpana
Hungarian (Hungary)	hu-HU	Male	hu-HU-Szabolcs
Indonesian (Indonesia)	id-ID	Male	id-ID-Andika
Italian (Italy)	it-IT	Male	it-IT-Cosimo
Italian (Italy)	it-IT	Female	it-IT-LuciaRUS
Japanese (Japan)	ja-JP	Female	ja-JP-Ayumi
Japanese (Japan)	ja-JP	Female	ja-JP-HarukaRUS
Japanese (Japan)	ja-JP	Male	ja-JP-Ichiro
Korean (Korea)	ko-KR	Female	ko-KR-HeamirUS
Malay (Malaysia)	ms-MY	Male	ms-MY-Rizwan
Norwegian (Bokmål, Norway)	nb-NO	Female	nb-NO-HuldaRUS
Polish (Poland)	pl-PL	Female	pl-PL-PaulinaRUS
Portuguese (Brazil)	pt-BR	Male	pt-BR-Daniel
Portuguese (Brazil)	pt-BR	Female	pt-BR-HeiloisaRUS
Portuguese (Portugal)	pt-PT	Female	pt-PT-HeliaRUS
Romanian (Romania)	ro-RO	Male	ro-RO-Andrei
Russian (Russia)	ru-RU	Female	ru-RU-EkaterinaRUS

Language	Locale (BCP-47)	Gender	Voice name
Russian (Russia)	ru-RU	Female	ru-RU-Irina
Russian (Russia)	ru-RU	Male	ru-RU-Pavel
Slovak (Slovakia)	sk-SK	Male	sk-SK-Filip
Slovenian (Slovenia)	sl-SI	Male	sl-SI-Lado
Spanish (Mexico)	es-MX	Female	es-MX-HildaRUS
Spanish (Mexico)	es-MX	Male	es-MX-Raul
Spanish (Spain)	es-ES	Female	es-ES-HelenaRUS
Spanish (Spain)	es-ES	Female	es-ES-Laura
Spanish (Spain)	es-ES	Male	es-ES-Pablo
Swedish (Sweden)	sv-SE	Female	sv-SE-HedvigRUS
Tamil (India)	ta-IN	Male	ta-IN-Valluvar
Telugu (India)	te-IN	Female	te-IN-Chitra
Thai (Thailand)	th-TH	Male	th-TH-Pattara
Turkish (Turkey)	tr-TR	Female	tr-TR-SedaRUS
Vietnamese (Vietnam)	vi-VN	Male	vi-VN-An

ⓘ Important

The en-US-Jessa voice has changed to en-US-Aria. If you were using "Jessa" before, convert over to "Aria".

You can continue to use the full service name mapping like "Microsoft Server Speech Text to Speech Voice (en-US, AriaRUS)" in your speech synthesis requests.

Regional support

Use this table to determine availability of standard voices by region/endpoint:

Region	Endpoint
Australia East	https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1

Region	Endpoint
Brazil South	https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/v1
Canada Central	https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1
Central US	https://centralus.tts.speech.microsoft.com/cognitiveservices/v1
East Asia	https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1
East US	https://eastus.tts.speech.microsoft.com/cognitiveservices/v1
East US 2	https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1
France Central	https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1
India Central	https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1
Japan East	https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1
Japan West	https://japanwest.tts.speech.microsoft.com/cognitiveservices/v1
Korea Central	https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1
North Central US	https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1
North Europe	https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1
South Central US	https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1
Southeast Asia	https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1
UK South	https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1
West Central US	https://westcentralus.tts.speech.microsoft.com/cognitiveservices/v1
West Europe	https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1
West US	https://westus.tts.speech.microsoft.com/cognitiveservices/v1
West US 2	https://westus2.tts.speech.microsoft.com/cognitiveservices/v1

Next steps

[Try out prebuilt neural voice](#)

Migrate code from v3.0 to v3.1 of the REST API

Article • 01/25/2023 • 5 minutes to read

The Speech-to-text REST API is used for [Batch transcription](#) and [Custom Speech](#). Changes from version 3.0 to 3.1 are described in the sections below.

Important

Speech-to-text REST API v3.1 is generally available. Version 3.0 of the [Speech to Text REST API](#) will be retired.

Base path

You must update the base path in your code from `/speechtotext/v3.0` to `/speechtotext/v3.1`. For example, to get base models in the `eastus` region, use `https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/models/base` instead of `https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/models/base`.

Please note these additional changes:

- The `/models/{id}/copyto` operation (includes '/') in version 3.0 is replaced by the `/models/{id}:copyto` operation (includes ':') in version 3.1.
- The `/webhooks/{id}/ping` operation (includes '/') in version 3.0 is replaced by the `/webhooks/{id}:ping` operation (includes ':') in version 3.1.
- The `/webhooks/{id}/test` operation (includes '/') in version 3.0 is replaced by the `/webhooks/{id}:test` operation (includes ':') in version 3.1.

For more details, see [Operation IDs](#) later in this guide.

Batch transcription

Note

Don't use Speech-to-text REST API v3.0 to retrieve a transcription created via Speech-to-text REST API v3.1. You'll see an error message such as the following: "The API version cannot be used to access this transcription. Please use API version v3.1 or higher."

In the [Transcriptions_Create](#) operation the following three properties are added:

- The `displayFormWordLevelTimestampsEnabled` property can be used to enable the reporting of word-level timestamps on the display form of the transcription results. The results are returned in the `displayPhraseElements` property of the transcription file.
- The `diarization` property can be used to specify hints for the minimum and maximum number of speaker labels to generate when performing optional diarization (speaker separation). With this feature, the service is now able to generate speaker labels for more than two speakers. The `diarizationEnabled` property is deprecated and will be removed in the next major version of the API.
- The `languageIdentification` property can be used to specify settings for language identification on the input prior to transcription. Up to 10 candidate locales are supported for language identification. The returned transcription will include a new `locale` property for the recognized language or the locale that you provided.

The `filter` property is added to the [Transcriptions_List](#), [Transcriptions_ListFiles](#), and [Projects_ListTranscriptions](#) operations. The `filter` expression can be used to select a subset of the available resources. You can filter by

`displayName`, `description`, `createdDateTime`, `lastActionDateTime`, `status`, and `locale`. For example:

```
filter=createdDateTime gt 2022-02-01T11:00:00Z
```

If you use webhook to receive notifications about transcription status, please note that the webhooks created via V3.0 API cannot receive notifications for V3.1 transcription requests. You need to create a new webhook endpoint via V3.1 API in order to receive notifications for V3.1 transcription requests.

Custom Speech

Datasets

The following operations are added for uploading and managing multiple data blocks for a dataset:

- [Datasets_UploadBlock](#) - Upload a block of data for the dataset. The maximum size of the block is 8MiB.
- [Datasets_GetBlocks](#) - Get the list of uploaded blocks for this dataset.
- [Datasets_CommitBlocks](#) - Commit blocklist to complete the upload of the dataset.

To support model adaptation with [structured text in markdown](#) data, the [Datasets_Create](#) operation now supports the **LanguageMarkdown** data kind. For more information, see [upload datasets](#).

Models

The [Models_ListBaseModels](#) and [Models_GetBaseModel](#) operations return information on the type of adaptation supported by each base model.

JSON

```
"features": {  
    "supportsAdaptationsWith": [  
        "Acoustic",  
        "Language",  
        "LanguageMarkdown",  
        "Pronunciation"  
    ]  
}
```

The [Models_Create](#) operation has a new `customModelWeightPercent` property where you can specify the weight used when the Custom Language Model (trained from plain or structured text data) is combined with the Base Language Model. Valid values are integers between 1 and 100. The default value is currently 30.

The `filter` property is added to the following operations:

- [Datasets_List](#)
- [Datasets_ListFiles](#)
- [Endpoints_List](#)
- [Evaluations_List](#)
- [Evaluations_ListFiles](#)
- [Models_ListBaseModels](#)
- [Models_ListCustomModels](#)
- [Projects_List](#)
- [Projects_ListDatasets](#)
- [Projects_ListEndpoints](#)
- [Projects_ListEvaluations](#)
- [Projects_ListModels](#)

The `filter` expression can be used to select a subset of the available resources. You can filter by `displayName`, `description`, `createdDateTime`, `lastActionDateTime`, `status`, `locale`, and `kind`. For example: `filter=locale eq 'en-US'`

Added the [Models_ListFiles](#) operation to get the files of the model identified by the given ID.

Added the [Models_GetFile](#) operation to get one specific file (identified with fileId) from a model (identified with ID). This lets you retrieve a **ModelReport** file that provides information on the data processed during training.

Operation IDs

You must update the base path in your code from `/speechtotext/v3.0` to `/speechtotext/v3.1`. For example, to get base models in the `eastus` region, use `https://eastus.api.cognitive.microsoft.com/speechtotext/v3.1/models/base` instead of `https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/models/base`.

The name of each `operationId` in version 3.1 is prefixed with the object name. For example, the `operationId` for "Create Model" changed from [CreateModel](#) in version 3.0 to [Models_Create](#) in version 3.1.

Path	Method	Version 3.1 Operation ID	Version 3.0 Operation ID
<code>/datasets</code>	GET	Datasets_List	GetDatasets
<code>/datasets</code>	POST	Datasets_Create	CreateDataset
<code>/datasets/{id}</code>	DELETE	Datasets_Delete	DeleteDataset
<code>/datasets/{id}</code>	GET	Datasets_Get	GetDataset
<code>/datasets/{id}</code>	PATCH	Datasets_Update	UpdateDataset
<code>/datasets/{id}/blocks:commit</code>	POST	Datasets_CommitBlocks	Not applicable
<code>/datasets/{id}/blocks</code>	GET	Datasets_GetBlocks	Not applicable
<code>/datasets/{id}/blocks</code>	PUT	Datasets_UploadBlock	Not applicable
<code>/datasets/{id}/files</code>	GET	Datasets_ListFiles	GetDatasetFiles
<code>/datasets/{id}/files/{fileId}</code>	GET	Datasets_GetFile	GetDatasetFile
<code>/datasets/locales</code>	GET	Datasets_ListSupportedLocales	GetSupportedLocalesForDatasets
<code>/datasets/upload</code>	POST	Datasets_Upload	UploadDatasetFromForm
<code>/endpoints</code>	GET	Endpoints_List	GetEndpoints
<code>/endpoints</code>	POST	Endpoints_Create	CreateEndpoint
<code>/endpoints/{id}</code>	DELETE	Endpoints_Delete	DeleteEndpoint
<code>/endpoints/{id}</code>	GET	Endpoints_Get	GetEndpoint
<code>/endpoints/{id}</code>	PATCH	Endpoints_Update	UpdateEndpoint
<code>/endpoints/{id}/files/logs</code>	DELETE	Endpoints_DeleteLogs	DeleteEndpointLogs
<code>/endpoints/{id}/files/logs</code>	GET	Endpoints_ListLogs	GetEndpointLogs
<code>/endpoints/{id}/files/logs/{logId}</code>	DELETE	Endpoints_DeleteLog	DeleteEndpointLog
<code>/endpoints/{id}/files/logs/{logId}</code>	GET	Endpoints_GetLog	GetEndpointLog
<code>/endpoints/base/{locale}/files/logs</code>	DELETE	Endpoints_DeleteBaseModelLogs	DeleteBaseModelLogs
<code>/endpoints/base/{locale}/files/logs</code>	GET	Endpoints_ListBaseModelLogs	GetBaseModelLogs

Path	Method	Version 3.1 Operation ID	Version 3.0 Operation ID
/endpoints/base/{locale}/files/logs/{logId}	DELETE	Endpoints_DeleteBaseModelLog ↗	DeleteBaseModelLog ↗
/endpoints/base/{locale}/files/logs/{logId}	GET	Endpoints_GetBaseModelLog ↗	GetBaseModelLog ↗
/endpoints/locales	GET	Endpoints_ListSupportedLocales ↗	GetSupportedLocalesForEndpoints ↗
/evaluations	GET	Evaluations_List ↗	GetEvaluations ↗
/evaluations	POST	Evaluations_Create ↗	CreateEvaluation ↗
/evaluations/{id}	DELETE	Evaluations_Delete ↗	DeleteEvaluation ↗
/evaluations/{id}	GET	Evaluations_Get ↗	GetEvaluation ↗
/evaluations/{id}	PATCH	Evaluations_Update ↗	UpdateEvaluation ↗
/evaluations/{id}/files	GET	Evaluations_ListFiles ↗	GetEvaluationFiles ↗
/evaluations/{id}/files/{fileId}	GET	Evaluations_GetFile ↗	GetEvaluationFile ↗
/evaluations/locales	GET	Evaluations_ListSupportedLocales ↗	GetSupportedLocalesForEvaluations ↗
/healthstatus	GET	HealthStatus_Get ↗	GetHealthStatus ↗
/models	GET	Models_ListCustomModels ↗	GetModels ↗
/models	POST	Models_Create ↗	CreateModel ↗
/models/{id}:copyto ¹	POST	Models_CopyTo ↗	CopyModelToSubscription ↗
/models/{id}	DELETE	Models_Delete ↗	DeleteModel ↗
/models/{id}	GET	Models_GetCustomModel ↗	GetModel ↗
/models/{id}	PATCH	Models_Update ↗	UpdateModel ↗
/models/{id}/files	GET	Models_ListFiles ↗	Not applicable
/models/{id}/files/{fileId}	GET	Models_GetFile ↗	Not applicable
/models/{id}/manifest	GET	Models_GetCustomModelManifest ↗	GetModelManifest ↗
/models/base	GET	Models_ListBaseModels ↗	GetBaseModels ↗
/models/base/{id}	GET	Models_Get BaseModel ↗	Get BaseModel ↗
/models/base/{id}/manifest	GET	Models_Get BaseModelManifest ↗	Get BaseModelManifest ↗
/models/locales	GET	Models_ListSupportedLocales ↗	GetSupportedLocalesForModels ↗
/projects	GET	Projects_List ↗	GetProjects ↗
/projects	POST	Projects_Create ↗	CreateProject ↗
/projects/{id}	DELETE	Projects_Delete ↗	DeleteProject ↗
/projects/{id}	GET	Projects_Get ↗	GetProject ↗
/projects/{id}	PATCH	Projects_Update ↗	UpdateProject ↗
/projects/{id}/datasets	GET	Projects_ListDatasets ↗	GetDatasetsForProject ↗
/projects/{id}/endpoints	GET	Projects_ListEndpoints ↗	GetEndpointsForProject ↗
/projects/{id}/evaluations	GET	Projects_ListEvaluations ↗	GetEvaluationsForProject ↗
/projects/{id}/models	GET	Projects_ListModels ↗	GetModelsForProject ↗

Path	Method	Version 3.1 Operation ID	Version 3.0 Operation ID
/projects/{id}/transcriptions	GET	Projects_ListTranscriptions ¹	GetTranscriptionsForProject ¹
/projects/locales	GET	Projects_ListSupportedLocales ¹	GetSupportedProjectLocales ¹
/transcriptions	GET	Transcriptions_List ²	GetTranscriptions ²
/transcriptions	POST	Transcriptions_Create ²	CreateTranscription ²
/transcriptions/{id}	DELETE	Transcriptions_Delete ²	DeleteTranscription ²
/transcriptions/{id}	GET	Transcriptions_Get ²	GetTranscription ²
/transcriptions/{id}	PATCH	Transcriptions_Update ²	UpdateTranscription ²
/transcriptions/{id}/files	GET	Transcriptions_ListFiles ²	GetTranscriptionFiles ²
/transcriptions/{id}/files/{fileId}	GET	Transcriptions_GetFile ²	GetTranscriptionFile ²
/transcriptions/locales	GET	Transcriptions_ListSupportedLocales ²	GetSupportedLocalesForTranscriptions ²
/webhooks	GET	WebHooks_List ²	GetHooks ²
/webhooks	POST	WebHooks_Create ²	CreateHook ²
/webhooks/{id}:ping ²	POST	WebHooks_Ping ²	PingHook ²
/webhooks/{id}:test ³	POST	WebHooks_Test ²	TestHook ²
/webhooks/{id}	DELETE	WebHooks_Delete ²	DeleteHook ²
/webhooks/{id}	GET	WebHooks_Get ²	GetHook ²
/webhooks/{id}	PATCH	WebHooks_Update ²	UpdateHook ²

¹ The /models/{id}/copyto operation (includes '/') in version 3.0 is replaced by the /models/{id}:copyto operation (includes ':') in version 3.1.

² The /webhooks/{id}/ping operation (includes '/') in version 3.0 is replaced by the /webhooks/{id}:ping operation (includes ':') in version 3.1.

³ The /webhooks/{id}/test operation (includes '/') in version 3.0 is replaced by the /webhooks/{id}:test operation (includes ':') in version 3.1.

Next steps

- [Speech-to-text REST API](#)
- [Speech-to-text REST API v3.1 reference²](#)
- [Speech-to-text REST API v3.0 reference²](#)

Additional resources

Documentation

[Asynchronous Conversation Transcription - Speech service - Azure Cognitive Services](#)

Learn how to use asynchronous Conversation Transcription using the Speech service. Available for Java and C# only.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[SpeechConfig Class \(Microsoft.CognitiveServices.Speech\) - Azure for .NET Developers](#)

Information about your subscription, including your key and region, endpoint, host, or authorization token.

[Speech SDK audio input stream concepts - Azure Cognitive Services](#)

An overview of the capabilities of the Speech SDK audio input stream API.

[Get batch transcription results - Speech service - Azure Cognitive Services](#)

With batch transcription, the Speech service transcribes the audio data and stores the results in a storage container. You can then retrieve the results from the storage container.

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[azure.cognitiveservices.speech.Recognizer class](#)

Base class for different recognizers

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[Show 5 more](#)

Migrate code from v2.0 to v3.0 of the REST API

Article • 11/30/2022 • 13 minutes to read

Compared to v2, the v3 version of the Speech services REST API for speech-to-text is more reliable, easier to use, and more consistent with APIs for similar services. Most teams can migrate from v2 to v3 in a day or two.

ⓘ Important

The Speech-to-text REST API v2.0 is deprecated and will be retired by February 29, 2024. Please migrate your applications to the Speech-to-text REST API v3.1. Complete the steps in this article and then see the [Migrate code from v3.0 to v3.1 of the REST API](#) guide for additional requirements.

Forward compatibility

All entities from v2 can also be found in the v3 API under the same identity. Where the schema of a result has changed, (for example, transcriptions), the result of a GET in the v3 version of the API uses the v3 schema. The result of a GET in the v2 version of the API uses the same v2 schema. Newly created entities on v3 aren't available in responses from v2 APIs.

Migration steps

This is a summary list of items you need to be aware of when you are preparing for migration. Details are found in the individual links. Depending on your current use of the API not all steps listed here may apply. Only a few changes require non-trivial changes in the calling code. Most changes just require a change to item names.

General changes:

1. [Change the host name](#)
2. [Rename the property id to self in your client code](#)
3. [Change code to iterate over collections of entities](#)
4. [Rename the property name to displayName in your client code](#)

5. Adjust the retrieval of the metadata of referenced entities

6. If you use Batch transcription:

- Adjust code for creating batch transcriptions
- Adapt code to the new transcription results schema
- Adjust code for how results are retrieved

7. If you use Custom model training/testing APIs:

- Apply modifications to custom model training
- Change how base and custom models are retrieved
- Rename the path segment accuracytests to evaluations in your client code

8. If you use endpoints APIs:

- Change how endpoint logs are retrieved

9. Other minor changes:

- Pass all custom properties as customProperties instead of properties in your POST requests
- Read the location from response header Location instead of Operation-Location

Breaking changes

Host name changes

Endpoint host names have changed from `{region}.cris.ai` to `{region}.api.cognitive.microsoft.com`. Paths to the new endpoints no longer contain `api/` because it's part of the hostname. The [Speech-to-text REST API v3.0](#) reference documentation lists valid regions and paths.

Important

Change the hostname from `{region}.cris.ai` to `{region}.api.cognitive.microsoft.com` where region is the region of your speech subscription. Also remove `api/` from any path in your client code.

Identity of an entity

The property `id` is now `self`. In v2, an API user had to know how our paths on the API are being created. This was non-extensible and required unnecessary work from the user. The property `id` (uuid) is replaced by `self` (string), which is location of the entity (URL). The value is still unique between all your entities. If `id` is stored as a string in your code, a rename is enough to support the new schema. You can now use the `self` content as the URL for the `GET`, `PATCH`, and `DELETE` REST calls for your entity.

If the entity has additional functionality available through other paths, they are listed under `links`. The following example for transcription shows a separate method to `GET` the content of the transcription:

ⓘ Important

Rename the property `id` to `self` in your client code. Change the type from `uuid` to `string` if needed.

v2 transcription:

JSON

```
{  
  "id": "9891c965-bb32-4880-b14b-6d44efb158f3",  
  "createdDateTime": "2019-01-07T11:34:12Z",  
  "lastActionDateTime": "2019-01-07T11:36:07Z",  
  "status": "Succeeded",  
  "locale": "en-US",  
  "name": "Transcription using locale en-US"  
}
```

v3 transcription:

JSON

```
{  
  "self":  
    "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions  
    /9891c965-bb32-4880-b14b-6d44efb158f3",  
  "createdDateTime": "2019-01-07T11:34:12Z",  
  "lastActionDateTime": "2019-01-07T11:36:07Z",  
  "status": "Succeeded",  
  "locale": "en-US",  
  "displayName": "Transcription using locale en-US",  
  "links": {  
    "files":
```

```
"https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions  
/9891c965-bb32-4880-b14b-6d44efb158f3/files"  
}  
}
```

Depending on your code's implementation, it may not be enough to rename the property. We recommend using the returned `self` and `links` values as the target urls of your REST calls, rather than generating paths in your client. By using the returned URLs, you can be sure that future changes in paths will not break your client code.

Working with collections of entities

Previously the v2 API returned all available entities in a result. To allow a more fine grained control over the expected response size in v3, all collection results are paginated. You have control over the count of returned entities and the starting offset of the page. This behavior makes it easy to predict the runtime of the response processor.

The basic shape of the response is the same for all collections:

JSON

```
{  
  "values": [  
    {  
    }  
  ],  
  "@nextLink":  
  "https://'{region}'.api.cognitive.microsoft.com/speechtotext/v3.0/{collection}  
  ?skip=100&top=100"  
}
```

The `values` property contains a subset of the available collection entities. The count and offset can be controlled using the `skip` and `top` query parameters. When `@nextLink` is not `null`, there's more data available and the next batch of data can be retrieved by doing a GET on `$.@nextLink`.

This change requires calling the `GET` for the collection in a loop until all elements have been returned.

ⓘ Important

When the response of a GET to `speechtotext/v3.1/{collection}` contains a value in `$.@nextLink`, continue issuing GETs on `$.@nextLink` until `$.@nextLink` is not set to retrieve all elements of that collection.

Creating transcriptions

A detailed description on how to create batches of transcriptions can be found in [Batch transcription How-to](#).

The v3 transcription API lets you set specific transcription options explicitly. All (optional) configuration properties can now be set in the `properties` property. Version v3 also supports multiple input files, so it requires a list of URLs rather than a single URL as v2 did. The v2 property name `recordingsUrl` is now `contentUrls` in v3. The functionality of analyzing sentiment in transcriptions has been removed in v3. See Microsoft Cognitive Service [Text Analysis](#) for sentiment analysis options.

The new property `timeToLive` under `properties` can help prune the existing completed entities. The `timeToLive` specifies a duration after which a completed entity will be deleted automatically. Set it to a high value (for example `PT12H`) when the entities are continuously tracked, consumed, and deleted and therefore usually processed long before 12 hours have passed.

v2 transcription POST request body:

JSON

```
{  
  "locale": "en-US",  
  "name": "Transcription using locale en-US",  
  "recordingsUrl": "https://contoso.com/mystoragelocation",  
  "properties": {  
    "AddDiarization": "False",  
    "AddWordLevelTimestamps": "False",  
    "PunctuationMode": "DictatedAndAutomatic",  
    "ProfanityFilterMode": "Masked"  
  }  
}
```

v3 transcription POST request body:

JSON

```
{  
  "locale": "en-US",  
  "displayName": "Transcription using locale en-US",  
  "contentUrls": [  
    "https://contoso.com/mystoragelocation",  
    "https://contoso.com/myotherstoragelocation"  
  ],
```

```
"properties": {  
    "diarizationEnabled": false,  
    "wordLevelTimestampsEnabled": false,  
    "punctuationMode": "DictatedAndAutomatic",  
    "profanityFilterMode": "Masked"  
}  
}
```

ⓘ Important

Rename the property `recordingsUrl` to `contentUrls` and pass an array of urls instead of a single url. Pass settings for `diarizationEnabled` or `wordLevelTimestampsEnabled` as `bool` instead of `string`.

Format of v3 transcription results

The schema of transcription results has changed slightly to align with transcriptions created by real-time endpoints. Find an in-depth description of the new format in the [Batch transcription How-to](#). The schema of the result is published in our [GitHub sample repository](#) under `samples/batch/transcriptionresult_v3.schema.json`.

Property names are now camel-cased and the values for `channel` and `speaker` now use integer types. Format for durations now use the structure described in ISO 8601, which matches duration formatting used in other Azure APIs.

Sample of a v3 transcription result. The differences are described in the comments.

JSON

```
{  
    "source": "...", // (new in v3) was AudioFileName /  
    AudioFileUrl  
    "timestamp": "2020-06-16T09:30:21Z", // (new in v3)  
    "durationInTicks": 41200000, // (new in v3) was  
    AudioLengthInSeconds  
    "duration": "PT4.12S", // (new in v3)  
    "combinedRecognizedPhrases": [ // (new in v3) was CombinedResults  
        {  
            "channel": 0, // (new in v3) was ChannelNumber  
            "lexical": "hello world",  
            "itn": "hello world",  
            "maskedITN": "hello world",  
            "display": "Hello world."  
        }  
    ],  
    "recognizedPhrases": [ // (new in v3) was SegmentResults
```

```

{
    "recognitionStatus": "Success",      //
    "channel": 0,                      // (new in v3) was ChannelNumber
    "offset": "PT0.07S",                // (new in v3) new format, was
OffsetInSeconds
    "duration": "PT1.59S",              // (new in v3) new format, was
DurationInSeconds
    "offsetInTicks": 700000.0,          // (new in v3) was Offset
    "durationInTicks": 15900000.0,     // (new in v3) was Duration

    // possible transcriptions of the current phrase with confidences
    "nBest": [
        {
            "confidence": 0.898652852, phrase
            "speaker": 1,
            "lexical": "hello world",
            "itn": "hello world",
            "maskedITN": "hello world",
            "display": "Hello world.",
        }

        "words": [
            {
                "word": "hello",
                "offset": "PT0.09S",
                "duration": "PT0.48S",
                "offsetInTicks": 900000.0,
                "durationInTicks": 4800000.0,
                "confidence": 0.987572
            },
            {
                "word": "world",
                "offset": "PT0.59S",
                "duration": "PT0.16S",
                "offsetInTicks": 5900000.0,
                "durationInTicks": 1600000.0,
                "confidence": 0.906032
            }
        ]
    }
}

```

ⓘ Important

Deserialize the transcription result into the new type as shown above. Instead of a single file per audio channel, distinguish channels by checking the property value of `channel` for each element in `recognizedPhrases`. There is now a single result file for each input file.

Getting the content of entities and the results

In v2, the links to the input or result files have been inlined with the rest of the entity metadata. As an improvement in v3, there is a clear separation between entity metadata (which is returned by a GET on `$.self`) and the details and credentials to access the result files. This separation helps protect customer data and allows fine control over the duration of validity of the credentials.

In v3, `links` include a sub-property called `files` in case the entity exposes data (datasets, transcriptions, endpoints, or evaluations). A GET on `$.links.files` will return a list of files and a SAS URL to access the content of each file. To control the validity duration of the SAS URLs, the query parameter `sasValidityInSeconds` can be used to specify the lifetime.

v2 transcription:

```
JSON

{
  "id": "9891c965-bb32-4880-b14b-6d44efb158f3",
  "status": "Succeeded",
  "reportFileUrl": "https://contoso.com/report.txt?st=2018-02-09T18%3A07%3A00Z&se=2018-02-10T18%3A07%3A00Z&sp=rl&sv=2017-04-17&sr=b&sig=6c044930-3926-4be4-be76-f728327c53b5",
  "resultsUrls": {
    "channel_0": "https://contoso.com/audiofile1.wav?st=2018-02-09T18%3A07%3A00Z&se=2018-02-10T18%3A07%3A00Z&sp=rl&sv=2017-04-17&sr=b&sig=6c044930-3926-4be4-be76-f72832e6600c",
    "channel_1": "https://contoso.com/audiofile2.wav?st=2018-02-09T18%3A07%3A00Z&se=2018-02-10T18%3A07%3A00Z&sp=rl&sv=2017-04-17&sr=b&sig=3e0163f1-0029-4d4a-988d-3fba7d7c53b5"
  }
}
```

v3 transcription:

```
JSON

{
  "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-b14b-6d44efb158f3",
  "links": {
    "files": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-b14b-6d44efb158f3/files"
  }
}
```

A GET on `$.links.files` would result in:

JSON

```
{  
  "values": [  
    {  
      "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-b14b-6d44efb158f3/files/f23e54f5-ed74-4c31-9730-2f1a3ef83ce8",  
      "name": "Name",  
      "kind": "Transcription",  
      "properties": {  
        "size": 200  
      },  
      "createdDateTime": "2020-01-13T08:00:00Z",  
      "links": {  
        "contentUrl": "https://customspeech-usw.blob.core.windows.net/artifacts/mywavefile1.wav.json?st=2018-02-09T18%3A07%3A00Z&se=2018-02-10T18%3A07%3A00Z&sp=rl&sv=2017-04-17&sr=b&sig=e05d8d56-9675-448b-820c-4318ae64c8d5"  
      }  
    },  
    {  
      "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-b14b-6d44efb158f3/files/28bc946b-c251-4a86-84f6-ea0f0a2373ef",  
      "name": "Name",  
      "kind": "TranscriptionReport",  
      "properties": {  
        "size": 200  
      },  
      "createdDateTime": "2020-01-13T08:00:00Z",  
      "links": {  
        "contentUrl": "https://customspeech-usw.blob.core.windows.net/artifacts/report.json?st=2018-02-09T18%3A07%3A00Z&se=2018-02-10T18%3A07%3A00Z&sp=rl&sv=2017-04-17&sr=b&sig=e05d8d56-9675-448b-820c-4318ae64c8d5"  
      }  
    }  
  ],  
  "@nextLink": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-b14b-6d44efb158f3/files?skip=2&top=2"  
}
```

The `kind` property indicates the format of content of the file. For transcriptions, the files of kind `TranscriptionReport` are the summary of the job and files of the kind `Transcription` are the result of the job itself.

ⓘ Important

To get the results of operations, use a `GET` on `/speechtotext/v3.0/{collection}/{id}/files`, they are no longer contained in the responses of `GET` on `/speechtotext/v3.0/{collection}/{id}` or `/speechtotext/v3.0/{collection}`.

Customizing models

Before v3, there was a distinction between an *acoustic model* and a *language model* when a model was being trained. This distinction resulted in the need to specify multiple models when creating endpoints or transcriptions. To simplify this process for a caller, we removed the differences and made everything depend on the content of the datasets that are being used for model training. With this change, the model creation now supports mixed datasets (language data and acoustic data). Endpoints and transcriptions now require only one model.

With this change, the need for a `kind` in the `POST` operation has been removed and the `datasets[]` array can now contain multiple datasets of the same or mixed kinds.

To improve the results of a trained model, the acoustic data is automatically used internally during language training. In general, models created through the v3 API deliver more accurate results than models created with the v2 API.

ⓘ Important

To customize both the acoustic and language model part, pass all of the required language and acoustic datasets in `datasets[]` of the POST to `/speechtotext/v3.0/models`. This will create a single model with both parts customized.

Retrieving base and custom models

To simplify getting the available models, v3 has separated the collections of "base models" from the customer owned "customized models". The two routes are now `GET /speechtotext/v3.0/models/base` and `GET /speechtotext/v3.0/models/`.

In v2, all models were returned together in a single response.

ⓘ Important

To get a list of provided base models for customization, use `GET` on `/speechtotext/v3.0/models/base`. You can find your own customized models with a `GET` on `/speechtotext/v3.0/models`.

Name of an entity

The `name` property is now `displayName`. This is consistent with other Azure APIs to not indicate identity properties. The value of this property must not be unique and can be changed after entity creation with a `PATCH` operation.

v2 transcription:

JSON

```
{  
    "name": "Transcription using locale en-US"  
}
```

v3 transcription:

JSON

```
{  
    "displayName": "Transcription using locale en-US"  
}
```

ⓘ Important

Rename the property `name` to `displayName` in your client code.

Accessing referenced entities

In v2, referenced entities were always inlined, for example the used models of an endpoint. The nesting of entities resulted in large responses and consumers rarely consumed the nested content. To shrink the response size and improve performance, the referenced entities are no longer inlined in the response. Instead, a reference to the other entity appears, and can directly be used for a subsequent `GET` (it's a URL as well), following the same pattern as the `self` link.

v2 transcription:

```
JSON

{
  "id": "9891c965-bb32-4880-b14b-6d44efb158f3",
  "models": [
    {
      "id": "827712a5-f942-4997-91c3-7c6cde35600b",
      "modelKind": "Language",
      "lastActionDateTime": "2019-01-07T11:36:07Z",
      "status": "Running",
      "createdDateTime": "2019-01-07T11:34:12Z",
      "locale": "en-US",
      "name": "Acoustic model",
      "description": "Example for an acoustic model",
      "datasets": [
        {
          "id": "702d913a-8ba6-4f66-ad5c-897400b081fb",
          "dataImportKind": "Language",
          "lastActionDateTime": "2019-01-07T11:36:07Z",
          "status": "Succeeded",
          "createdDateTime": "2019-01-07T11:34:12Z",
          "locale": "en-US",
          "name": "Language dataset",
        }
      ]
    },
  ],
}
```

v3 transcription:

```
JSON

{
  "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-b14b-6d44efb158f3",
  "model": {
    "self": "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/models/021a72d0-54c4-43d3-8254-27336ead9037"
  }
}
```

If you need to consume the details of a referenced model as shown in the above example, just issue a GET on `$.model.self`.

 **Important**

To retrieve the metadata of referenced entities, issue a GET on `$.{referencedEntity}.self`, for example to retrieve the model of a transcription do a GET on `$.model.self`.

Retrieving endpoint logs

Version v2 of the service supported logging endpoint results. To retrieve the results of an endpoint with v2, you would create a "data export", which represented a snapshot of the results defined by a time range. The process of exporting batches of data was inflexible. The v3 API gives access to each individual file and allows iteration through them.

A successfully running v3 endpoint:

JSON

```
{  
  "self":  
    "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/afa0  
669c-a01e-4693-ae3a-93baf40f26d6",  
  "links": {  
    "logs":  
      "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/afa0  
669c-a01e-4693-ae3a-93baf40f26d6/files/logs"  
  }  
}
```

Response of GET `$.links.logs`:

JSON

```
{  
  "values": [  
    {  
      "self":  
        "https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/6d72  
ad7e-f286-4a6f-b81b-a0532ca6bcaa/files/logs/2019-09-20_080000_3b5f4628-e225-  
439d-bd27-8804f9eed13f.wav",  
      "name": "2019-09-20_080000_3b5f4628-e225-439d-bd27-8804f9eed13f.wav",  
      "kind": "Audio",  
      "properties": {  
        "size": 12345  
      },  
      "createdDateTime": "2020-01-13T08:00:00Z",  
      "links": {  
        "contentUrl": "https://customspeech-  
usw.blob.core.windows.net/artifacts/2019-09-20_080000_3b5f4628-e225-439d-  
439d-bd27-8804f9eed13f.wav"  
      }  
    }  
  ]  
}
```

```
bd27-8804f9eed13f.wav?st=2018-02-09T18%3A07%3A00Z&se=2018-02-  
10T18%3A07%3A00Z&sp=r&sv=2017-04-17&sr=b&sig=e05d8d56-9675-448b-820c-  
4318ae64c8d5"  
    }  
}  
],  
"@nextLink":  
"https://eastus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/afa0  
669c-a01e-4693-ae3a-93baf40f26d6/files/logs?  
top=2&SkipToken=2!188!MDAwMDk1ITZhMjhiMDl1LTg0MDYtNDViMi1hMGRkLWF1NzR1OGRhZW  
JkNi8yMDIwLTA0LTAxLzEyNDY0M182MzI5NGRkMi1mZGYzLTRhZmEtOTA0NC1mODU5ZTcxOWJiYz  
Yud2F2ITAwMDAyOCE50Tk5LTEyLTMxVDIzOjU50jk50Tk50TlaIQ--"  
}
```

Pagination for endpoint logs works similar to all other collections, except that no offset can be specified. Due to the large amount of available data, pagination is determined by the server.

In v3, each endpoint log can be deleted individually by issuing a `DELETE` operation on the `self` of a file, or by using `DELETE` on `$.links.logs`. To specify an end date, the query parameter `endDate` can be added to the request.

ⓘ Important

Instead of creating log exports on `/api/speechtotext/v2.0/endpoints/{id}/data` use `/v3.0/endpoints/{id}/files/logs/` to access log files individually.

Using custom properties

To separate custom properties from the optional configuration properties, all explicitly named properties are now located in the `properties` property and all properties defined by the callers are now located in the `customProperties` property.

v2 transcription entity:

JSON

```
{  
  "properties": {  
    "customerDefinedKey": "value",  
    "diarizationEnabled": "False",  
    "wordLevelTimestampsEnabled": "False"  
  }  
}
```

v3 transcription entity:

JSON

```
{  
  "properties": {  
    "diarizationEnabled": false,  
    "wordLevelTimestampsEnabled": false  
  },  
  "customProperties": {  
    "customerDefinedKey": "value"  
  }  
}
```

This change also lets you use correct types on all explicitly named properties under `properties` (for example boolean instead of string).

ⓘ Important

Pass all custom properties as `customProperties` instead of `properties` in your `POST` requests.

Response headers

v3 no longer returns the `Operation-Location` header in addition to the `Location` header on `POST` requests. The value of both headers in v2 was the same. Now only `Location` is returned.

Because the new API version is now managed by Azure API management (APIM), the throttling related headers `X-RateLimit-Limit`, `X-RateLimit-Remaining`, and `X-RateLimit-Reset` aren't contained in the response headers.

ⓘ Important

Read the location from response header `Location` instead of `Operation-Location`. In case of a 429 response code, read the `Retry-After` header value instead of `X-RateLimit-Limit`, `X-RateLimit-Remaining`, or `X-RateLimit-Reset`.

Accuracy tests

Accuracy tests have been renamed to evaluations because the new name describes better what they represent. The new paths are:

<https://{}api.cognitive.microsoft.com/speechtotext/v3.0/evaluations>.

 **Important**

Rename the path segment `accuracytests` to `evaluations` in your client code.

Next steps

- [Speech-to-text REST API](#)
- [Speech-to-text REST API v3.0 reference](#) ↗

Migrate code from Long Audio API to Batch synthesis API

Article • 01/11/2023 • 2 minutes to read

The [Batch synthesis API](#) (Preview) provides asynchronous synthesis of long-form text to speech. Benefits of upgrading from Long Audio API to Batch synthesis API, and details about how to do so, are described in the sections below.

ⓘ Important

Batch synthesis API is currently in public preview. Once it's generally available, the Long Audio API will be deprecated.

Base path

You must update the base path in your code from

`/texttospeech/v3.0/longaudiosynthesis` to `/texttospeech/3.1-preview1/batchsynthesis`.

For example, to list synthesis jobs for your Speech resource in the `eastus` region, use

`https://eastus.customvoice.api.speech.microsoft.com/api/texttospeech/3.1-preview1/batchsynthesis` instead of

`https://eastus.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/longaudio/synthesis`.

Regions and endpoints

Batch synthesis API is available in all [Speech regions](#).

The Long Audio API is limited to the following regions:

Region	Endpoint
Australia East	<code>https://australiaeast.customvoice.api.speech.microsoft.com</code>
East US	<code>https://eastus.customvoice.api.speech.microsoft.com</code>
India Central	<code>https://centralindia.customvoice.api.speech.microsoft.com</code>
South Central US	<code>https://southcentralus.customvoice.api.speech.microsoft.com</code>
Southeast Asia	<code>https://southeastasia.customvoice.api.speech.microsoft.com</code>

Region	Endpoint
UK South	https://uksouth.customvoice.api.speech.microsoft.com
West Europe	https://westeurope.customvoice.api.speech.microsoft.com

Voices list

Batch synthesis API supports all [text-to-speech voices and styles](#).

The Long Audio API is limited to the set of voices returned by a GET request to

<https://<endpoint>/api/texttospeech/v3.0/longaudiosynthesis/voices>.

Text inputs

Batch synthesis text inputs are sent in a JSON payload of up to 500 kilobytes.

Long Audio API text inputs are uploaded from a file that meets the following requirements:

- One plain text (.txt) or SSML text (.txt) file encoded as [UTF-8 with Byte Order Mark \(BOM\)](#). Don't use compressed files such as ZIP. If you have more than one input file, you must submit multiple requests.
- 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 a new line. For SSML text, each SSML piece is considered a paragraph. Separate SSML pieces by different paragraphs.

With Batch synthesis API, you can use any of the [supported SSML elements](#), including the `audio`, `mstts:backgroundaudio`, and `lexicon` elements. The `audio`, `mstts:backgroundaudio`, and `lexicon` elements aren't supported by Long Audio API.

Audio output formats

Batch synthesis API supports all [text-to-speech audio output formats](#).

The Long Audio API is limited to the following set of audio output formats. The sample rate for long audio voices is 24kHz, not 48kHz. Other sample rates can be obtained through upsampling or downsampling when synthesizing.

- 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

Getting results

With batch synthesis API, use the URL from the `outputs.result` property of the GET batch synthesis response. The `results` are in a ZIP file that contains the audio (such as `0001.wav`), summary, and debug details.

Long Audio API text inputs and results are returned via two separate content URLs as shown in the following example. The one with `"kind": "LongAudioSynthesisScript"` is the input script submitted. The other one with `"kind": "LongAudioSynthesisResult"` is the result of this request. Both ZIP files can be downloaded from the URL in their `links.contentUrl` property.

Cleaning up resources

Batch synthesis API supports up to 200 batch synthesis jobs that don't have a status of "Succeeded" or "Failed". The Speech service will keep each synthesis history for up to 31 days, or the duration of the request `timeToLive` property, whichever comes sooner. The date and time of automatic deletion (for synthesis jobs with a status of "Succeeded" or "Failed") is equal to the `lastActionDateTime` + `timeToLive` properties.

The Long Audio API is limited to 20,000 requests for each Azure subscription account. The Speech service doesn't remove job history automatically. You must remove the previous job run history before making new requests that would otherwise exceed the limit.

Next steps

- [Batch synthesis API](#)
- [Speech Synthesis Markup Language \(SSML\)](#)
- [Text-to-speech quickstart](#)

What are Speech devices?

Article • 02/20/2022 • 2 minutes to read

The [Speech service](#) works with a wide variety of devices and audio sources. You can use the default audio processing available on a device. Otherwise, the Speech SDK has an option for you to use our advanced audio processing algorithms that are designed to work well with the [Speech service](#). It provides accurate far-field [speech recognition](#) via noise suppression, echo cancellation, beamforming, and dereverberation.

Audio processing

Audio processing is enhancements applied to a stream of audio to improve the audio quality. Examples of common enhancements include automatic gain control (AGC), noise suppression, and acoustic echo cancellation (AEC). The Speech SDK integrates [Microsoft Audio Stack \(MAS\)](#), allowing any application or product to use its audio processing capabilities on input audio.

Microphone array recommendations

The Speech SDK works best with a microphone array that has been designed according to our recommended guidelines. For details, see [Microphone array recommendations](#).

Device development kits

The Speech SDK is designed to work with purpose-built development kits, and varying microphone array configurations. For example, you can use one of these Azure development kits.

- [Azure Percept DK](#) contains a preconfigured audio processor and a four-microphone linear array. You can use voice commands, keyword spotting, and far-field speech with the help of Azure Cognitive Services.
- [Azure Kinect DK](#) is a spatial computing developer kit with advanced AI sensors that provide sophisticated computer vision and speech models. As an all-in-one small device with multiple modes, it contains a depth sensor, spatial microphone array with a video camera, and orientation sensor.

Next steps

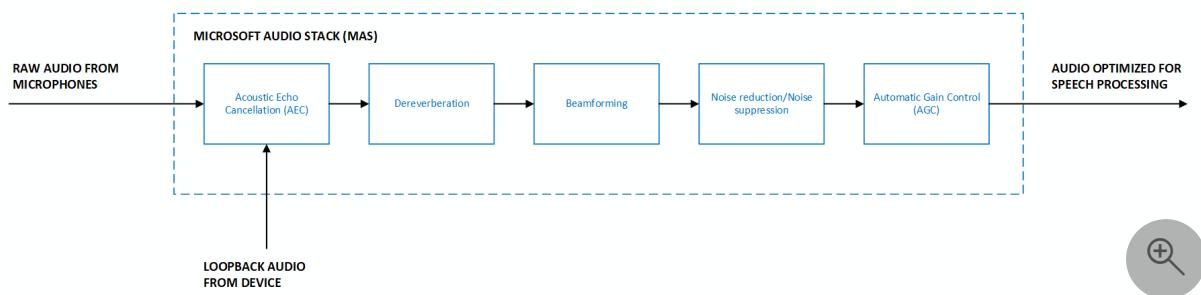
- [Audio processing concepts](#)

Audio processing

Article • 12/01/2022 • 3 minutes to read

The Microsoft Audio Stack is a set of enhancements optimized for speech processing scenarios. This includes examples like keyword recognition and speech recognition. It consists of various enhancements/components that operate on the input audio signal:

- **Noise suppression** - Reduce the level of background noise.
- **Beamforming** - Localize the origin of sound and optimize the audio signal using multiple microphones.
- **Dereverberation** - Reduce the reflections of sound from surfaces in the environment.
- **Acoustic echo cancellation** - Suppress audio being played out of the device while microphone input is active.
- **Automatic gain control** - Dynamically adjust the person's voice level to account for soft speakers, long distances, or non-calibrated microphones.



Different scenarios and use-cases can require different optimizations that influence the behavior of the audio processing stack. For example, in telecommunications scenarios such as telephone calls, it is acceptable to have minor distortions in the audio signal after processing has been applied. This is because humans can continue to understand the speech with high accuracy. However, it is unacceptable and disruptive for a person to hear their own voice in an echo. This contrasts with speech processing scenarios, where distorted audio can adversely impact a machine-learned speech recognition model's accuracy, but it is acceptable to have minor levels of echo residual.

Processing is performed fully locally where the Speech SDK is being used. No audio data is streamed to Microsoft's cloud services for processing by the Microsoft Audio Stack. The only exception to this is for the Conversation Transcription Service, where raw audio is sent to Microsoft's cloud services for processing.

The Microsoft Audio Stack also powers a wide range of Microsoft products:

- **Windows** - Microsoft Audio Stack is the default speech processing pipeline when using the Speech audio category.
- **Microsoft Teams Displays and Microsoft Teams Room devices** - Microsoft Teams Displays and Teams Room devices use the Microsoft Audio Stack to enable high quality hands-free, voice-based experiences with Cortana.

Speech SDK integration

The Speech SDK integrates Microsoft Audio Stack (MAS), allowing any application or product to use its audio processing capabilities on input audio. Some of the key Microsoft Audio Stack features available via the Speech SDK include:

- **Realtime microphone input & file input** - Microsoft Audio Stack processing can be applied to real-time microphone input, streams, and file-based input.
- **Selection of enhancements** - To allow for full control of your scenario, the SDK allows you to disable individual enhancements like dereverberation, noise suppression, automatic gain control, and acoustic echo cancellation. For example, if your scenario does not include rendering output audio that needs to be suppressed from the input audio, you have the option to disable acoustic echo cancellation.
- **Custom microphone geometries** - The SDK allows you to provide your own custom microphone geometry information, in addition to supporting preset geometries like linear two-mic, linear four-mic, and circular 7-mic arrays (see more information on supported preset geometries at [Microphone array recommendations](#)).
- **Beamforming angles** - Specific beamforming angles can be provided to optimize audio input originating from a predetermined location, relative to the microphones.

Minimum requirements to use Microsoft Audio Stack

Microsoft Audio Stack can be used by any product or application that can meet the following requirements:

- **Raw audio** - Microsoft Audio Stack requires raw (unprocessed) audio as input to yield the best results. Providing audio that is already processed limits the audio stack's ability to perform enhancements at high quality.
- **Microphone geometries** - Geometry information about each microphone on the device is required to correctly perform all enhancements offered by the Microsoft

Audio Stack. Information includes the number of microphones, their physical arrangement, and coordinates. Up to 16 input microphone channels are supported.

- **Loopback or reference audio** - An audio channel that represents the audio being played out of the device is required to perform acoustic echo cancellation.
- **Input format** - Microsoft Audio Stack supports down sampling for sample rates that are integral multiples of 16 kHz. A minimum sampling rate of 16 kHz is required. Additionally, the following formats are supported: 32-bit IEEE little endian float, 32-bit little endian signed int, 24-bit little endian signed int, 16-bit little endian signed int, and 8-bit signed int.

Next steps

[Use the Speech SDK for audio processing](#)

Use the Microsoft Audio Stack (MAS)

Article • 09/20/2022 • 5 minutes to read

The Speech SDK integrates Microsoft Audio Stack (MAS), allowing any application or product to use its audio processing capabilities on input audio. See the [Audio processing](#) documentation for an overview.

In this article, you learn how to use the Microsoft Audio Stack (MAS) with the Speech SDK.

Default options

This sample shows how to use MAS with all default enhancement options on input from the device's default microphone.

C#

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

var audioProcessingOptions =
    AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCESSING_ENABLE_DEFAULT);
var audioInput =
    AudioConfig.FromDefaultMicrophoneInput(audioProcessingOptions);

var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Preset microphone geometry

This sample shows how to use MAS with a predefined microphone geometry on a specified audio input device. In this example:

- **Enhancement options** - The default enhancements are applied on the input audio stream.
- **Preset geometry** - The preset geometry represents a linear 2-microphone array.
- **Audio input device** - The audio input device ID is `hw:0,1`. For more information on how to select an audio input device, see [How to: Select an audio input device with the Speech SDK](#).

C#

```
C#  
  
var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey",  
    "YourServiceRegion");  
  
var audioProcessingOptions =  
    AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCE  
SSING_ENABLE_DEFAULT, PresetMicrophoneArrayGeometry.Linear2);  
var audioInput = AudioConfig.FromMicrophoneInput("hw:0,1",  
    audioProcessingOptions);  
  
var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Custom microphone geometry

This sample shows how to use MAS with a custom microphone geometry on a specified audio input device. In this example:

- **Enhancement options** - The default enhancements will be applied on the input audio stream.
- **Custom geometry** - A custom microphone geometry for a 7-microphone array is provided via the microphone coordinates. The units for coordinates are millimeters.
- **Audio input** - The audio input is from a file, where the audio within the file is expected from an audio input device corresponding to the custom geometry specified.

C#

```
C#  
  
var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey",  
    "YourServiceRegion");  
  
MicrophoneCoordinates[] microphoneCoordinates = new  
MicrophoneCoordinates[7]  
{  
    new MicrophoneCoordinates(0, 0, 0),  
    new MicrophoneCoordinates(40, 0, 0),  
    new MicrophoneCoordinates(20, -35, 0),  
    new MicrophoneCoordinates(-20, -35, 0),  
    new MicrophoneCoordinates(-40, 0, 0),  
    new MicrophoneCoordinates(-20, 35, 0),  
    new MicrophoneCoordinates(0, 35, 0),  
};
```

```
    new MicrophoneCoordinates(20, 35, 0)
};

var microphoneArrayGeometry = new
MicrophoneArrayGeometry(MicrophoneArrayType.Planar,
microphoneCoordinates);
var audioProcessingOptions =
AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCE
SSING_ENABLE_DEFAULT, microphoneArrayGeometry,
SpeakerReferenceChannel.LastChannel);
var audioInput = AudioConfig.FromWavFileInput("katiesteve.wav",
audioProcessingOptions);

var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Select enhancements

This sample shows how to use MAS with a custom set of enhancements on the input audio. By default, all enhancements are enabled but there are options to disable dereverberation, noise suppression, automatic gain control, and echo cancellation individually by using `AudioProcessingOptions`.

In this example:

- **Enhancement options** - Echo cancellation and noise suppression will be disabled, while all other enhancements remain enabled.
- **Audio input device** - The audio input device is the default microphone of the device.

C#

```
C#  
  
var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey",
"YourServiceRegion");  
  
var audioProcessingOptions =
AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCE
SSING_DISABLE_ECHO_CANCELLATION |
AudioProcessingConstants.AUDIO_INPUT_PROCESSING_DISABLE_NOISE_SUPPRESSIO
N | AudioProcessingConstants.AUDIO_INPUT_PROCESSING_ENABLE_DEFAULT);
var audioInput =
AudioConfig.FromDefaultMicrophoneInput(audioProcessingOptions);  
  
var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Specify beamforming angles

This sample shows how to use MAS with a custom microphone geometry and beamforming angles on a specified audio input device. In this example:

- **Enhancement options** - The default enhancements will be applied on the input audio stream.
- **Custom geometry** - A custom microphone geometry for a 4-microphone array is provided by specifying the microphone coordinates. The units for coordinates are millimeters.
- **Beamforming angles** - Beamforming angles are specified to optimize for audio originating in that range. The units for angles are degrees. In the sample code below, the start angle is set to 70 degrees and the end angle is set to 110 degrees.
- **Audio input** - The audio input is from a push stream, where the audio within the stream is expected from an audio input device corresponding to the custom geometry specified.

C#

```
C#  
  
var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey",  
"YourServiceRegion");  
  
MicrophoneCoordinates[] microphoneCoordinates = new  
MicrophoneCoordinates[4]  
{  
    new MicrophoneCoordinates(-60, 0, 0),  
    new MicrophoneCoordinates(-20, 0, 0),  
    new MicrophoneCoordinates(20, 0, 0),  
    new MicrophoneCoordinates(60, 0, 0)  
};  
var microphoneArrayGeometry = new  
MicrophoneArrayGeometry(MicrophoneArrayType.Linear, 70, 110,  
microphoneCoordinates);  
var audioProcessingOptions =  
AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCE  
SSING_ENABLE_DEFAULT, microphoneArrayGeometry,  
SpeakerReferenceChannel.LastChannel);  
var pushStream = AudioInputStream.CreatePushStream();  
var audioInput = AudioConfig.FromStreamInput(pushStream,  
audioProcessingOptions);  
  
var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Reference channel for echo cancellation

Microsoft Audio Stack requires the reference channel (also known as loopback channel) to perform echo cancellation. The source of the reference channel varies by platform:

- **Windows** - The reference channel is automatically gathered by the Speech SDK if the `SpeakerReferenceChannel::LastChannel` option is provided when creating `AudioProcessingOptions`.
- **Linux** - ALSA (Advanced Linux Sound Architecture) must be configured to provide the reference audio stream as the last channel for the audio input device used. ALSA is configured in addition to providing the `SpeakerReferenceChannel::LastChannel` option when creating `AudioProcessingOptions`.

Language and platform support

Language	Platform	Reference docs
C++	Windows, Linux	C++ docs
C#	Windows, Linux	C# docs
Java	Windows, Linux	Java docs

Next steps

[Setup development environment](#)

Microphone array recommendations

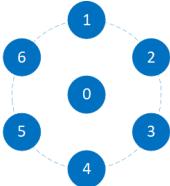
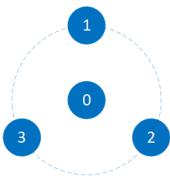
Article • 01/09/2022 • 4 minutes to read

In this article, you learn how to design a microphone array customized for use with the Speech SDK. This is most pertinent if you are selecting, specifying, or building hardware for speech solutions.

The Speech SDK works best with a microphone array that has been designed according to these guidelines, including the microphone geometry, component selection, and architecture.

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.

Array	Microphones	Geometry
Circular - 7 Microphones		6 Outer, 1 Center, Radius = 42.5 mm, Evenly Spaced
Circular - 4 Microphones		3 Outer, 1 Center, Radius = 42.5 mm, Evenly Spaced
Linear - 4 Microphones		Length = 120 mm, Spacing = 40 mm

Array	Microphones	Geometry
Linear - 2 Microphones		Spacing = 40 mm



Microphone channels should be ordered ascending from 0, according to the numbering depicted above for each array. 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
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 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 dBSPL, A-weighted noise)
Output Sensitivity	-26 dBFS/Pa @ 1 kHz (recommended)
Amplitude Matching	± 2 dB, 200-8000 Hz
THD%*	≤ 1%, 200-8000 Hz, 94 dBSPL, 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
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 $TCLw = TCLw_{measured} + (Measured\ Level - Target\ Output\ Sensitivity)$ $TCLw = TCLw_{measured} + (Measured\ Level - (-26))$

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. 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 USB Audio Devices Rev3 Spec
Microphone Geometry	Drivers must implement Microphone Array Geometry Descriptors 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 a SoC that runs the [Microsoft Audio Stack \(MAS\)](#)) 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

Release notes: Speech Devices SDK (retired)

Article • 10/20/2022 • 3 minutes to read

The following sections list changes in the most recent releases.

ⓘ Note

The Speech Devices SDK is deprecated. Archived sample code is available on [GitHub](#). All users of the Speech Devices SDK are advised to migrate to using Speech SDK version 1.19.0 or later.

Speech Devices SDK 1.16.0:

- Fixed [GitHub issue #22](#).
- Updated the [Speech SDK](#) component to version 1.16.0. For more information, see its [release notes](#).

Speech Devices SDK 1.15.0:

- Upgraded to new Microsoft Audio Stack (MAS) with improved beamforming and noise reduction for speech.
- Reduced the binary size by as much as 70% depending on target.
- Support for [Azure Percept Audio](#) with [binary release](#).
- Updated the [Speech SDK](#) component to version 1.15.0. For more information, see its [release notes](#).

Speech Devices SDK 1.11.0:

- Support for arbitrary microphone array geometries and setting the working angle through a [configuration file](#).
- Support for [Urbetter DDK](#).
- Released binaries for the [GGEC Speaker](#) used in our [Voice Assistant sample](#).
- Released binaries for [Linux ARM32](#) and [Linux ARM 64](#) for Raspberry Pi and similar devices.
- Updated the [Speech SDK](#) component to version 1.11.0. For more information, see its [release notes](#).

Speech Devices SDK 1.9.0:

- Initial binaries for [Urbetter DDK](#) (Linux ARM64) are provided.
- Roobo v1 now uses Maven for the Speech SDK
- Updated the [Speech SDK](#) component to version 1.9.0. For more information, see its [release notes](#).

Speech Devices SDK 1.7.0:

- Linux ARM is now supported.
- Initial binaries for [Roobo v2 DDK](#) 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 issue where the process would stop responding 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 keyword recognition 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.

Speech Devices SDK 0.2.12733: 2018-May release

The first public preview release of the Cognitive Services Speech Devices SDK.

Summary

Article • 01/31/2022 • 2 minutes to read

Members	Descriptions
namespace Microsoft::CognitiveServices::Speech	
namespace Microsoft::CognitiveServices::Speech::Audio	
namespace Microsoft::CognitiveServices::Speech::Diagnostics::Logging	
namespace Microsoft::CognitiveServices::Speech::Dialog	
namespace Microsoft::CognitiveServices::Speech::Intent	
namespace Microsoft::CognitiveServices::Speech::Speaker	
namespace Microsoft::CognitiveServices::Speech::Transcription	
namespace Microsoft::CognitiveServices::Speech::Translation	
class Transcription::ConversationTranscriber::PrivatePropertyCollection	

Microsoft.CognitiveServices.Speech Namespace

Reference

In this article

[Classes](#)

[Enums](#)

Classes

AudioDataStream	Provides audio data as a stream. Added in 1.4.0
AutoDetectSourceLanguageConfig	Configures options for automatic detection of languages. Updated in 1.13.0
AutoDetectSourceLanguageResult	Contains languages detected by the Speech service. Added in 1.9.0
CancellationDetails	Contains detailed information about why a result was canceled.
ClassLanguageModel	Represents a list of grammars for dynamic grammar scenarios. Added in 1.7.0
Connection	A proxy class for managing connection to the speech service of the specified Recognizer. Added in 1.2.0
ConnectionEventArgs	Contains payload for Connected/Disconnected events Added in 1.2.0
ConnectionMessage	Represents implementation-specific messages sent to and received from the speech service. For debugging only. Added in 1.10.0
ConnectionMessageEventArgs	Contains payload for MessageReceived events of a Connection instance. Added in 1.10.0
DetailedSpeechRecognitionResult	Contains recognition details including confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity. Changed in 1.7.0
EmbeddedSpeechConfig	Class that defines embedded (offline) configurations for speech recognition and speech synthesis.
Grammar	Represents base class grammar for customizing speech recognition. Added in 1.5.0
GrammarList	Represents a list of grammars for dynamic grammar scenarios.

	Added in 1.7.0
GrammarPhrase	Represents a phrase that can be spoken by the user. Added in 1.5.0
HybridSpeechConfig	Class that defines hybrid (cloud and embedded) configurations for speech recognition and speech synthesis.
KeywordRecognitionEvent Args	Class for the events emitted by the KeywordRecognizer .
KeywordRecognitionModel	Represents keyword recognition model that can trigger an event when pre-defined keywords are spoken.
KeywordRecognitionResult	Contains the results emitted by the KeywordRecognizer .
KeywordRecognizer	Recognizes a word or short phrase using a keyword model.
NoMatchDetails	Contains detailed information for NoMatch recognition results.
PhonemeLevelTimingResult	Phoneme level timing result. Added in 1.14.0
PhraseListGrammar	Identifies known phrases in audio data. Added in 1.5.0
PronunciationAssessment NBestPhoneme	Pronunciation assessment nbest phoneme result Added in 1.20.0
PropertyCollection	Class to retrieve or set a property value from a property collection.
RecognitionEventArgs	Contains payload for recognition events like Speech Start/End Detected.
RecognitionResult	Contains detailed information about result of a recognition operation.
Recognizer	Base class that mostly contains common event handlers.
SessionEventArgs	Contains payload for SessionStarted and SessionStopped events.
SourceLanguageConfig	Source Language configuration. Added in 1.17.0
SourceLanguageRecognizer	Detects the source language from audio stream Added in version 1.17.0
SpeechConfig	Information about your subscription, including your key and region, endpoint, host, or authorization token.
SpeechRecognitionCanceled EventArgs	Contains payload of speech recognition canceled result events.
SpeechRecognitionEventArgs	Contains payload of speech recognizing/recognized events.

SpeechRecognitionModel	Speech recognition model information.
SpeechRecognitionResult	Contains result of speech recognition.
SpeechRecognitionResultExtensions	Extension methods for speech recognition result
SpeechRecognizer	Transcribes speech into text. Speech can arrive via microphone, audio file, or other audio input stream.
SpeechSynthesisBookmarkEventArgs	Contains bookmark event in synthesized speech. Added in 1.16.0
SpeechSynthesisCancellationDetails	Contains detailed information about why a speech synthesis result was canceled. Added in 1.4.0
SpeechSynthesisEventArgs	Contains payload of speech synthesis events. Added in 1.4.0
SpeechSynthesisResult	Contains detailed information about result of a speech synthesis operation. Added in 1.4.0
SpeechSynthesisVisemeEventArgs	Contains facial pose events that correspond to time-based offsets in synthesized speech. Added in 1.16.0
SpeechSynthesisWordBoundaryEventArgs	Contains location and length details about words in synthesized speech. Added in 1.7.0
SpeechSynthesizer	Performs speech synthesis to speaker, file, or other audio output streams, and gets synthesized audio as result. Updated in 1.16.0
SpeechTranslationConfig	Speech translation configuration.
SyllableLevelTimingResult	Syllable level timing result. Added in 1.20.0
SynthesisVoicesResult	Contains detailed information about the retrieved synthesis voices list. Added in 1.16.0
VoiceInfo	Contains detailed information about the synthesis voice. Updated in 1.17.0
WordLevelTimingResult	For a recognized word in speech audio, contains the offset to the start and the duration, in ticks. 1 tick = 100 ns. Added in 1.7.0

Enums

CancellationErrorCode	Lists error codes possible when CancellationReason is Error . Added in 1.1.0
CancellationReason	Lists the possible reasons a recognition result might be canceled.
NoMatchReason	Lists the possible reasons a recognition result was not

	recognized.
OutputFormat	Output format.
ProfanityOption	Removes profanity (swearing), or replaces letters of profane words with stars. Added in 1.5.0
PropertyId	Lists speech property ids. Changed in 1.9.0
RecognitionFactorScope	Lists the scope that a recognition factor applies to.
ResultReason	Describes a recognition result.
ServicePropertyChannel	Lists channels used to pass property settings to service. Added in 1.5.0
SpeechSynthesisBoundaryType	Defines the boundary type of speech synthesis boundary event Added in version 1.21.0
SpeechSynthesisOutputFormat	Lists synthesis output audio formats.
StreamStatus	Lists possible status values of an audio data stream. Added in 1.4.0
SynthesisVoiceGender	Lists synthesis voice gender. Added in version 1.17.0
SynthesisVoiceType	Lists synthesis voice types.

com.microsoft.cognitiveservices.speech

Reference

Package: com.microsoft.cognitiveservices.speech

Maven Artifact: [com.microsoft.cognitiveservices.speech:client-sdk:1.24.2](#)

In this article

[Classes](#)

[Enums](#)

Classes

AudioDataStream	Represents audio data stream used for operating audio data as a stream.
AutoDetectSourceLanguageConfig	Represents auto detect source language configuration used for specifying the possible source language candidates Note: close() must be called in order to release underlying resources held by the object.
AutoDetectSourceLanguageResult	Represents the result of auto detecting source languages Added in version 1.8.0
CancellationDetails	Contains detailed information about why a result was canceled.
ClassLanguageModel	Represents a ClassLanguageModel.
Connection	Connection is a proxy class for managing connection to the speech service of the specified Recognizer.
ConnectionEventArgs	Defines payload for connection events like Connected/Disconnected.
ConnectionMessage	ConnectionMessage represents implementation specific messages sent to and received from the speech service.
ConnectionMessageEventArgs	Defines payload for Connection's MessageReceived events.
Diagnostics	Native logging and other diagnostics
EmbeddedSpeechConfig	Class that defines embedded (offline) configurations for speech recognition and speech synthesis.
Grammar	Represents a generic grammar used to assist in improving speech recogniton accuracy.
GrammarList	Allows adding multiple grammars to a SpeechRecognizer to improve the accuracy of speech recognition.

HybridSpeechConfig	Class that defines hybrid (cloud and embedded) configurations for speech recognition and speech synthesis.
KeywordRecognitionEventArgs	Defines content of an keyword recognizing/recognized events.
KeywordRecognitionModel	Represents a keyword recognition model for recognizing when the user says a keyword to initiate further speech recognition.
KeywordRecognitionResult	Defines result of keyword recognition.
KeywordRecognizer	Performs keyword recognition on the speech input.
NoMatchDetails	Contains detailed information for NoMatch recognition results.
PhraseListGrammar	Allows additions of new phrases to improve speech recognition.
PronunciationAssessmentConfig	Represents pronunciation assessment configuration.
PronunciationAssessmentResult	Represents the result of pronunciation assessment.
PropertyCollection	Represents collection of properties and their values.
RecognitionEventArgs	Defines payload for recognition events like Speech Start/End Detected
RecognitionResult	Contains detailed information about result of a recognition operation.
Recognizer	Defines the base class Recognizer which mainly contains common event handlers.
SessionEventArgs	Defines payload for SessionStarted/Stopped events.
SourceLanguageConfig	Represents source language configuration used for specifying recognition source language.
SpeechConfig	Speech configuration.
SpeechRecognitionCanceledEventArgs	Defines payload of speech recognition canceled events.
SpeechRecognitionEventArgs	Defines contents of speech recognizing/recognized event.
SpeechRecognitionModel	Contains detailed speech recognition model information.
SpeechRecognitionResult	Defines result of speech recognition.
SpeechRecognizer	Performs speech recognition from microphone, file, or other audio input streams, and gets transcribed text as result.

SpeechSynthesisBookmarkEventArgs	Defines contents of speech synthesis bookmark event.
SpeechSynthesisCancellationDetails	Contains detailed information about why a speech synthesis was canceled.
SpeechSynthesisEventArgs	Defines contents of speech synthesis related event.
SpeechSynthesisResult	Contains detailed information about result of a speech synthesis operation.
SpeechSynthesisVisemeEventArgs	Defines contents of speech synthesis viseme event.
SpeechSynthesisWordBoundaryEventArgs	Defines contents of speech synthesis word boundary event.
SpeechSynthesizer	Performs speech synthesis to speaker, file, or other audio output streams, and gets synthesized audio as result.
SynthesisVoicesResult	Contains detailed information about the retrieved synthesis voices list.
VoiceInfo	Contains detailed information about the synthesis voice information.

Enums

CancellationErrorCode	Defines error code in case that CancellationReason is Error.
CancellationReason	Defines the possible reasons a recognition result might be canceled.
NoMatchReason	Defines the possible reasons a recognition result might not be recognized.
OutputFormat	Define Speech Recognizer output formats.
ProfanityOption	Define profanity option for response result.
PronunciationAssessmentGradingSystem	Defines the point system for pronunciation score calibration; default value is FivePoint.
PronunciationAssessmentGranularity	Defines the pronunciation evaluation granularity; default value is Phoneme.
PropertyId	Defines property ids.
ResultReason	Defines the possible reasons a recognition result might be generated.

ServicePropertyChannel	Defines channels used to send service properties.
SpeechSynthesisBoundaryType	Defines the boundary type of speech synthesis boundary event.
SpeechSynthesisOutputFormat	Defines the possible speech synthesis output audio format.
StreamStatus	Defines the possible status of audio data stream.
SynthesisVoiceGender	Define synthesis voice gender.
SynthesisVoiceType	Define synthesis voice type.

microsoft-cognitiveservices-speech-sdk package

Reference

In this article

- [Classes](#)
- [Interfaces](#)
- [Type Aliases](#)
- [Enums](#)
- [Functions](#)
- [Function Details](#)

Classes

ConsoleLoggingListener	
File AudioSource	
Mic AudioSource	
Pcm Recorder	
Proxy Info	
Replayable Audio Node	
Rest Config Base	
Rest Message Adapter	
Websocket Connection	
Websocket Message Adapter	
Added Lm Intent	
Agent Config	Represents the JSON used in the agent.config message sent to the speech service.
Cognitive Subscription Key Authentication	
Cognitive Token Authentication	
Connection Factory Base	
Conversation Service Recognizer	

DialogServiceAdapter	
DynamicGrammarBuilder	Responsible for building the object to be sent to the speech service to support dynamic grammars.
EnumTranslation	
AuthInfo	
IntentConnectionFactory	
IntentServiceRecognizer	
ConnectingToServiceEvent	
ListeningStartedEvent	
RecognitionEndedEvent	
RecognitionStartedEvent	
RecognitionTriggeredEvent	
SpeechRecognitionEvent	
Device	
OS	
RecognizerConfig	
SpeechServiceConfig	
System	
RequestSession	
DetailedSpeechPhrase	
IntentResponse	
SimpleSpeechPhrase	
SpeechDetected	
SpeechHypothesis	
SpeechKeyword	
SynthesisAudioMetadata	
TranslationHypothesis	
TranslationPhrase	

TranslationSynthesisEnd	
ServiceRecognizerBase	
SpeakerIdMessageAdapter	Implements methods for speaker recognition classes, sending requests to endpoint and parsing response into expected format
SpeakerRecognitionConfig	
SpeechConnectionFactory	
SpeechConnectionMessage	
SpeechContext	Represents the JSON used in the speech.context message sent to the speech service. The dynamic grammar is always refreshed from the encapsulated dynamic grammar object.
SpeechServiceRecognizer	
SpeechSynthesisConnectionFactory	
SynthesisAdapterBase	
SynthesisContext	Represents the JSON used in the synthesis.context message sent to the speech service. The dynamic grammar is always refreshed from the encapsulated dynamic grammar object.
ConnectingToSynthesisServiceEvent	
SpeechSynthesisEvent	
SynthesisStartedEvent	
SynthesisTriggeredEvent	
SynthesisRestAdapter	Implements methods for speaker recognition classes, sending requests to endpoint and parsing response into expected format
SynthesisTurn	
SynthesizerConfig	
TranscriberConnectionFactory	
ConversationConnectionConfig	
ConversationManager	
ConversationReceivedTranslationEventArgs	
LockRoomEventArgs	
MuteAllEventArgs	

ParticipantAttributeEventArgs	
ParticipantEventArgs	
ParticipantsListEventArgs	
InternalParticipants	Users participating in the conversation
ConversationRecognizerFactory	
ConversationTranslatorRecognizer	Sends messages to the Conversation Translator websocket and listens for incoming events containing websocket messages. Based off the recognizers in the SDK folder.
TranscriberRecognizer	
TranscriptionServiceRecognizer	
TranslationConnectionFactory	
TranslationServiceRecognizer	
WebsocketMessageFormatter	
AudioSourceErrorEvent	
 AudioSourceEvent	
 AudioSourceInitializingEvent	
 AudioSourceOffEvent	
 AudioSourceReadyEvent	
 AudioStreamNodeAttachedEvent	
 AudioStreamNodeAttachingEvent	
 AudioStreamNodeDetachedEvent	
 AudioStreamNodeErrorEvent	
 AudioStreamNodeEvent	
 BackgroundEvent	
 ChunkedArrayBufferStream	
 ConnectionClosedEvent	
 ConnectionErrorEvent	
 ConnectionEstablishErrorEvent	

ConnectionEstablishedEvent	
ConnectionEvent	
ConnectionMessageReceivedEvent	
ConnectionMessageSentEvent	
ConnectionStartEvent	
ServiceEvent	
ConnectionMessage	ConnectionMessage represents implementation specific messages sent to and received from the speech service. These messages are provided for debugging purposes and should not be used for production use cases with the Azure Cognitive Services Speech Service. Messages sent to and received from the Speech Service are subject to change without notice. This includes message contents, headers, payloads, ordering, etc. Added in version 1.11.0.
ConnectionOpenResponse	
DialogEvent	
SendingAgentContextMessageEvent	
ArgumentNullException	The error that is thrown when an argument passed in is null.
InvalidOperationException	The error that is thrown when an invalid operation is performed in the code.
ObjectDisposedError	The error that is thrown when an object is disposed.
EventSource	
Events	
List	
OCSPCacheEntryExpiredEvent	
OCSPCacheEntryNeedsRefreshEvent	
OCSPCacheFetchErrorEvent	
OCSPCacheHitEvent	
OCSPCacheMissEvent	
OCSPCacheUpdateCompleteEvent	
OCSPCacheUpdateErrorEvent	

OCSPCacheUpdateNeededEvent	
OCSPDiskCacheHitEvent	
OCSPDiskCacheStoreEvent	
OCSPEvent	
OCSPMemoryCacheHitEvent	
OCSPMemoryCacheStoreEvent	
OCSPResponseRetrievedEvent	
OCSPStapleReceivedEvent	
OCSPVerificationFailedEvent	
OCSPWSUpgradeStartedEvent	
PlatformEvent	
Deferred	
PromiseResult	
PromiseResultEventSource	
Sink	
Queue	
RawWebsocketMessage	
RiffPcmEncoder	
Stream	
Timeout	
ActivityReceivedEventArgs	Defines contents of received message/events.
AudioConfig	Represents audio input configuration used for specifying what type of input to use (microphone, file, stream).
AudioOutputConfigImpl	
AudioInputStream	Represents audio input stream used for custom audio input configurations.
PullAudioInputStream	
PushAudioInputStream	Represents memory backed push audio input stream used for custom audio input configurations.

AudioOutputStream	Represents audio output stream used for custom audio output configurations.
PullAudioOutputStream	Represents memory backed push audio output stream used for custom audio output configurations.
PushAudioOutputStream	
AudioStreamFormat	Represents audio stream format used for custom audio input configurations.
BaseAudioPlayer	Base audio player class TODO: Plays only PCM for now.
PullAudioInputStreamCallback	An abstract base class that defines callback methods (read() and close()) for custom audio input streams).
PushAudioOutputStream Callback	An abstract base class that defines callback methods (write() and close()) for custom audio output streams).
SpeakerAudioDestination	Represents the speaker playback audio destination, which only works in browser. Note: the SDK will try to use Media Source Extensions to play audio. Mp3 format has better supports on Microsoft Edge, Chrome and Safari (desktop), so, it's better to specify mp3 format for playback.
AutoDetectSourceLanguage Config	Language auto detect configuration.
AutoDetectSourceLanguage Result	Output format
BotFrameworkConfig	Class that defines configurations for the dialog service connector object for using a Bot Framework backend.
CancellationDetails	Contains detailed information about why a result was canceled.
CancellationDetailsBase	Contains detailed information about why a result was canceled.
CancellationEventArgsBase	Defines content of a CancellationEvent.
Connection	Connection is a proxy class for managing connection to the speech service of the specified Recognizer. By default, a Recognizer autonomously manages connection to service when needed. The Connection class provides additional methods for users to explicitly open or close a connection and to subscribe to connection status changes. The use of Connection is optional, and mainly for scenarios where fine tuning of application behavior based on connection status is needed. Users can optionally call Open() to manually set up a connection in advance before starting recognition on the Recognizer associated with this Connection. If the Recognizer needs to connect or disconnect to service, it will setup or shutdown the

	connection independently. In this case the Connection will be notified by change of connection status via Connected/Disconnected events. Added in version 1.2.1.
ConnectionEventArgs	Defines payload for connection events like Connected/Disconnected. Added in version 1.2.0
ConnectionMessageImpl	
ConnectionMessageEventArgs	
ConversationTranscriptionCanceledEventArgs	Defines content of a RecognitionErrorEvent.
CustomCommandsConfig	Class that defines configurations for the dialog service connector object for using a CustomCommands backend.
Diagnostics	Defines diagnostics API for managing console output Added in version 1.21.0
DialogServiceConfig	Class that defines base configurations for dialog service connector
DialogServiceConfigImpl	Dialog Service configuration.
DialogServiceConnector	Dialog Service Connector
IntentRecognitionCanceledEventArgs	Define payload of intent recognition canceled result events.
IntentRecognitionEventArgs	Intent recognition result event arguments.
IntentRecognitionResult	Intent recognition result.
IntentRecognizer	Intent recognizer.
KeywordRecognitionModel	Represents a keyword recognition model for recognizing when the user says a keyword to initiate further speech recognition.
LanguageUnderstandingModel	Language understanding model
NoMatchDetails	Contains detailed information for NoMatch recognition results.
PhraseListGrammar	Allows additions of new phrases to improve speech recognition. Phrases added to the recognizer are effective at the start of the next recognition, or the next time the SpeechSDK must reconnect to the speech service.
PronunciationAssessmentConfig	Pronunciation assessment configuration.
PronunciationAssessment	Pronunciation assessment results.

Result	
PropertyCollection	Represents collection of properties and their values.
RecognitionEventArgs	Defines payload for session events like Speech Start/End Detected
RecognitionResult	Defines result of speech recognition.
Recognizer	Defines the base class Recognizer which mainly contains common event handlers.
ServiceEventArgs	Defines payload for any Service message event Added in version 1.9.0
SessionEventArgs	Defines content for session events like SessionStarted/Stopped, SoundStarted/Stopped.
SourceLanguageConfig	Source Language configuration.
SpeakerIdentificationModel	Defines SpeakerIdentificationModel class for Speaker Recognition Model contains a set of profiles against which to identify speaker(s)
SpeakerRecognitionCancellationDetails	
SpeakerRecognitionResult	Output format
SpeakerRecognizer	Defines SpeakerRecognizer class for Speaker Recognition Handles operations from user for Voice Profile operations (e.g. createProfile, deleteProfile)
SpeakerVerificationModel	Defines SpeakerVerificationModel class for Speaker Recognition Model contains a profile against which to verify a speaker
SpeechConfig	Speech configuration.
SpeechConfigImpl	
SpeechRecognitionCanceledEventArgs	
ConversationTranscriptionEventArgs	Defines contents of conversation transcribed/transcribing event.
SpeechRecognitionEventArgs	Defines contents of speech recognizing/recognized event.
SpeechRecognitionResult	Defines result of speech recognition.
SpeechRecognizer	Performs speech recognition from microphone, file, or other audio input streams, and gets transcribed text as result.
SpeechSynthesisBookmarkEventArgs	Defines contents of speech synthesis bookmark event.

SpeechSynthesisEventArgs	Defines contents of speech synthesis events.
SpeechSynthesisResult	Defines result of speech synthesis.
SpeechSynthesisVisemeEventArgs	Defines contents of speech synthesis viseme event.
SpeechSynthesisWordBoundaryEventArgs	Defines contents of speech synthesis word boundary event.
SpeechSynthesizer	Defines the class SpeechSynthesizer for text to speech. Updated in version 1.16.0
SynthesisRequest	
SpeechTranslationConfig	Speech translation configuration.
SynthesisResult	Base class for synthesis results
SynthesisVoicesResult	Defines result of speech synthesis.
Conversation	
ConversationImpl	
ConversationCommon	
ConversationExpirationEventArgs	
ConversationParticipantsChangedEventArgs	
ConversationTranscriber	
ConversationTranslationCanceledEventArgs	
ConversationTranslationEventArgs	
ConversationTranslationResult	
ConversationTranslator	Join, leave or connect to a conversation.
Participant	
User	
TranslationRecognitionCanceledEventArgs	Define payload of speech recognition canceled result events.
TranslationRecognitionEventArgs	Translation text result event arguments.
TranslationRecognitionResult	Translation text result.
TranslationRecognizer	Translation recognizer

TranslationSynthesisEventArgs	Translation Synthesis event arguments
TranslationSynthesisResult	Defines translation synthesis result, i.e. the voice output of the translated text in the target language.
Translations	Represents collection of parameters and their values.
TurnStatusReceivedEventArgs	Defines contents of received message/events.
VoiceInfo	Information about Speech Synthesis voice Added in version 1.20.0.
VoiceProfile	Defines Voice Profile class for Speaker Recognition
VoiceProfileClient	Defines VoiceProfileClient class for Speaker Recognition Handles operations from user for Voice Profile operations (e.g. createProfile, deleteProfile)
VoiceProfileEnrollmentCancellationDetails	
VoiceProfileEnrollmentResult	Output format
VoiceProfilePhraseResult	Output format
VoiceProfileCancellationDetails	
VoiceProfileResult	Output format

Interfaces

IRecorder	
IRequestOptions	HTTP request helper
IRestParams	
IRestResponse	
IAgentConfig	
IDynamicGrammar	Top level grammar node
IDynamicGrammarGeneric	Generic phrase based dynamic grammars
IDynamicGrammarGroup	Group of Dynamic Grammar items of a common type.
IDynamicGrammarPeople	
IAuthentication	
IConnectionFactory	

[ISynthesisConnectionFactory](#)

[Context](#)

[ISpeechConfigAudio](#)

[ISpeechConfigAudioDevice](#)

[IDetailedSpeechPhrase](#)

[IPhrase](#)

[IWord](#)

[IIIntentEntity](#)

[IIIntentResponse](#)

[ISingleIntent](#)

[IPrimaryLanguage](#)

[ISimpleSpeechPhrase](#)

[ISpeechDetected](#)

[ISpeechHypothesis](#)

[ISpeechKeyword](#)

[ISynthesisAudioMetadata](#)

[ISynthesisMetadata](#)

[ITranslationHypothesis](#)

[ITranslationPhrase](#)

[ITranslationSynthesisEnd](#)

[IResultErrorDetails](#)

[ISpeechEndDetectedResult](#)

[ITranslation](#)

[ITranslations](#)

[ITurnStart](#)

[ITurnStartContext](#)

[ISynthesisResponse](#)

ISynthesisResponseAudio	
ISynthesisResponseContext	
ConversationRecognizer	Recognizer for handling Conversation Translator websocket messages
IChangeNicknameCommand	Change nickname command
IClientMessage	Base message command
 ICommandMessage	Command message
IConversationResponseError	Error returned from the Conversation Translator websocket
IConversationResponseErrorMessage	Error message returned from the Conversation Translator websocket
IEjectParticipantCommand	Remove participant command
IInstantMessageCommand	Text message command
IInternalConversation	Internal conversation data
IInternalParticipant	The user who is participating in the conversation.
ILockConversationCommand	Lock command
IMuteAllCommand	Mute all command
IMuteCommand	Mute participant command
IResponse	HTTP response helper
IAudioDestination	
IAudioSource	
IAudioStreamNode	
IConnection	
IDetachable	
INumberDictionary	
IStringDictionary	
IDisposable	
IErrorMessages	
IEventListener	

IEventSource	
ITimer	
IWebSocketMessageFormatter	
IList	
IDeferred	
IQueue	
IStreamChunk	
IWorkerTimers	
IPlayer	Represents audio player interface to control the audio playback, such as pause, resume, etc.
CancellationEventArgs	
DetailResult	
WordResult	
ConversationHandler	
ConversationTranscription Handler	A conversation transcriber that enables a connected experience where conversations can be logged with each participant recognized.
IConversationTranslator	A conversation translator that enables a connected experience where participants can use their own devices to see everyone else's recognitions and IMs in their own languages. Participants can also speak and send IMs to others.
ConversationInfo	
ConversationProperties	
IConversation	Manages conversations. Added in version 1.4.0
IParticipant	Represents a participant in a conversation. Added in version 1.4.0
IUser	Represents a user in a conversation. Added in version 1.4.0
TranscriptionParticipant	
VoiceSignature	
EnrollmentResultDetails	
EnrollmentResultJSON	

Type Aliases

[Callback](#)

Enums

RestRequestType	
RecognitionCompletionStatus	
RecognitionMode	
SpeechResultFormat	
connectivity	
type	
RecognitionStatus	
MetadataType	
SynthesisServiceType	
TranslationStatus	Defines translation status.
MessageType	
ConnectionState	
EventType	
PromiseState	
AudioFormatTag	
CancellationErrorCode	Defines error code in case that CancellationReason is Error. Added in version 1.1.0.
CancellationReason	Defines the possible reasons a recognition result might be canceled.
LanguageIDMode	Language Identification mode
NoMatchReason	Defines the possible reasons a recognition result might not be recognized.
OutputFormat	Define Speech Recognizer output formats.
ProfanityOption	Profanity option. Added in version 1.7.0.

PronunciationAssessmentGradingSystem	Defines the point system for pronunciation score calibration; default value is FivePoint. Added in version 1.15.0
PronunciationAssessmentGranularity	Defines the pronunciation evaluation granularity; default value is Phoneme. Added in version 1.15.0
PropertyId	Defines speech property ids.
ResultReason	Defines the possible reasons a recognition result might be generated.
ServicePropertyChannel	Defines channels used to pass property settings to service. Added in version 1.7.0.
SpeakerRecognitionResultType	
SpeechSynthesisBoundaryType	Defines the boundary type of speech synthesis boundary event.
SpeechSynthesisOutputFormat	Define speech synthesis audio output formats.
SpeechState	
ParticipantChangedReason	
SynthesisVoiceGender	Defines the gender of synthesis voices. Added in version 1.20.0.
SynthesisVoiceType	
VoiceProfileType	Output format

Functions

```
marshalPromise
ToCallbacks<T>(Promise<T>,
(value: T) => void, (error:
string) => void)
```

Function Details

marshalPromiseToCallbacks<T>(Promise<T>, (value: T) => void, (error: string) => void)

TypeScript

```
function marshalPromiseToCallbacks<T>(promise: Promise<T>, cb?: (value: T) => void, err?: (error: string) => void)
```

Parameters

promise Promise<T>

cb (value: T) => void

err (error: string) => void

Speech SDK for Objective-C Reference

Article • 04/16/2022 • 2 minutes to read

Classes

- [SPXAudioConfiguration class](#)
- [SPXAudioDataStream class](#)
- [SPXAudioInputStream class](#)
- [SPXAudioOutputStream class](#)
- [SPXAudioStreamFormat class](#)
- [SPXAutoDetectSourceLanguageConfiguration class](#)
- [SPXAutoDetectSourceLanguageResult class](#)
- [SPXCancellationDetails class](#)
- [SPXConnection class](#)
- [SPXConnectionEventArgs class](#)
- [SPXConversation class](#)
- [SPXConversationExpirationEventArgs class](#)
- [SPXConversationParticipantsChangedEventArgs class](#)
- [SPXConversationTranscriber class](#)
- [SPXConversationTranscriptionCanceledEventArgs class](#)
- [SPXConversationTranscriptionEventArgs class](#)
- [SPXConversationTranscriptionResult class](#)
- [SPXConversationTranslationCanceledEventArgs class](#)
- [SPXConversationTranslationEventArgs class](#)
- [SPXConversationTranslationResult class](#)
- [SPXConversationTranslator class](#)
- [SPXDialogBotFrameworkConfiguration class](#)
- [SPXDialogCustomCommandsConfiguration class](#)
- [SPXDialogServiceConfiguration class](#)
- [SPXDialogServiceConnector class](#)
- [SPXDialogServiceConnectorActivityReceivedEventArgs class](#)
- [SPXDialogServiceConnectorTurnStatusReceivedEventArgs class](#)
- [SPXGrammar class](#)
- [SPXGrammarPhrase class](#)
- [SPXIntentRecognitionCanceledEventArgs class](#)
- [SPXIntentRecognitionEventArgs class](#)
- [SPXIntentRecognitionResult class](#)
- [SPXIntentRecognizer class](#)
- [SPXKeywordRecognitionCanceledEventArgs class](#)

- [SPXKeywordRecognitionEventArgs](#) class
- [SPXKeywordRecognitionModel](#) class
- [SPXKeywordRecognitionResult](#) class
- [SPXKeywordRecognizer](#) class
- [SPXLanguageUnderstandingModel](#) class
- [SPXNBestPhoneme](#) class
- [SPXNoMatchDetails](#) class
- [SPXParticipant](#) class
- [SPXPhonemeLevelTimingResult](#) class
- [SPXPhraseListGrammar](#) class
- [SPXPronunciationAssessmentConfiguration](#) class
- [SPXPronunciationAssessmentResult](#) class
- [SPXPullAudioInputStream](#) class
- [SPXPullAudioOutputStream](#) class
- [SPXPushAudioInputStream](#) class
- [SPXPushAudioOutputStream](#) class
- [SPXRecognitionEventArgs](#) class
- [SPXRecognitionResult](#) class
- [SPXRecognizer](#) class
- [SPXSessionEventArgs](#) class
- [SPXSourceLanguageConfiguration](#) class
- [SPXSpeechConfiguration](#) class
- [SPXSpeechRecognitionCanceledEventArgs](#) class
- [SPXSpeechRecognitionEventArgs](#) class
- [SPXSpeechRecognitionResult](#) class
- [SPXSpeechRecognizer](#) class
- [SPXSpeechSynthesisBookmarkEventArgs](#) class
- [SPXSpeechSynthesisCancellationDetails](#) class
- [SPXSpeechSynthesisEventArgs](#) class
- [SPXSpeechSynthesisResult](#) class
- [SPXSpeechSynthesisVisemeEventArgs](#) class
- [SPXSpeechSynthesisWordBoundaryEventArgs](#) class
- [SPXSpeechSynthesizer](#) class
- [SPXSpeechTranslationConfiguration](#) class
- [SPXSyllableLevelTimingResult](#) class
- [SPXSynthesisVoicesResult](#) class
- [SPXTimingResult](#) class
- [SPXTranslationRecognitionCanceledEventArgs](#) class
- [SPXTranslationRecognitionEventArgs](#) class
- [SPXTranslationRecognitionResult](#) class

- [SPXTranslationRecognizer](#) class
- [SPXTranslationSynthesisEventArgs](#) class
- [SPXTranslationSynthesisResult](#) class
- [SPXUser](#) class
- [SPXVoiceInfo](#) class
- [SPXWordLevelTimingResult](#) class

Protocols

- [SPXPropertyCollection](#) protocol

Enums

- [SPXAudioStreamContainerFormat](#) enum
- [SPXCancellationErrorCode](#) enum
- [SPXCancellationReason](#) enum
- [SPXNoMatchReason](#) enum
- [SPXOutputFormat](#) enum
- [SPXParticipantChangedReason](#) enum
- [SPXPronunciationAssessmentGradingSystem](#) enum
- [SPXPronunciationAssessmentGranularity](#) enum
- [SPXPropertyId](#) enum
- [SPXResultReason](#) enum
- [SPXServicePropertyChannel](#) enum
- [SPXSpeechConfigProfanityOption](#) enum
- [SPXSpeechSynthesisBoundaryType](#) enum
- [SPXSpeechSynthesisOutputFormat](#) enum
- [SPXStreamStatus](#) enum
- [SPXSynthesisVoiceGender](#) enum
- [SPXSynthesisVoiceType](#) enum

Typedefs

- [SPXConnectionEventHandler](#) typedef
- [SPXConversationExpirationEventHandler](#) typedef
- [SPXConversationParticipantsChangedEventHandler](#) typedef
- [SPXConversationSessionEventHandler](#) typedef
- [SPXConversationTranscriptionCanceledEventHandler](#) typedef
- [SPXConversationTranscriptionEventHandler](#) typedef

- [SPXConversationTranslationCanceledEventHandler](#) typedef
- [SPXConversationTranslationEventHandler](#) typedef
- [SPXDialogServiceConnectorActivityReceivedEventHandler](#) typedef
- [SPXDialogServiceConnectorAsyncCompletionHandler](#) typedef
- [SPXDialogServiceConnectorCanceledEventHandler](#) typedef
- [SPXDialogServiceConnectorInteractionIdHandler](#) typedef
- [SPXDialogServiceConnectorRecognitionEventHandler](#) typedef
- [SPXDialogServiceConnectorRecognitionResultHandler](#) typedef
- [SPXDialogServiceConnectorTurnStatusReceivedEventHandler](#) typedef
- [SPXIntentRecognitionCanceledEventHandler](#) typedef
- [SPXIntentRecognitionEventHandler](#) typedef
- [SPXKeywordRecognitionCanceledEventHandler](#) typedef
- [SPXKeywordRecognitionEventHandler](#) typedef
- [SPXPullAudioInputStreamCloseHandler](#) typedef
- [SPXPullAudioInputStreamGetPropertyHandler](#) typedef
- [SPXPullAudioInputStreamReadHandler](#) typedef
- [SPXPushAudioOutputStreamCloseHandler](#) typedef
- [SPXPushAudioOutputStreamWriteHandler](#) typedef
- [SPXRecognitionEventHandler](#) typedef
- [SPXSessionEventHandler](#) typedef
- [SPXSpeechRecognitionCanceledEventHandler](#) typedef
- [SPXSpeechRecognitionEventHandler](#) typedef
- [SPXSpeechSynthesisBookmarkEventHandler](#) typedef
- [SPXSpeechSynthesisEventHandler](#) typedef
- [SPXSpeechSynthesisVisemeEventHandler](#) typedef
- [SPXSpeechSynthesisWordBoundaryEventHandler](#) typedef
- [SPXTranslationRecognitionCanceledEventHandler](#) typedef
- [SPXTranslationRecognitionEventHandler](#) typedef
- [SPXTranslationSynthesisEventHandler](#) typedef

azure-cognitiveservices-speech Package

Reference

In this article

[Packages](#)

[Modules](#)

Packages

[speech](#)

Microsoft Speech SDK for Python

Modules

audio	Classes that are concerned with the handling of audio input to the various recognizers, and audio output from the speech synthesizer.
dialog	Classes related to dialog service connector.
enums	
intent	Classes related to intent recognition from speech.
interop	
languageconfig	Classes that are concerned with the handling of language configurations
properties	
speech	Classes related to recognizing text from speech, synthesizing speech from text, and general classes used in the various recognizers.
transcription	Classes related to conversation transcription.
translation	Classes related to translation of speech to other languages.
version	

Speech-to-text REST API

Article • 01/25/2023 • 5 minutes to read

Speech-to-text REST API is used for [Batch transcription](#) and [Custom Speech](#).

ⓘ Important

Speech-to-text REST API v3.1 is generally available. Version 3.0 of the [Speech to Text REST API](#) will be retired. For more information, see the [Migrate code from v3.0 to v3.1 of the REST API](#) guide.

[See the Speech to Text API v3.1 reference documentation](#)

[See the Speech to Text API v3.0 reference documentation](#)

Use Speech-to-text REST API to:

- [Custom Speech](#): With Custom Speech, you can upload your own data, test and train a custom model, compare accuracy between models, and deploy a model to a custom endpoint. Copy models to other subscriptions if you want colleagues to have access to a model that you built, or if you want to deploy a model to more than one region.
- [Batch transcription](#): Transcribe audio files as a batch from multiple URLs or an Azure container.

Speech-to-text REST API includes such features as:

- Get logs for each endpoint if logs have been requested for that endpoint.
- Request the manifest of the models that you create, to set up on-premises containers.
- Upload data from Azure storage accounts by using a shared access signature (SAS) URI.
- Bring your own storage. Use your own storage accounts for logs, transcription files, and other data.
- Some operations support webhook notifications. You can register your webhooks where notifications are sent.

Datasets

Datasets are applicable for [Custom Speech](#). You can use datasets to train and test the performance of different models. For example, you can compare the performance of a model trained with a specific dataset to the performance of a model trained with a different dataset.

See [Upload training and testing datasets](#) for examples of how to upload datasets. This table includes all the operations that you can perform on datasets.

Path	Method	Version 3.1	Version 3.0
/datasets	GET	Datasets_List	GetDatasets
/datasets	POST	Datasets_Create	CreateDataset
/datasets/{id}	DELETE	Datasets_Delete	DeleteDataset
/datasets/{id}	GET	Datasets_Get	GetDataset
/datasets/{id}	PATCH	Datasets_Update	UpdateDataset
/datasets/{id}/blocks:commit	POST	Datasets_CommitBlocks	Not applicable

Path	Method	Version 3.1	Version 3.0
/datasets/{id}/blocks	GET	Datasets_GetBlocks ↗	Not applicable
/datasets/{id}/blocks	PUT	Datasets_UploadBlock ↗	Not applicable
/datasets/{id}/files	GET	Datasets_ListFiles ↗	GetDatasetFiles ↗
/datasets/{id}/files/{fileId}	GET	Datasets_GetFile ↗	GetDatasetFile ↗
/datasets/locales	GET	Datasets_ListSupportedLocales ↗	GetSupportedLocalesForDatasets ↗
/datasets/upload	POST	Datasets_Upload ↗	UploadDatasetFromForm ↗

Endpoints

Endpoints are applicable for [Custom Speech](#). You must deploy a custom endpoint to use a Custom Speech model.

See [Deploy a model](#) for examples of how to manage deployment endpoints. This table includes all the operations that you can perform on endpoints.

Path	Method	Version 3.1	Version 3.0
/endpoints	GET	Endpoints_List ↗	GetEndpoints ↗
/endpoints	POST	Endpoints_Create ↗	CreateEndpoint ↗
/endpoints/{id}	DELETE	Endpoints_Delete ↗	DeleteEndpoint ↗
/endpoints/{id}	GET	Endpoints_Get ↗	GetEndpoint ↗
/endpoints/{id}	PATCH	Endpoints_Update ↗	UpdateEndpoint ↗
/endpoints/{id}/files/logs	DELETE	Endpoints_DeleteLogs ↗	DeleteEndpointLogs ↗
/endpoints/{id}/files/logs	GET	Endpoints_ListLogs ↗	GetEndpointLogs ↗
/endpoints/{id}/files/logs/{logId}	DELETE	Endpoints_DeleteLog ↗	DeleteEndpointLog ↗
/endpoints/{id}/files/logs/{logId}	GET	Endpoints_GetLog ↗	GetEndpointLog ↗
/endpoints/base/{locale}/files/logs	DELETE	Endpoints_DeleteBaseModelLogs ↗	DeleteBaseModelLogs ↗
/endpoints/base/{locale}/files/logs	GET	Endpoints_ListBaseModelLogs ↗	GetBaseModelLogs ↗
/endpoints/base/{locale}/files/logs/{logId}	DELETE	Endpoints_DeleteBaseModelLog ↗	DeleteBaseModelLog ↗
/endpoints/base/{locale}/files/logs/{logId}	GET	Endpoints_GetBaseModelLog ↗	GetBaseModelLog ↗
/endpoints/locales	GET	Endpoints_ListSupportedLocales ↗	GetSupportedLocalesForEndpoints ↗

Evaluations

Evaluations are applicable for [Custom Speech](#). You can use evaluations to compare the performance of different models. For example, you can compare the performance of a model trained with a specific dataset to the performance of a model trained with a different dataset.

See [Test recognition quality](#) and [Test accuracy](#) for examples of how to test and evaluate Custom Speech models. This table includes all the operations that you can perform on evaluations.

Path	Method	Version 3.1	Version 3.0
/evaluations	GET	Evaluations_List ↗	GetEvaluations ↗
/evaluations	POST	Evaluations_Create ↗	CreateEvaluation ↗
/evaluations/{id}	DELETE	Evaluations_Delete ↗	DeleteEvaluation ↗
/evaluations/{id}	GET	Evaluations_Get ↗	GetEvaluation ↗
/evaluations/{id}	PATCH	Evaluations_Update ↗	UpdateEvaluation ↗
/evaluations/{id}/files	GET	Evaluations_ListFiles ↗	GetEvaluationFiles ↗
/evaluations/{id}/files/{fileId}	GET	Evaluations_GetFile ↗	GetEvaluationFile ↗
/evaluations/locales	GET	Evaluations_ListSupportedLocales ↗	GetSupportedLocalesForEvaluations ↗

Health status

Health status provides insights about the overall health of the service and sub-components.

Path	Method	Version 3.1	Version 3.0
/healthstatus	GET	HealthStatus_Get ↗	GetHealthStatus ↗

Models

Models are applicable for [Custom Speech](#) and [Batch Transcription](#). You can use models to transcribe audio files. For example, you can use a model trained with a specific dataset to transcribe audio files.

See [Train a model](#) and [Custom Speech model lifecycle](#) for examples of how to train and manage Custom Speech models. This table includes all the operations that you can perform on models.

Path	Method	Version 3.1	Version 3.0
/models	GET	Models_ListCustomModels ↗	GetModels ↗
/models	POST	Models_Create ↗	CreateModel ↗
/models/{id}:copyto ¹	POST	Models_CopyTo ↗	CopyModelToSubscription ↗
/models/{id}	DELETE	Models_Delete ↗	DeleteModel ↗
/models/{id}	GET	Models_GetCustomModel ↗	GetModel ↗
/models/{id}	PATCH	Models_Update ↗	UpdateModel ↗
/models/{id}/files	GET	Models_ListFiles ↗	Not applicable
/models/{id}/files/{fileId}	GET	Models_GetFile ↗	Not applicable
/models/{id}/manifest	GET	Models_GetCustomModelManifest ↗	GetModelManifest ↗
/models/base	GET	Models_ListBaseModels ↗	GetBaseModels ↗
/models/base/{id}	GET	Models_GetBaseModel ↗	Get BaseModel ↗
/models/base/{id}/manifest	GET	Models_GetBaseModelManifest ↗	Get BaseModelManifest ↗

Path	Method	Version 3.1	Version 3.0
/models/locales	GET	Models_ListSupportedLocales ↗	GetSupportedLocalesForModels ↗

Projects

Projects are applicable for [Custom Speech](#). Custom Speech projects contain models, training and testing datasets, and deployment endpoints. Each project is specific to a [locale](#). For example, you might create a project for English in the United States.

See [Create a project](#) for examples of how to create projects. This table includes all the operations that you can perform on projects.

Path	Method	Version 3.1	Version 3.0
/projects	GET	Projects_List ↗	GetProjects ↗
/projects	POST	Projects_Create ↗	CreateProject ↗
/projects/{id}	DELETE	Projects_Delete ↗	DeleteProject ↗
/projects/{id}	GET	Projects_Get ↗	GetProject ↗
/projects/{id}	PATCH	Projects_Update ↗	UpdateProject ↗
/projects/{id}/datasets	GET	Projects_ListDatasets ↗	GetDatasetsForProject ↗
/projects/{id}/endpoints	GET	Projects_ListEndpoints ↗	GetEndpointsForProject ↗
/projects/{id}/evaluations	GET	Projects_ListEvaluations ↗	GetEvaluationsForProject ↗
/projects/{id}/models	GET	Projects_ListModels ↗	GetModelsForProject ↗
/projects/{id}/transcriptions	GET	Projects_ListTranscriptions ↗	GetTranscriptionsForProject ↗
/projects/locales	GET	Projects_ListSupportedLocales ↗	GetSupportedProjectLocales ↗

Transcriptions

Transcriptions are applicable for [Batch Transcription](#). Batch transcription is used to transcribe a large amount of audio in storage. You should send multiple files per request or point to an Azure Blob Storage container with the audio files to transcribe.

See [Create a transcription](#) for examples of how to create a transcription from multiple audio files. This table includes all the operations that you can perform on transcriptions.

Path	Method	Version 3.1	Version 3.0
/transcriptions	GET	Transcriptions_List ↗	GetTranscriptions ↗
/transcriptions	POST	Transcriptions_Create ↗	CreateTranscription ↗
/transcriptions/{id}	DELETE	Transcriptions_Delete ↗	DeleteTranscription ↗
/transcriptions/{id}	GET	Transcriptions_Get ↗	GetTranscription ↗
/transcriptions/{id}	PATCH	Transcriptions_Update ↗	UpdateTranscription ↗
/transcriptions/{id}/files	GET	Transcriptions_ListFiles ↗	GetTranscriptionFiles ↗

Path	Method	Version 3.1	Version 3.0
/transcriptions/{id}/files/{fileId}	GET	Transcriptions_GetFile ↗	GetTranscriptionFile ↗
/transcriptions/locales	GET	Transcriptions_ListSupportedLocales ↗	GetSupportedLocalesForTranscriptions ↗

Web hooks

Web hooks are applicable for [Custom Speech](#) and [Batch Transcription](#). In particular, web hooks apply to [datasets](#), [endpoints](#), [evaluations](#), [models](#), and [transcriptions](#). Web hooks can be used to receive notifications about creation, processing, completion, and deletion events.

This table includes all the web hook operations that are available with the speech-to-text REST API.

Path	Method	Version 3.1	Version 3.0
/webhooks	GET	WebHooks_List ↗	GetHooks ↗
/webhooks	POST	WebHooks_Create ↗	CreateHook ↗
/webhooks/{id}:ping ¹	POST	WebHooks_Ping ↗	PingHook ↗
/webhooks/{id}:test ²	POST	WebHooks_Test ↗	TestHook ↗
/webhooks/{id}	DELETE	WebHooks_Delete ↗	DeleteHook ↗
/webhooks/{id}	GET	WebHooks_Get ↗	GetHook ↗
/webhooks/{id}	PATCH	WebHooks_Update ↗	UpdateHook ↗

¹ The `/webhooks/{id}/ping` operation (includes '/') in version 3.0 is replaced by the `/webhooks/{id}:ping` operation (includes ':') in version 3.1.

² The `/webhooks/{id}/test` operation (includes '/') in version 3.0 is replaced by the `/webhooks/{id}:test` operation (includes ':') in version 3.1.

Next steps

- [Create a Custom Speech project](#)
- [Get familiar with batch transcription](#)

Additional resources

Documentation

[How to use compressed input audio - Speech service - Azure Cognitive Services](#)

Learn how to use compressed input audio the Speech SDK and CLI.

[Improve recognition accuracy with phrase list - Azure Cognitive Services](#)

Phrase lists can be used to customize speech recognition results based on context.

[Get speech recognition results - Speech service - Azure Cognitive Services](#)

Learn how to get speech recognition results.

[Language identification - Speech service - Azure Cognitive Services](#)

Language identification is used to determine the language being spoken in audio when compared against a list of provided languages.

[Display text formatting with speech to text - Speech service - Azure Cognitive Services](#)

An overview of key concepts for display text formatting with speech to text.

[How to recognize speech - Speech service - Azure Cognitive Services](#)

Learn how to convert speech to text, including object construction, supported audio input formats, and configuration options for speech recognition.

[Speech-to-text overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the speech-to-text feature of the Speech Service.

[Training and testing datasets - Speech service - Azure Cognitive Services](#)

Learn about types of training and testing data for a Custom Speech project, along with how to use and manage that data.

[Show 5 more](#)

Speech-to-text REST API for short audio

Article • 01/11/2023 • 12 minutes to read

Use cases for the speech-to-text REST API for short audio are limited. Use it only in cases where you can't use the [Speech SDK](#).

Before you use the speech-to-text REST API for short audio, consider the following limitations:

- Requests that use the REST API for short audio and transmit audio directly can contain no more than 60 seconds of audio. The input [audio formats](#) are more limited compared to the [Speech SDK](#).
- The REST API for short audio returns only final results. It doesn't provide partial results.
- [Speech translation](#) is not supported via REST API for short audio. You need to use [Speech SDK](#).
- [Batch transcription](#) and [Custom Speech](#) are not supported via REST API for short audio. You should always use the [Speech to Text REST API](#) for batch transcription and Custom Speech.

Before you use the speech-to-text REST API for short audio, understand that you need to complete a token exchange as part of authentication to access the service. For more information, see [Authentication](#).

Regions and endpoints

The endpoint for the REST API for short audio has this format:

```
https://<REGION_IDENTIFIER>.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
```

Replace `<REGION_IDENTIFIER>` with the identifier that matches the [region](#) of your Speech resource.

Note

For Azure Government and Azure China endpoints, see [this article about sovereign clouds](#).

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	Bit rate	Sample rate
WAV	PCM	256 kbps	16 kHz, mono
OGG	OPUS	256 kbps	16 kHz, mono

ⓘ Note

The preceding formats are supported through the REST API for short audio and WebSocket in the Speech service. The [Speech SDK](#) supports the WAV format with PCM codec as well as [other formats](#).

Request headers

This table lists required and optional headers for speech-to-text requests:

Header	Description	Required or optional
<code>Ocp-Apim-Subscription-Key</code>	Your resource key for the Speech service.	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 Authentication .	Either this header or <code>ocp-Apim-Subscription-Key</code> is required.
<code>Pronunciation-Assessment</code>	Specifies the parameters for showing pronunciation scores in recognition results. These scores assess the pronunciation quality of speech input, with indicators like accuracy, fluency, and completeness. This parameter is a Base64-encoded JSON that contains multiple detailed parameters. To learn how to build this header, see Pronunciation assessment parameters .	Optional

Header	Description	Required or optional
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. Use this header only if you're chunking audio data.	Optional
Expect	If you're using chunked transfer, send <code>Expect: 100-continue</code> . The Speech service acknowledges the initial request and awaits additional data.	Required if you're 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's good practice to always include <code>Accept</code> .	Optional, but recommended.

Query parameters

These parameters might be included in the query string of the REST request.

① Note

You must append the language parameter to the URL to avoid receiving a 4xx HTTP error. For example, the language set to US English via the West US endpoint is:

```
https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US.
```

Parameter	Description	Required or optional
language	Identifies the spoken language that's being recognized. See Supported languages .	Required
format	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 four different representations of display text. The default setting is <code>simple</code> .	Optional

Parameter	Description	Required or optional
<code>profanity</code>	<p>Specifies how to handle profanity in recognition results. Accepted values are:</p> <ul style="list-style-type: none"> <code>masked</code>, which replaces profanity with asterisks. <code>removed</code>, which removes all profanity from the result. <code>raw</code>, which includes profanity in the result. <p>The default setting is <code>masked</code>.</p>	Optional
<code>cid</code>	<p>When you're using the Speech Studio to create custom models, you can take advantage of the Endpoint ID value from the Deployment page. Use the Endpoint ID value as the argument to the <code>cid</code> query string parameter.</p>	Optional

Pronunciation assessment parameters

This table lists required and optional parameters for pronunciation assessment:

Parameter	Description	Required or optional
<code>ReferenceText</code>	The text that the pronunciation will be evaluated against.	Required
<code>GradingSystem</code>	<p>The point system for score calibration. The <code>FivePoint</code> system gives a 0-5 floating point score, and <code>HundredMark</code> gives a 0-100 floating point score. Default: <code>FivePoint</code>.</p>	Optional
<code>Granularity</code>	<p>The evaluation granularity. Accepted values are:</p> <ul style="list-style-type: none"> <code>Phoneme</code>, which shows the score on the full-text, word, and phoneme levels. <code>Word</code>, which shows the score on the full-text and word levels. <code>FullText</code>, which shows the score on the full-text level only. <p>The default setting is <code>Phoneme</code>.</p>	Optional

Parameter	Description	Required or optional
Dimension	<p>Defines the output criteria. Accepted values are:</p> <ul style="list-style-type: none"> <code>Basic</code>, which shows the accuracy score only. <code>Comprehensive</code>, which shows scores on more dimensions (for example, fluency score and completeness score on the full-text level, and error type on the word level). <p>To see definitions of different score dimensions and word error types, see Response properties. The default setting is <code>Basic</code>.</p>	Optional
EnableMispue	<p>Enables mispue calculation. With this parameter enabled, the pronounced words will be compared to the reference text. They'll be marked with omission or insertion based on the comparison.</p> <p>Accepted values are <code>False</code> and <code>True</code>. The default setting is <code>False</code>.</p>	Optional
ScenarioId	A GUID that indicates a customized point system.	Optional

Here's example JSON that contains the pronunciation assessment parameters:

```
JSON
{
  "ReferenceText": "Good morning.",
  "GradingSystem": "HundredMark",
  "Granularity": "FullText",
  "Dimension": "Comprehensive"
}
```

The following sample code shows how to build the pronunciation assessment parameters into the `Pronunciation-Assessment` header:

```
C#
var pronAssessmentParamsJson = $"{{{{\"ReferenceText\":\"Good
morning.\",\"GradingSystem\":\"HundredMark\",\"Granularity\":\"FullText\",\"Dimension\":\"Comprehensive\"}}}}";
var pronAssessmentParamsBytes =
Encoding.UTF8.GetBytes(pronAssessmentParamsJson);
var pronAssessmentHeader =
Convert.ToBase64String(pronAssessmentParamsBytes);
```

We strongly recommend streaming ([chunked transfer](#)) uploading while you're posting the audio data, which can significantly reduce the latency. To learn how to enable streaming, see the [sample code in various programming languages](#).

Note

For more information, see [pronunciation assessment](#).

Sample request

The following sample includes the host name 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 but not required.

HTTP

```
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_RESOURCE_KEY
Host: westus.stt.speech.microsoft.com
Transfer-Encoding: chunked
Expect: 100-continue
```

To enable pronunciation assessment, you can add the following header. To learn how to build this header, see [Pronunciation assessment parameters](#).

HTTP

```
Pronunciation-Assessment: eyJSZWZlc...  
...
```

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP status code	Description	Possible reasons
100	Continue	The initial request has been accepted. Proceed with sending the rest of the data. (This code is used with chunked transfer.)
200	OK	The request was successful. The response body is a JSON object.
400	Bad request	The language code wasn't provided, the language isn't supported, or the audio file is invalid (for example).

HTTP status code	Description	Possible reasons
401	Unauthorized	A resource key or an authorization token is invalid in the specified region, or an endpoint is invalid.
403	Forbidden	A resource key or authorization token is missing.

Sample responses

Here's a typical response for `simple` recognition:

JSON

```
{
  "RecognitionStatus": "Success",
  "DisplayText": "Remind me to buy 5 pencils.",
  "Offset": "1236645672289",
  "Duration": "1236645672289"
}
```

Here's a typical response for `detailed` recognition:

JSON

```
{
  "RecognitionStatus": "Success",
  "Offset": "1236645672289",
  "Duration": "1236645672289",
  "NBest": [
    {
      "Confidence": 0.9052885,
      "Display": "What's the weather like?",
      "ITN": "what's the weather like",
      "Lexical": "what's the weather like",
      "MaskedITN": "what's the weather like"
    },
    {
      "Confidence": 0.92459863,
      "Display": "what is the weather like",
      "ITN": "what is the weather like",
      "Lexical": "what is the weather like",
      "MaskedITN": "what is the weather like"
    }
  ]
}
```

Here's a typical response for recognition with pronunciation assessment:

JSON

```
{  
    "RecognitionStatus": "Success",  
    "Offset": "400000",  
    "Duration": "11000000",  
    "NBest": [  
        {  
            "Confidence" : "0.87",  
            "Lexical" : "good morning",  
            "ITN" : "good morning",  
            "MaskedITN" : "good morning",  
            "Display" : "Good morning.",  
            "PronScore" : 84.4,  
            "AccuracyScore" : 100.0,  
            "FluencyScore" : 74.0,  
            "CompletenessScore" : 100.0,  
            "Words": [  
                {  
                    "Word" : "Good",  
                    "AccuracyScore" : 100.0,  
                    "ErrorType" : "None",  
                    "Offset" : 500000,  
                    "Duration" : 2700000  
                },  
                {  
                    "Word" : "morning",  
                    "AccuracyScore" : 100.0,  
                    "ErrorType" : "None",  
                    "Offset" : 5300000,  
                    "Duration" : 900000  
                }  
            ]  
        }  
    ]  
}
```

Response properties

Results are provided as JSON. The `simple` format includes the following top-level fields:

Property	Description
<code>RecognitionStatus</code>	Status, such as <code>Success</code> for successful recognition. See the next table.

Property	Description
DisplayText	The recognized text after capitalization, punctuation, inverse text normalization, and profanity masking. Present only on success. Inverse text normalization is conversion of spoken text to shorter forms, such as 200 for "two hundred" or "Dr. Smith" for "doctor smith."
Offset	The time (in 100-nanosecond units) at which the recognized speech begins in the audio stream.
Duration	The duration (in 100-nanosecond units) of the recognized speech in the audio stream.

The `RecognitionStatus` field might contain these values:

Status	Description
Success	The recognition was successful, and the <code>DisplayText</code> field is present.
NoMatch	Speech was detected in the audio stream, but no words from the target language were matched. This status usually means that the recognition language is different from the language that the user is speaking.
InitialSilenceTimeout	The start of the audio stream contained only silence, and the service timed out while waiting for speech.
BabbleTimeout	The start of the audio stream contained only noise, and the service timed out while waiting for speech.
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 additional forms of recognized results. When you're using the `detailed` format, `DisplayText` is provided as `Display` for each result in the `NBest` list.

The object in the `NBest` list can include:

Property	Description
----------	-------------

Property	Description
Confidence	The confidence score of the entry, from 0.0 (no confidence) to 1.0 (full confidence).
Lexical	The lexical form of the recognized text: the actual words recognized.
ITN	The inverse-text-normalized (ITN) or canonical form of the recognized text, with phone numbers, numbers, abbreviations ("doctor smith" to "dr smith"), and other transformations applied.
MaskedITN	The ITN form with profanity masking applied, if requested.
Display	The display form of the recognized text, with punctuation and capitalization added. This parameter is the same as what <code>DisplayText</code> provides when the format is set to <code>simple</code> .
AccuracyScore	Pronunciation accuracy of the speech. Accuracy indicates how closely the phonemes match a native speaker's pronunciation. The accuracy score at the word and full-text levels is aggregated from the accuracy score at the phoneme level.
FluencyScore	Fluency of the provided speech. Fluency indicates how closely the speech matches a native speaker's use of silent breaks between words.
CompletenessScore	Completeness of the speech, determined by calculating the ratio of pronounced words to reference text input.
PronScore	Overall score that indicates the pronunciation quality of the provided speech. This score is aggregated from <code>AccuracyScore</code> , <code>FluencyScore</code> , and <code>CompletenessScore</code> with weight.
ErrorType	Value that indicates whether a word is omitted, inserted, or badly pronounced, compared to <code>ReferenceText</code> . Possible values are <code>None</code> (meaning no error on this word), <code>Omission</code> , <code>Insertion</code> , and <code>Mispronunciation</code> .

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's transmitted. The REST API for short audio does not provide partial or interim results.

The following 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 that's connected to the appropriate REST endpoint. `audioFile` is the path to an audio file on disk.

C#

```
var 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"] = "YOUR_RESOURCE_KEY";
request.AllowWriteStreamBuffering = false;

using (var fs = new FileStream(audioFile, FileMode.Open, FileAccess.Read))
{
    // Open a request stream and write 1,024-byte chunks in the stream one
    // at a time.
    byte[] buffer = null;
    int bytesRead = 0;
    using (var requestStream = request.GetRequestStream())
    {
        // Read 1,024 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);
        }

        requestStream.Flush();
    }
}
```

Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each feature:

Supported authorization header	Speech-to-text	Text-to-speech
Ocp-Apim-Subscription-Key	Yes	Yes
Authorization: Bearer	Yes	Yes

When you're using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your resource key. For example:

HTTP

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When you're using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your resource key for an access token that's valid for 10 minutes.

How to get an access token

To get an access token, you need to make a request to the `issueToken` endpoint by using `Ocp-Apim-Subscription-Key` and your resource key.

The `issueToken` endpoint has this format:

HTTP

```
https://<REGION_IDENTIFIER>.api.cognitive.microsoft.com/sts/v1.0/issueToken
```

Replace `<REGION_IDENTIFIER>` with the identifier that matches the [region](#) of your subscription.

Use the following 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 resource key for the Speech service. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

HTTP

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: eastus.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 resource key for the Speech service. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

PowerShell

```
$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://eastus.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 resource key for the Speech service. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

Console

```
curl -v -X POST \
"https://eastus.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 resource key for the Speech service 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.

C#

```
public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://eastus.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

Python

```

# 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://eastus.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, but to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the speech-to-text REST API for short audio:

```
HTTP  
  
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...
```

Next steps

- [Customize speech models](#)
- [Get familiar with batch transcription](#)

Additional resources

Documentation

[SpeechRecognizer class](#)

Performs speech recognition from microphone, file, or other audio input streams, and gets transcribed text as result.

[azure.cognitiveservices.speech.SpeechConfig class](#)

Class that defines configurations for speech / intent recognition and speech synthesis. The configuration can be initialized in different ways: from subscription: pass a subscription key and a region from endpoint: pass an endpoint. Subscription key or authorization token are optional. from...

[Troubleshoot the Speech SDK - Speech service - Azure Cognitive Services](#)

This article provides information to help you solve issues you might encounter when you use the Speech SDK.

[azure.cognitiveservices.speech package](#)

Microsoft Speech SDK for Python

[The Azure Speech CLI - Azure Cognitive Services](#)

In this article, you learn about the Speech CLI, a command-line tool for using Speech service without

having to write any code.

[How to lower speech synthesis latency using Speech SDK - Azure Cognitive Services](#)

How to lower speech synthesis latency using Speech SDK, including streaming, pre-connection, and so on.

[Speaker Recognition quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you use speaker recognition to confirm who is speaking. Learn about common design patterns for working with speaker verification and identification.

[Speech SDK logging - Speech service - Azure Cognitive Services](#)

Learn about how to enable logging in the Speech SDK (C++, C#, Python, Objective-C, Java).

[Show 5 more](#)

Text-to-speech REST API

Article • 02/02/2023 • 12 minutes to read

The Speech service allows you to [convert text into synthesized speech](#) and [get a list of supported voices](#) for a region by using a REST API. In this article, you'll learn about authorization options, query options, how to structure a request, and how to interpret a response.

💡 Tip

Use cases for the text-to-speech REST API are limited. Use it only in cases where you can't use the [Speech SDK](#). For example, with the Speech SDK you can [subscribe to events](#) for more insights about the text-to-speech processing and results.

The text-to-speech REST API supports neural text-to-speech voices, which support specific languages and dialects that are identified by locale. Each available endpoint is associated with a region. A Speech resource key for the endpoint or region that you plan to use is required. Here are links to more information:

- For a complete list of voices, see [Language and voice support for the Speech service](#).
- For information about regional availability, see [Speech service supported regions](#).
- For Azure Government and Azure China endpoints, see [this article about sovereign clouds](#).

ⓘ Important

Costs vary for prebuilt neural voices (called *Neural* on the pricing page) and custom neural voices (called *Custom Neural* on the pricing page). For more information, see [Speech service pricing](#).

Before you use the text-to-speech REST API, understand that you need to complete a token exchange as part of authentication to access the service. For more information, see [Authentication](#).

Get a list of voices

You can use the `tts.speech.microsoft.com/cognitiveservices/voices/list` endpoint to get a full list of voices for a specific region or endpoint. Prefix the voices list endpoint with a region to get a list of voices for that region. For example, to get a list of voices for the `westus` region, use the `https://westus.tts.speech.microsoft.com/cognitiveservices/voices/list` endpoint. For a list of all supported regions, see the [regions](#) documentation.

ⓘ Note

Voice and styles in preview are only available in three service regions: East US, West Europe, and Southeast Asia.

Request headers

This table lists required and optional headers for text-to-speech requests:

Header	Description	Required or optional
<code>Ocp-Apim-Subscription-Key</code>	Your Speech resource 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 Authentication .	Either this header or <code>Ocp-Apim-Subscription-Key</code> is required.

Request body

A body isn't required for `GET` requests to this endpoint.

Sample request

This request requires only an authorization header:

```
HTTP  
  
GET /cognitiveservices/voices/list HTTP/1.1  
  
Host: westus.tts.speech.microsoft.com  
Ocp-Apim-Subscription-Key: YOUR RESOURCE KEY
```

Here's an example curl command:

```
curl --location --request GET  
'https://YOUR_RESOURCE_REGION.tts.speech.microsoft.com/cognitiveservices/voices/list' \  
--header 'Ocp-Apim-Subscription-Key: YOUR RESOURCE KEY'
```

Sample response

You should receive a response with a JSON body that includes all supported locales, voices, gender, styles, and other details. The `WordsPerMinute` property for each voice can be used to estimate the length of the output speech. This JSON example shows partial results to illustrate the structure of a response:

```
"ShortName": "en-US-JennyNeural",
"Gender": "Female",
"Locale": "en-US",
"LocaleName": "English (United States)",
"StyleList": [
    "assistant",
    "chat",
    "customerservice",
    "newscast",
    "angry",
    "cheerful",
    "sad",
    "excited",
    "friendly",
    "terrified",
    "shouting",
    "unfriendly",
    "whispering",
    "hopeful"
],
"SampleRateHertz": "24000",
"VoiceType": "Neural",
>Status": "GA",
"ExtendedPropertyMap": {
    "IsHighQuality48K": "True"
},
"WordsPerMinute": "152"
},
// Redacted for brevity
{
    "Name": "Microsoft Server Speech Text to Speech Voice (en-US,
JennyMultilingualNeutral)",
    "DisplayName": "Jenny Multilingual",
    "LocalName": "Jenny Multilingual",
    "ShortName": "en-US-JennyMultilingualNeutral",
    "Gender": "Female",
    "Locale": "en-US",
    "LocaleName": "English (United States)",
    "SecondaryLocaleList": [
        "de-DE",
        "en-AU",
        "en-CA",
        "en-GB",
        "es-ES",
        "es-MX",
        "fr-CA",
        "fr-FR",
        "it-IT",
        "ja-JP",
        "ko-KR",
        "pt-BR",
        "zh-CN"
    ],
    "SampleRateHertz": "24000",
    "VoiceType": "Neural",
    "Status": "GA",
    "WordsPerMinute": "190"
},
// Redacted for brevity
{
    "Name": "Microsoft Server Speech Text to Speech Voice (ga-IE, OrlaNeural)",
    "DisplayName": "Orla",
```

```

        "LocalName": "Orla",
        "ShortName": "ga-IE-OrlaNeural",
        "Gender": "Female",
        "Locale": "ga-IE",
        "LocaleName": "Irish (Ireland)",
        "SampleRateHertz": "24000",
        "VoiceType": "Neural",
        "Status": "GA",
        "WordsPerMinute": "139"
    },
    // Redacted for brevity
{
    "Name": "Microsoft Server Speech Text to Speech Voice (zh-CN, YunxiNeural)",
    "DisplayName": "Yunxi",
    "LocalName": "云希",
    "ShortName": "zh-CN-YunxiNeural",
    "Gender": "Male",
    "Locale": "zh-CN",
    "LocaleName": "Chinese (Mandarin, Simplified)",
    "StyleList": [
        "narration-relaxed",
        "embarrassed",
        "fearful",
        "cheerful",
        "disgruntled",
        "serious",
        "angry",
        "sad",
        "depressed",
        "chat",
        "assistant",
        "newscast"
    ],
    "SampleRateHertz": "24000",
    "VoiceType": "Neural",
    "Status": "GA",
    "RolePlayList": [
        "Narrator",
        "YoungAdultMale",
        "Boy"
    ],
    "WordsPerMinute": "293"
},
    // Redacted for brevity
]

```

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 reason is a header that's too long.
401	Unauthorized	The request is not authorized. Make sure your resource 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 resource.
502	Bad gateway	There's a network or server-side problem. This status might also indicate invalid headers.

Convert text to speech

The `cognitiveservices/v1` endpoint allows you to convert text to speech by using [Speech Synthesis Markup Language \(SSML\)](#).

Regions and endpoints

These regions are supported for text-to-speech through the REST API. Be sure to select the endpoint that matches your Speech resource region.

Prebuilt neural voices

Use this table to determine *availability of neural voices* by region or endpoint:

Region	Endpoint
Australia East	https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1
Brazil South	https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/v1
Canada Central	https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1
Central US	https://centralus.tts.speech.microsoft.com/cognitiveservices/v1
East Asia	https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1
East US	https://eastus.tts.speech.microsoft.com/cognitiveservices/v1
East US 2	https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1
France Central	https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1
Germany West Central	https://germanywestcentral.tts.speech.microsoft.com/cognitiveservices/v1
India Central	https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1

Region	Endpoint
Japan East	https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1
Japan West	https://japanwest.tts.speech.microsoft.com/cognitiveservices/v1
Jio India West	https://jioindiawest.tts.speech.microsoft.com/cognitiveservices/v1
Korea Central	https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1
North Central US	https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1
North Europe	https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1
Norway East	https://norwayeast.tts.speech.microsoft.com/cognitiveservices/v1
South Central US	https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1
Southeast Asia	https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1
Sweden Central	https://swedencentral.tts.speech.microsoft.com/cognitiveservices/v1
Switzerland North	https://switzerlandnorth.tts.speech.microsoft.com/cognitiveservices/v1
Switzerland West	https://switzerlandwest.tts.speech.microsoft.com/cognitiveservices/v1
UAE North	https://uaenorth.tts.speech.microsoft.com/cognitiveservices/v1
US Gov Arizona	https://usgovarizona.tts.speech.azure.us/cognitiveservices/v1
US Gov Virginia	https://usgovvirginia.tts.speech.azure.us/cognitiveservices/v1
UK South	https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1
West Central US	https://westcentralus.tts.speech.microsoft.com/cognitiveservices/v1
West Europe	https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1
West US	https://westus.tts.speech.microsoft.com/cognitiveservices/v1
West US 2	https://westus2.tts.speech.microsoft.com/cognitiveservices/v1
West US 3	https://westus3.tts.speech.microsoft.com/cognitiveservices/v1

💡 Tip

Voices in preview are available in only these three regions: East US, West Europe, and Southeast Asia.

Custom neural voices

If you've created a custom neural voice font, use the endpoint that you've created. You can also use the following endpoints. Replace `{deploymentId}` with the deployment ID for your neural voice model.

Region	Training	Deployment	Endpoint
Australia East	Yes	Yes	<code>https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Brazil South	No	Yes	<code>https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Canada Central	No	Yes	<code>https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Central US	No	Yes	<code>https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
East Asia	No	Yes	<code>https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
East US	Yes	Yes	<code>https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
East US 2	Yes	Yes	<code>https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
France Central	No	Yes	<code>https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Germany West Central	No	Yes	<code>https://germanywestcentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
India Central	Yes	Yes	<code>https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Japan East	Yes	Yes	<code>https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Japan West	No	Yes	<code>https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Jio India West	No	Yes	<code>https://jioindiawest.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Korea Central	Yes	Yes	<code>https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
North Central US	No	Yes	<code>https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
North Europe	Yes	Yes	<code>https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
Norway East	No	Yes	<code>https://norwayeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>
South Africa North	No	Yes	<code>https://southafricanorth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</code>

Region	Training	Deployment	Endpoint
South Central US	Yes	Yes	https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Southeast Asia	Yes	Yes	https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Switzerland North	No	Yes	https://switzerlandnorth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Switzerland West	No	Yes	https://switzerlandwest.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
UAE North	No	Yes	https://uaenorth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
UK South	Yes	Yes	https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West Central US	No	Yes	https://westcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West Europe	Yes	Yes	https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West US	Yes	Yes	https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West US 2	Yes	Yes	https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West US 3	No	Yes	https://westus3.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}

⚠ Note

The preceding regions are available for neural voice model hosting and real-time synthesis. Custom neural voice training is only available in some regions. But users can easily copy a neural voice model from these regions to other regions in the preceding list.

Long Audio API

The Long Audio API is available in multiple regions with unique endpoints:

Region	Endpoint
Australia East	https://australiaeast.customvoice.api.speech.microsoft.com
East US	https://eastus.customvoice.api.speech.microsoft.com
India Central	https://centralindia.customvoice.api.speech.microsoft.com

Region	Endpoint
South Central US	https://southcentralus.customvoice.api.speech.microsoft.com
Southeast Asia	https://southeastasia.customvoice.api.speech.microsoft.com
UK South	https://uksouth.customvoice.api.speech.microsoft.com
West Europe	https://westeurope.customvoice.api.speech.microsoft.com

Request headers

This table lists required and optional headers for text-to-speech requests:

Header	Description	Required or 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 provided value must be fewer than 255 characters.	Required

Request body

If you're using a custom neural voice, the body of a request can be sent as plain text (ASCII or UTF-8). Otherwise, the body of each `POST` request is sent as [SSML](#). SSML allows you to choose the voice and language of the synthesized speech that the text-to-speech feature returns. For a complete list of supported voices, see [Language and voice support for the Speech service](#).

Sample request

This HTTP request uses SSML to specify the voice and language. If the body length is long, and the resulting audio exceeds 10 minutes, it's truncated to 10 minutes. In other words, the audio length can't exceed 10 minutes.

HTTP
<pre>POST /cognitiveservices/v1 HTTP/1.1 X-Microsoft-OutputFormat: riff-24khz-16bit-mono-pcm Content-Type: application/ssml+xml Host: westus.tts.speech.microsoft.com Content-Length: <Length> Authorization: Bearer [Base64 access_token] User-Agent: <Your application name></pre>

```

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Male'
    name='en-US-ChristopherNeural'>
        Microsoft Speech Service Text-to-Speech API
    </voice></speak>

```

* For the Content-Length, you should use your own content length. In most cases, this value is calculated automatically.

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 reason is a header that's too long.
401	Unauthorized	The request is not authorized. Make sure your Speech resource key or token is valid and in the correct region.
415	Unsupported media type	It's possible that the wrong <code>Content-Type</code> value was provided. <code>Content-Type</code> should be set to <code>application/ssml+xml</code> .
429	Too many requests	You have exceeded the quota or rate of requests allowed for your resource.
502	Bad gateway	There's a network or server-side problem. This status might 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.

Audio outputs

The supported streaming and non-streaming audio formats are sent in each request as the `X-Microsoft-OutputFormat` header. Each format incorporates a bit rate and encoding type. The Speech service supports 48-kHz, 24-kHz, 16-kHz, and 8-kHz audio outputs. Each prebuilt neural voice model is available at 24kHz and high-fidelity 48kHz.

Streaming

```

amr-wb-16000hz
audio-16khz-16bit-32kbps-mono-opus
audio-16khz-32kbitrate-mono-mp3

```

```
audio-16khz-64kbitrate-mono-mp3
audio-16khz-128kbitrate-mono-mp3
audio-24khz-16bit-24kbps-mono-opus
audio-24khz-16bit-48kbps-mono-opus
audio-24khz-48kbitrate-mono-mp3
audio-24khz-96kbitrate-mono-mp3
audio-24khz-160kbitrate-mono-mp3
audio-48khz-96kbitrate-mono-mp3
audio-48khz-192kbitrate-mono-mp3
ogg-16khz-16bit-mono-opus
ogg-24khz-16bit-mono-opus
ogg-48khz-16bit-mono-opus
raw-8khz-8bit-mono-alaw
raw-8khz-8bit-mono-mulaw
raw-8khz-16bit-mono-pcm
raw-16khz-16bit-mono-pcm
raw-16khz-16bit-mono-truesilk
raw-22050hz-16bit-mono-pcm
raw-24khz-16bit-mono-pcm
raw-24khz-16bit-mono-truesilk
raw-44100hz-16bit-mono-pcm
raw-48khz-16bit-mono-pcm
webm-16khz-16bit-mono-opus
webm-24khz-16bit-24kbps-mono-opus
webm-24khz-16bit-mono-opus
```

ⓘ Note

If you select 48kHz output format, the high-fidelity voice model with 48kHz will be invoked accordingly. The sample rates other than 24kHz and 48kHz can be obtained through upsampling or downsampling when synthesizing, for example, 44.1kHz is downsampled from 48kHz.

If your selected voice and output format have different bit rates, the audio is resampled as necessary. You can decode the `ogg-24khz-16bit-mono-opus` format by using the [Opus codec ↗](#).

Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each feature:

Supported authorization header	Speech-to-text	Text-to-speech
<code>Ocp-Apim-Subscription-Key</code>	Yes	Yes
<code>Authorization: Bearer</code>	Yes	Yes

When you're using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your resource key. For example:

HTTP

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When you're using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your resource key for an access token that's valid for 10 minutes.

How to get an access token

To get an access token, you need to make a request to the `issueToken` endpoint by using `Ocp-Apim-Subscription-Key` and your resource key.

The `issueToken` endpoint has this format:

HTTP

```
https://<REGION_IDENTIFIER>.api.cognitive.microsoft.com/sts/v1.0/issueToken
```

Replace `<REGION_IDENTIFIER>` with the identifier that matches the [region](#) of your subscription.

Use the following 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 resource key for the Speech service. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

HTTP

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: eastus.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 resource key for the Speech service. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

PowerShell

```
$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://eastus.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 resource key for the Speech service. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

Console

```
curl -v -X POST \
"https://eastus.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 resource key for the Speech service 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.

C#

```
public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://eastus.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

Python

```

# 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://eastus.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, but to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the speech-to-text REST API for short audio:

HTTP

```

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...

```

Next steps

- [Create a free Azure account ↗](#)
- [Get started with custom neural voice](#)
- [Batch synthesis](#)

Additional resources

Documentation

[Text-to-speech quickstart - Speech service - Azure Cognitive Services](#)

In this quickstart, you convert text to speech. Learn about object construction and design patterns, supported audio output formats, and custom configuration options for speech synthesis.

[Text-to-speech overview - Speech service - Azure Cognitive Services](#)

Get an overview of the benefits and capabilities of the text-to-speech feature of the Speech service.

[Voice and sound with Speech Synthesis Markup Language \(SSML\) - Speech service - Azure Cognitive Services](#)

Learn about Speech Synthesis Markup Language (SSML) elements to determine what your output audio will sound like.

[Speech Synthesis Markup Language \(SSML\) overview - Speech service - Azure Cognitive Services](#)

Use the Speech Synthesis Markup Language to control pronunciation and prosody in text-to-speech.

[Install the Speech SDK - Azure Cognitive Services](#)

In this quickstart, you'll learn how to install the Speech SDK for your preferred programming language.

[Regions - Speech service - Azure Cognitive Services](#)

A list of available regions and endpoints for the Speech service, including speech-to-text, text-to-speech, and speech translation.

[Speech Synthesis Markup Language \(SSML\) document structure and events - Speech service - Azure Cognitive Services](#)

Learn about the Speech Synthesis Markup Language (SSML) document structure.

[Audio Content Creation - Speech service - Azure Cognitive Services](#)

Audio Content Creation is an online tool that allows you to run text-to-speech synthesis without writing any code.

[Show 5 more](#)

Speaker Recognition

Article • 09/13/2022 • 4 minutes to read

The Azure Cognitive Service Speaker Recognition service provides algorithms that verify and identify speakers by their unique voice characteristics. Speaker Recognition is used to answer the question "who is speaking?". [Learn more ↗](#).

Voice has unique characteristics that can be associated with an individual. We provide Speaker Verification APIs and Speaker Identification APIs for two major applications of Speaker Recognition technologies.

Speaker Verification

Speaker verification can be either text-dependent or text-independent. Text-dependent verification means speakers need to choose the same passphrase to use during both enrollment and verification phases. The verification of both speech content and voice signature facilitates a multi-factor verification scenario; Text-independent verification means speakers can speak in everyday language in the enrollment and verification phrases.

Text Dependent Speaker Verification

In the speaker enrollment phase, the speaker's voice is recorded by saying a passphrase from a set of predefined phrases. Voice features are extracted from the audio recording to form a unique voice signature while the chosen passphrase is recognized. Together, the voice signature and the passphrase would be used to verify the speaker.

In the verification phase, the ID associated with the individual to be verified is sent to the speaker verification API. The speaker verification service extracts voice features and the passphrase from the input speech recording. Then it compares the voice features and the passphrase against the enrollment profile of the corresponding speaker.

The response returns "Accept" or "Reject" with a similarity score ranging from 0 to 1. The "Accept" or "Reject" response is a result combining both the speaker verification result and speech recognition result, while the similarity score only measures the voice similarity. We return "Accept" when the speech recognition result matches the enrollment phrase and the voice similarity score is greater or equal to 0.5. However, the result should be determined based on the scenario and other verification factors that are being used. We recommend you experiment on your own data and determine your threshold to override "Accept" or "Reject" response as appropriate.

In current version of text-dependent speaker verification API, we provide 10 English phrases for the speakers to choose from.

- I am going to make him an offer he cannot refuse.
- Houston we have had a problem.
- My voice is my passport verify me.
- Apple juice tastes funny after toothpaste.
- You can get in without your password.
- You can activate security system now.
- My voice is stronger than passwords.
- My password is not your business.
- My name is unknown to you.
- Be yourself everyone else is already taken"

You can create your own passphrases by sending separate requests to the text-independent speaker verification API and speech-to-text API. Combining the speaker verification result and speech recognition result, you can determine the speaker's identity.

The APIs are not intended to determine whether the audio is from a live person or an imitation or a recording of an enrolled speaker. Generating random phrases for the speaker to read is considered effective to prevent replay attack.

Text Independent Speaker Verification

Speaker Verification can also be text-independent, which means that there are no restrictions on what the speaker says in the audio.

In the enrollment phase, voice features are extracted from a speaker's audio to form a unique voice signature.

In the verification phase, the audio and the ID associated with the individual to be verified are sent to the speaker verification API. The speaker verification service extracts voice features from the input speech recording. Then it compares the voice features against the voice signature in enrollment profile of the corresponding speaker.

The response returns "Accept" or "Reject" with a similarity score ranging from 0 to 1. The "Accept" response is returned when the similarity score is greater or equal to 0.5. However, the result should be determined based on the scenario and other verification factors that are being used. We recommend you experiment on your own data and determine your threshold to override "Accept" or "Reject" response as appropriate.

The APIs are not intended to determine whether the audio is from a live person or an imitation or a recording of an enrolled speaker.

Speaker Identification

Speaker identification is the task of determining the identity of an unknown voice among a set of candidate speakers. The Speaker Identification API returns a list of "best matches" based on the similarity scores against a provided list of IDs. The Speaker Identification API is text-independent as it does not compare what was said at the enrollment and recognition.

Text Independent Speaker Identification

Enrollment for speaker identification is text-independent, which means that there are no restrictions on what the speaker says in the audio. No passphrase is required. In the enrollment phase, the speaker's voice is recorded, and voice features are extracted to form a unique voice signature.

In the identification phase, the speaker identification service extracts voice features from the input speech recording. Then it compares the features against the voice signatures in the enrollment data of a specified list of speakers (up to 50 candidate speakers in each request). The response included one identified ID and five top-ranked IDs with similarity scores ranging from 0 to 1. The identified ID is determined based on the similarity score of the best matched speaker. If none of the candidate speakers returns a similarity score of greater or equal than 0.5, the response returns a string of zero to represent "no match is found". However, the result should be determined based on your scenario and other factors that are being used. We recommend you experiment with your data and determine your threshold to override the default "match or no match" as appropriate.

The APIs are not intended to determine whether the audio is from a live person or an imitation or a recording of an enrolled speaker.

See Also

- [Tutorial for text-independent speaker verification API](#) ↗

Use cases for Speech-to-Text

Article • 07/18/2022 • 8 minutes to read

What is a Transparency note?

An AI system includes not only the technology, but also the people who will use it, the people who will be affected by it, and the environment in which it is deployed. Creating a system that is fit for its intended purpose requires an understanding of how the technology works, its capabilities and limitations, and how to achieve the best performance. Microsoft's Transparency Notes are intended to help you understand how our AI technology works, the choices system owners can make that influence system performance and behavior, and the importance of thinking about the whole system, including the technology, the people, and the environment. You can use Transparency Notes when developing or deploying your own system, or share them with the people who will use or be affected by your system.

Microsoft's Transparency notes are part of a broader effort at Microsoft to put our AI principles into practice. To find out more, see [Microsoft's AI principles](#).

Introduction to Speech-to-Text

Speech-to-Text, also known as automatic speech recognition (ASR), is a feature under Speech Services (See [What is Speech-to-Text?](#)), that converts spoken audio into text. Speech-to-Text supports over 140 locales for inputs. Please see the latest list of supported locales in [Language and voice support for the Speech service](#).

Terms and definitions

Term	Definition
Audio input	The streamed audio data or audio file that is used as an input for the speech-to-text feature. Audio input may contain not only voice, but also silence and non-speech noise. Speech-to-Text generates text for the voice parts of audio input.
Utterance	A component of audio input that contains human voice. One utterance may consist of a single word or multiple words, such as a phrase.

Term	Definition
Transcription	The text output of the Speech-to-Text feature. This automatically generated text output leverages speech models (defined below) and is sometimes referred to as machine transcription or automated speech recognition (ASR). Transcription in this context is fully automated and therefore different from human transcription, which is text generated by human transcribers.
Speech model	An automatically generated, machine-learned numerical representation of an utterance used to infer a transcription from an audio input. Speech models are trained on voice data that includes various speech styles, languages, accents, dialects and intonations, as well as acoustic variations generated by different types of recording devices. A speech model numerically represents both acoustic and linguistic features, which are used to predict what text should be associated with the utterance.
Real-time API	An API that accepts requests with audio input, and returns a response in real time with transcription within the same network connection.
Language Detection API	A type of real-time API that detects what language is spoken in a given audio input. A language is inferred based on voice sound in the audio input.
Speech Translation API	Another type of real-time API that generates transcriptions of a given audio input then translates them into a language specified by the user. This is a cascaded service of Speech Services and Text Translator.
Batch API	A service to send audio input to be transcribed at a later time. Customers specify location(s) of audio file(s) with other parameters, such as a language name. The service loads the audio input asynchronously and transcribes it. Once transcription is complete, text file(s) are loaded back to a location specified by the customer.
Speaker Separation (diarization)	Speaker Separation (also called diarization) answers the question of who spoke when. It differentiates speakers in an audio input based on their voice characteristics. The Batch API supports diarization and is capable of differentiating two speakers' voices on mono channel recordings. Diarization is combined with speech to text functionality in order to provide transcription outputs that contain a speaker entry for each transcribed phrase. This diarization feature only supports the differentiation of two voices, and the transcription output is tagged as Speaker 1 or 2. If more than two speakers are within the audio, this will result in inaccurate tagging.

Speech-to-Text features

There are two ways to use the Speech-to-Text service - Real-time and Batch.

Real-time Speech-to-Text API

This is a common API call via Speech SDK/REST API to send an audio input and receive a text transcription in real time. The speech system uses a “speech model” to recognize what is spoken in a given input audio. A speech model consists of an acoustic model and a language model, which combine to calculate likely spoken content. The acoustic model maps a small linguistic unit called a “phoneme” to voice sound in an audio input, which leads to a probability of specific word or phrase. The language model calculates probabilities of word combinations in a specified language. The Speech-to-Text technology then infers a final phrase or text based on combined probabilities. This feature is often used, for example, for voice-enabled queries or dictation within a customer's service or application.

Sub-features and options of the Real-time Speech-to-Text API

- **Language Detection:** Unlike in a default API call, where a language (or a locale) for an audio input must be specified in advance, with language detection, customers can specify multiple locales and let the service detect a language.
- **Speech Translation:** This API converts audio input to text, and then translates and transcribes it into another language. The translated transcription output can be returned in text format or the customer may choose to have the text synthesized into audible speech by [Text-to-Speech \(TTS\)](#). See [What is the Translator service](#) for more information about the text translation service.

Batch transcription API

This is another type of API call, typically used to send lengthy audio inputs and to receive transcribed text asynchronously (i.e., at a later time). To use this API, users can specify locations of multiple audio files. The Speech-to-Text technology reads the audio input from the file streams and generates transcription text files which are returned to the customer's specified storage location. This feature is used to support larger transcription jobs (e.g., an audio broadcast for later text transcription) where it is not necessary to provide end users the transcription content in real time.

Sub-features and options of the Batch transcription API

- **Speaker Separation (diarization):** This feature is disabled by default. If a customer chooses to enable this feature, the service will differentiate between up to two speakers' utterances. The resulting transcription text contains a "speaker property" that indicates either Speaker 1 or Speaker 2, which denotes which of two speakers is speaking in an audio file. Speaker separation only works with two speakers. If audio input sent via the Batch API contains more than two speakers, the service will

still separate voice into two speakers, leading to errors in speaker tags tied to transcription output.

Example use cases

Speech-to-Text can offer more convenient or alternative ways for end users to interact with applications and devices. Instead of typing words on a keyboard or using their hands for touch screen interactions, Speech-to-Text technology allows users to operate applications and devices by voice and through dictation.

- **Smart Assistants** -- Companies developing smart assistants on appliances, cars, and homes can use Speech-to-Text to enable natural interface search queries or command-and-control features by voice.
- **Chat Bots** -- Companies can build chat bot applications, in which users can use voice enabled queries or commands to interact with bots.
- **Voice Typing** -- Apps may allow users to use voice to dictate long-form text. Voice typing can be used to enter text for texting, emails, and documents.
- **Voice Commanding** -- Users can trigger certain actions by voice. This is called "command and control", and typical examples are entering query text by voice and selecting a menu item by voice.
- **Voice translation** -- Customers can use speech translation features of Speech-to-Text technology to communicate by voice with other users who speak different languages. Speech translation enables voice-to-voice communication across multiple languages. See the latest list of supported locales in [Language and voice support for the Speech service](#).
- **Call Center transcriptions** -- Customers often record conversations with their end users in scenarios such as customer support calls. Audio recordings can be sent to the Batch API for transcription.
- **Mixed-language dictation** -- Customers can use Speech-to-Text technology to dictate in multiple languages. Using Language Detection, a dictation application may automatically detect spoken languages and transcribe appropriately without forcing users to specify which language they speak.

Considerations when choosing other use cases

Speech-to-Text API can offer users convenient options for enabling voice enabled applications, but it is very important to consider the context within which you will

integrate the API and ensure that you comply with all laws and regulations that apply to your application. This includes understanding your obligations under privacy and communication laws, including national and regional eavesdropping and wiretap laws, that apply to your jurisdiction. Only collect and process audio that is within the reasonable expectations of your users; this includes ensuring you have all necessary and appropriate permissions from users for your collection, processing and storage of audio data.

Many applications are designed and intended to be used by a specific individual user for voice enabled queries, commands or dictation, however, the microphone for your application may pick up sound or voice from non-primary users. To avoid capturing the voices of non-primary users unintentionally, you should take the following into consideration.

Microphone considerations: Often you cannot control who may speak around the input device that sends audio input to the Speech-to-Text cloud service. You should therefore encourage your end-users to take extra care when using voice enabled features and applications in a public/open environment where other people's voices may be easily captured and/or for allowing non-primary users to use the voice-enabled application.

Only use Speech-to-Text in experiences and features that are within the reasonable expectations of your users: Audio data of a person speaking is personal information. Speech-to-Text is not intended to be used for covert audio surveillance purposes, in a manner that violates legal requirements, or in applications and devices in public spaces or locations where users may have a reasonable expectation of privacy. Use speech services only to collect and process audio in ways that are within the reasonable expectations of your users; this includes ensuring you have all necessary and appropriate permissions from users for your collection, processing, and storage of audio data.

Next steps

- [Characteristics and Limitations for Speech-to-Text](#)

Characteristics and limitations of Speech-to-Text

Article • 07/18/2022 • 5 minutes to read

Speech-to-Text recognizes what's spoken in an audio input and generates transcription outputs. This requires proper setup for an expected language used in the audio input and spoken styles. Non-optimal settings may lead to lower accuracy.

Language of Accuracy

The industry standard to measure Speech-to-Text 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 correct transcript (often created by human labeling).

To understand the detailed WER calculation, please see [this document about Speech-to-Text evaluation and improvement](#).

Transcription Accuracy and System Limitations

Speech-to-Text uses a unified speech recognition machine learning model to understand what is spoken in a wide range of contexts and topic domains, such as command-and-control, or dictation and conversations. Therefore, you don't need to consider using different models for your application or feature scenarios.

However, you need to specify a language (or locale) for each audio input. This must match with actual language spoken in an input voice. Please check [the list of supported locales](#) for more details.

There are many factors that lead to a lower accuracy in transcription.

- **Acoustic quality:** Speech-to-Text enabled applications and devices may use a wide variety of microphone types and specifications. The unified speech models have been created based on various voice audio device scenarios, such as telephones, mobile phones and speaker devices. However, voice quality may be degraded by the way users speak to microphones, even if they use high-quality microphones. For example, if users stay far from a microphone, the input quality would be too low, while speaking too close to a microphone would cause audio quality deterioration. Both cases may adversely impact Speech-to-Text accuracy.

- **Non-speech noise:** If an input audio contains a certain level of noise, it has an impact on accuracy. Noise may come from the audio devices used to make a recording, while audio input itself may contain noise, such as background noise and environmental noise.
- **Overlapped speech:** There could be multiple speakers within range of an audio input device, and they may speak at the same time. Also, other speakers may speak in the background while the main user is speaking.
- **Vocabularies:** The Speech-to-Text model has knowledge of wide variety of words in many domains. However, users may speak company specific terms and jargon (which are "out of vocabulary"). If a word appears in the audio that doesn't exist in a model, it will result in a mis-transcription.
- **Accents:** Even within one locale (such as "English -- United States"), many people have different accents. Very particular accents may also lead to a mis-transcription.
- **Mismatched locales:** Users may not speak the languages you expect. If you specified English -- United States (en-US) for an audio input, but a user spoke Swedish, for instance, accuracy would be worse.

Due to these acoustic and linguistic variations, customers should expect a certain level of inaccuracy in the output text when designing an application.

Best practices to improve system accuracy

As discussed above, acoustic conditions, such as background noise, side speech, distance to microphone, speaking styles and characteristics can adversely affect the accuracy of what is being recognized.

Please consider the following application/service design principles for better speech experiences.

- **Design UIs to match input locales:** Mismatched locales will deteriorate accuracy. The Speech SDK supports [automatic language detection](#), but it only detects one out of four locales specified at runtime. You still need to understand in what locale your users will speak. Your user interfaces should clearly indicate which languages the users may speak by such measures as a drop-down list of languages allowed to be used. Please see [supported locales](#).
- **Allow users to try again:** Misrecognition may occur due to some temporary issues, such as unclear or fast speech or a long pause. If your application expects specific transcriptions, such as predefined action commands (such as "Yes" and "No"), and

did not get any of them, users should be able to try again. A typical method is to tell users "Sorry, I didn't get that. Please try again".

- **Confirm before taking an action by voice:** Just as with keyboard/click/tap-based user interfaces, if an audio input can trigger an action, users should be given an opportunity to confirm the action, especially by displaying or playing back what was recognized/transcribed. A typical example is sending a text message by voice. An app repeats what was recognized and asks for confirmation: "You said, 'Thank you'. Send it or change it?".
- **Add custom vocabularies:** The general speech recognition model provided by Speech-to-Text covers a broad vocabulary. However, scenario specific jargon and named entities (people names, product names, etc.) may be underrepresented and what words and phrases are likely to be spoken can vary significantly depending on the scenario. If you can anticipate which words and phrases will be spoken (for instance, when a user selects an item from a list), you may want to use the phrasal list grammar, see "Improving recognition accuracy" in [Get started with Speech-to-Text](#).
- **Use Custom Speech:** If Speech-to-Text accuracy in your application scenarios remains low, you might want to consider customizing the model for your acoustic and linguistic variations. You may create your own models by training with your own voice audio data or text data. See more details about [Custom Speech](#).

Fairness

At Microsoft, we strive to empower every person on the planet to achieve more. An essential part of this goal is working to create technologies and products that are fair and inclusive. Fairness is a multi-dimensional, sociotechnical topic and impacts many different aspects of our product development. You can learn more about Microsoft's approach to fairness [here ↗](#).

One dimension we need to consider is how well the system performs for different groups of people. Research has shown that without conscious effort focused on improving performance for all groups, it is often possible for the performance of an AI system to vary across groups based on factors such as race, ethnicity, region, gender and age.

Speech-to-Text models are trained and tuned with voice audio with variations including:

- Microphones and device specifications
- Speech environment

- Speech scenarios
- Languages and accents
- Age and genders
- Ethnic background

Each version of the Speech-to-Text model is tested and evaluated against various test sets to make sure the model can perform without a large gap in each of the evaluation criteria.

Evaluating Text-to-Speech in your applications

The performance of Text-to-Speech will vary depending on the real-world uses and conditions that customers implement. In order to ensure optimal performance in their scenarios, customers should conduct their own evaluations of the solutions they implement using Text-to-Speech.

A test voice dataset should consist of actual voice inputs collected in your applications in production. Customers should randomly sample data to reflect real user variations over a certain period of time. Additionally, the test dataset should be refreshed periodically to reflect changes in the variations.

Next steps

- [Guidance for integration and responsible use with Speech-to-Text](#)

Guidance for integration and responsible use of Speech-to-Text

Article • 07/18/2022 • 3 minutes to read

ⓘ Note

This article is provided for informational purposes only and not for the purpose of providing legal advice. We strongly recommend seeking specialist legal advice when implementing Speech Services.

Integration and responsible use

As Microsoft works to help customers responsibly develop and deploy solutions using **Speech-to-Text**, we are taking a principled approach to upholding personal agency and dignity by considering the AI systems' fairness, reliability & safety, privacy & security, inclusiveness, transparency, and human accountability. These considerations reflect our commitment to developing Responsible AI.

When getting ready to deploy AI-powered products or features, the following activities help to set you up for success:

- **Understand what it can do:** Fully assess the capabilities of any AI system you are using to understand its capabilities and limitations. Understand how it will perform in your particular scenario and context by thoroughly testing it with real life conditions and data.
- **Respect an individual's right to privacy:** Only collect data and information from individuals for lawful and justifiable purposes. Only use data and information that you have consent to use for this purpose.
- **Legal review:** Obtain appropriate legal advice to review your solution, particularly if you will use it in sensitive or high-risk applications. Understand what restrictions you might need to work within and your responsibility to resolve any issues that might come up in the future. Do not provide any legal advice or guidance.
- **Human-in-the-loop:** Keep a human-in-the-loop and include human oversight as a consistent pattern area to explore. This means ensuring constant human oversight of the AI-powered product or feature, and maintaining the role of humans in decision making. Ensure you can have real-time human intervention in the solution

to prevent harm. This enables you to manage situations when the AI model does not perform as required.

- **Security:** Ensure your solution is secure and has adequate controls to preserve the integrity of your content and prevent unauthorized access.
- **Build trust with affected stakeholders:** Communicate the expected benefits and potential risks to affected stakeholders. Help people understand why the data is needed and how the use of the data will lead to their benefit. Describe data handling in an understandable way.
- **Customer feedback loop:** Provide a feedback channel that allows users and individuals to report issues with the service once it's been deployed. Once you've deployed an AI-powered product or feature it requires ongoing monitoring and improvement -- be ready to implement any feedback and suggestions for improvement. Establish channels to collect questions and concerns from affected stakeholders (people who may be directly or indirectly impacted by the system, including employees, visitors, and the general public). Examples of such channels are:
 - Feedback features built into app experiences,
 - An easy-to-remember email address for feedback,
 - Anonymous feedback boxes placed in semi-private spaces, and
 - Knowledgeable representatives on site.
- **Feedback:** Seek out feedback from a diverse sampling of the community during the development and evaluation process (for example, historically marginalized groups, people with disabilities, and service workers). See: [Community Jury](#).
- **User Study:** Any consent or disclosure recommendations should be framed in a user study. Evaluate the first and continuous-use experience with a representative sample of the community to validate that the design choices lead to effective disclosure. Conduct user research with 10-20 community members (affected stakeholders) to evaluate their comprehension of the information and to determine if their expectations are met.

Recommendations for Preserving Privacy

A successful privacy approach empowers individuals with information and provides controls and protection to preserve their privacy.

Consent to process and store audio input: Be sure to have all necessary permissions from your end-users before using the Speech-to-Text enabled features in your

applications or devices. Also ensure you have permission for Microsoft to process this data as your third-party cloud service processor. Note that the real-time API does not separately store any of the audio input and transcription output data. However, you may design your application or device to retain end-user data, such as transcription text. You have an option to turn on local data logging via the Speech SDK (See [Enabling logging in the Speech SDK](#)).

Next steps

- [Data, privacy, and security for Speech-to-Text](#)

Data and Privacy for Speech-to-Text

Article • 09/26/2022 • 5 minutes to read

ⓘ Note

This article is provided for informational purposes only and not for the purpose of providing legal advice. We strongly recommend seeking specialist legal advice when implementing Speech Services.

This article provides some high-level details regarding how Speech-to-Text processes data provided by Customers. Note that audio data of humans speaking and the related text transcripts may be considered personal data and/or sensitive data under various privacy regulations and laws because it contains not only the voice of humans, but the content of the audio may also contain personal information depending on the context within which the audio was collected. Audio data and the related text transcripts may also be regulated under various communications laws or other law and regulations. As an important reminder, you are responsible for the implementation of this technology and are required to obtain all necessary permissions for processing of the data, as well as any licenses, permissions or other proprietary rights required for the content you input into the speech to text service. It is your responsibility to comply with all applicable laws and regulations in your jurisdiction.

What data does Speech-to-Text process?

Speech-to-Text processes the following types of data:

- **Audio input or voice audio:** All Speech-to-Text features accept voice audio as an input that is streamed via Speech SDK/REST API into the service endpoint. In batch transcription, audio input will be sent to a storage location instructed by the customer, and the Speech Service accesses and processes the audio input for the purposes of providing the transcription services requested. See more information about how to specify storage in [How to use batch transcription](#).
- **Input transcription text:** In the pronunciation assessment, transcribed text is sent together with an input voice audio as "correct" text. Pronunciations are assessed based on the input transcriptions.
- **Transcription for speech translation:** When the speech translation feature is used, transcribed text that Speech-to-Text generated is translated into a specified language through [Translator Service](#).

The text translation service is used only to convert text from one language to another. No input/output data is retained by Speech Service after completion of translation request. See [What is the Translator service](#) for more information about the text translation service.

If users need transcribed/translated text in an audio format, the feature sends the output text to [Text-to-Speech \(TTS\)](#). Again, no data is persisted in the TTS data processing.

How does Speech-to-Text process data?

Real-time Speech-to-Text

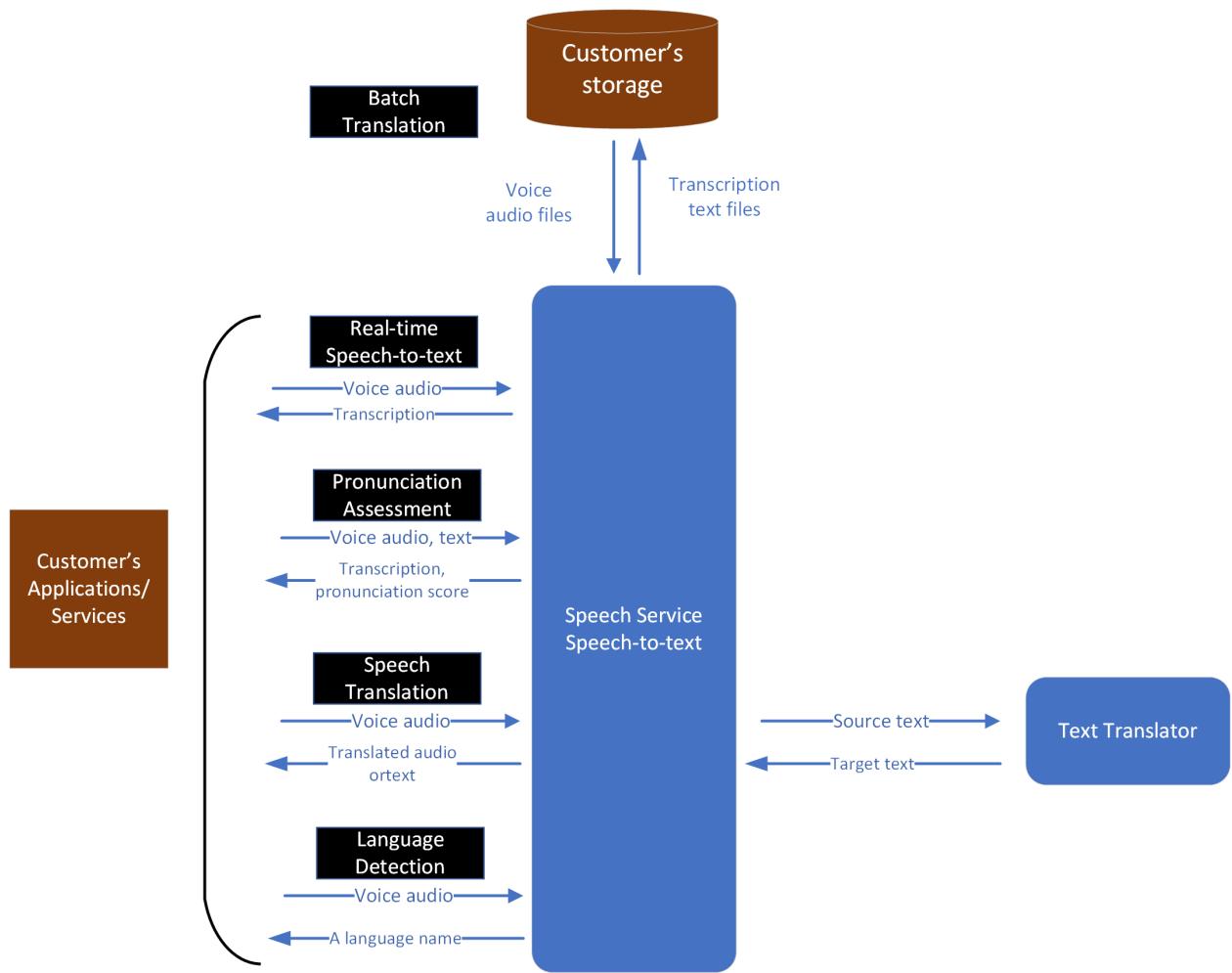
When a client application sends audio input to Speech-to-Text, the speech recognition engine parses audio and converts it to text. Relying upon its acoustic and linguistic or language understanding features, Speech-to-Text selects candidate words and phrases that may be uttered in the audio input. The transcription output represents the best inference or prediction in text format of what was spoken in the audio input.

For real-time Speech-to-Text, audio input is processed only on the Azure's server memory, and no data is stored at rest. All data in-transit are encrypted for protection. See [Trusted Cloud: security, privacy, compliance, resiliency, and IP](#) for more information about Azure-wide security and privacy protection.

Batch transcription

In batch transcription, customers specify their chosen storage location of both audio input and output transcription text files for Speech Services to access, process, and provide the transcription output . The Customer controls the storage of this data, including the retention of such data. Customers may set a retention time for generated transcription text files by using a parameter called "timeToLive". See [Batch Transcription -- Configuration Properties](#) for more detail.

Please see the data flows for each Speech-to-Text feature:



Speaker Separation/Diarization

This feature is available for the Batch API only. When customers enable the speaker separation (diarization) option (disabled by default), the Speech-to-Text engine analyzes and extracts unique voice characteristics signals from the audio input to differentiate the audio between two speakers. These voice characteristics signals are used and temporarily retained for the sole purpose of annotating the transcription output with markers next to text for Speaker 1 or Speaker 2. Upon completion of the batch process, all signal data used to separate the speakers is discarded. The speaker separation feature only supports the separation of two speakers in a single audio file. Speaker Separation does not support speaker identity recognition enrollment or the ability to track unique speakers across multiple audio files.

Language Detection

Language detection is similar to speech recognition except that the model calculates probabilities of mapping between phonemes and languages. Each language has specific phonemes and phoneme combinations, which characterize the language. The language detection model identifies the characteristics in phonemes to calculate likelihood of languages used in an input voice.

Speech Translation

When Speech Translation is used, first, an audio input is used to generate machine-transcribed text with Speech-to-Text. Then the machine-transcribed text is sent to the text translation service to convert the text (in the source language) to another language. If customers need translated text in an audio format, this feature can send the translated text to [text-to-speech \(TTS\)](#). Customers have the option to produce translated text only or translated voice output.

Speech Containers

With Speech Containers, customers deploy Speech Services APIs to their own environment via Docker containers. Since all speech components run on customers' controlled environment, audio data inputs and transcription outputs are processed within Customers' container and is not sent to the cloud based Speech Service. See [Install and run Docker containers for the Speech service APIs](#) for more information.

Security for customers' data in Speech Container

The security of customer data is a shared responsibility. Details on the security model of Azure Cognitive Services containers, like the speech container can be found here - [Azure Cognitive Services container security](#).

You are responsible for securing and maintaining the equipment and infrastructure required to operate speech containers located on your premises, such as your edge device and network.

To learn more about Microsoft's privacy and security commitments visit [the Microsoft Trust Center](#).

Data storage and retention

No data trace

When using real-time Speech-to-Text, pronunciation assessment and speech translation, Microsoft does not retain or store the data provided by customers. In batch transcription, customers specify their own storage locations to send the audio input. Generated transcription text may be stored either in customer's own storage or Microsoft storage if no storage is specified. If output transcriptions are stored in Microsoft storage, Customers may delete the data either by calling a deletion API or

setting the timeToLive parameter to automatically delete the data in a specified time. See more details in [How to use batch transcription - Speech service - Azure Cognitive Services](#).

To learn more about Microsoft's privacy and security commitments visit the Microsoft [\[Trust Center\]{.ul}](#) .

Transparency Note

Article • 06/08/2022 • 6 minutes to read

An AI system includes not only the technology, but also the people who will use it, the people who will be affected by it, and the environment in which it is deployed. Creating a system that is fit for its intended purpose requires an understanding of how the technology works, its capabilities and limitations, and how to achieve the best performance.

Microsoft provides *Transparency Notes* to help you understand how our AI technology works. This includes the choices system owners can make that influence system performance and behavior, and the importance of thinking about the whole system, including the technology, the people, and the environment. You can use Transparency Notes when developing or deploying your own system, or share them with the people who will use or be affected by your system.

Transparency Notes are part of a broader effort at Microsoft to put our AI principles into practice. To find out more, see [Microsoft's AI principles](#).

Introduction to Pronunciation Assessment

The [Pronunciation Assessment](#) API takes both audio and reference text as inputs to evaluate speech pronunciation and gives speakers feedback on the accuracy and fluency of spoken audio, by comparing a machine generated transcript of the input audio with the reference text. The feature currently supports released en-US language and [other languages](#) in preview.

With Pronunciation Assessment, language learners can practice, get instant feedback, and improve their pronunciation so that they can speak and present with confidence. Educators can use Pronunciation Assessment to evaluate the pronunciation of multiple speakers in real time.

The basics of Pronunciation Assessment

The Pronunciation Assessment API provides speech evaluation results compared to reference text by using a machine learning-based approach that correlates highly with speech assessments conducted by native experts. The pronunciation Assessment model was trained with 100,000+ hours of speech data from native speakers. It can provide accurate results when people miss, repeat, or add phrases compared to the reference text. It also enables rich configuration parameters to support flexibility in using the API,

such as setting **Granularity** to change information granularity in evaluation. (For more information, please see more in [sample code](#)).

Pronunciation Assessment evaluates three aspects of pronunciation: accuracy, fluency, and completeness. It also provides evaluations at multiple levels of granularity and returns accuracy scores for specific phonemes, syllables, words, sentences, or even whole articles. For more information, see [how to use Speech SDK for Pronunciation Assessment](#).

The following table describes the key results. For more information, see the full [response parameters](#). By using [natural language processing \(NLP\)](#) techniques and the **EnableMiscue** settings, Pronunciation Assessment can detect errors such as extra, missing, or repeated words when compared to the reference text. This information helps obtain more accurate scoring to be used as diagnosis information. This capability is useful for longer paragraphs of text.

Parameter	Description
AccuracyScore	Gives a 0-5 score on pronunciation accuracy of the speech with phoneme, syllable, word, or full-text-level granularity. Accuracy indicates how closely the speech matches a native speaker's pronunciation in different granularities, from phoneme to full text.
FluencyScore	Gives a 0-5 score on fluency of the speech. Fluency indicates how closely the speech matches a native speaker's use of silent breaks between words.
CompletenessScore	Gives a 0-5 score on completeness of the speech. Determined by calculating the ratio of pronounced words to the reference text input.
ErrorType	This value indicates whether a word is omitted, inserted, or mispronounced compared to the reference text. Possible values are None (meaning no error on this word), Omission, Insertion, Repetition, and Mispronunciation.

Another set of parameters returned by Pronunciation Assessment are Offset and Duration (referred to together as the "timestamp") The timestamp of speech is returned in structured JSON format. Pronunciation Assessment can calculate pronunciation errors on each phoneme. Pronunciation Assessment can also flag the errors to specific timestamps in the input audio. Customers developing applications can use the signal to offer a learning path to help students focus on the error in multiple ways. For example, the application can highlight the original speech, reply to the audio to compare it with standard pronunciation, or recommend similar words to practice with.

Parameter	Description
-----------	-------------

Parameter	Description
Offset	The time (in 100-nanosecond units) at which the recognized speech begins in the audio stream.
Duration	The duration (in 100-nanosecond units) of the recognized speech in the audio stream.

Example use cases

Pronunciation Assessment can be used for [remote learning](#), exam practice, or other scenarios that demand pronunciation feedback. The following examples are use cases that are deployed or that we've designed for customers using pronunciation Assessment:

- **Educational service provider:** Providers can build applications with the use of Pronunciation Assessment to help students practice language learning remotely with real-time feedback. This use case is typical when an application needs to support real-time feedback. We support [streaming upload](#) on audio files for immediate feedback.
- **Education in a game:** App developers, for example, can build a language learning app by combining comprehensive lessons in games with state-of-the-art speech technology to help children learn English. The program can cover a wide range of English skills, such as speaking, reading, and listening, and also train children on grammar and vocabulary, with Pronunciation Assessment used to support children as they learn to speak English. These multiple learning formats ensure that children learn English with ease based on a fun learning style.
- **Education in a communication app:** Microsoft Teams Reading Progress assists the teacher in evaluating a student's speaking assignment with autodetection assistance for omission, insertion, and mispronunciation. It also enables students to practice pronunciation more conveniently before they submit their homework.

Considerations when choosing other use cases

Online learning has grown rapidly as schools and organizations adapt to new ways of connecting and methods of education. Speech technology can play a significant role in making distance learning more engaging and accessible to students of all backgrounds. With Azure Cognitive Services, developers can quickly add speech capabilities to applications, bringing online learning to life.

One key element in language learning is improving pronunciation skills. For new language learners, practicing pronunciation and getting timely feedback are essential to becoming a more fluent speaker. For the solution provider that seeks to support learners or students in language learning, the ability to practice anytime, anywhere by using Pronunciation Assessment would be a good fit for this scenario. It can also be integrated as a virtual assistant for teachers and help to improve their efficiency.

The following recommendations pertain to use cases where Pronunciation Assessment should be used carefully:

- **Include a human-in-the-loop for any formal examination scenarios:** Pronunciation Assessment system is powered by AI systems, and external factors like voice quality and background noise may impact the accuracy. A human-in-the-loop in formal examinations ensures the assessment results are as expected.
- **Consider using different thresholds per scenario:** Currently, the Pronunciation Assessment score only represents the similarity distance to the native speakers used to train the model. Such similarity distance can be mapped toward different scenarios with rule-based conditions or weighted counting to help provide pronunciation feedback. For example, the grading method for children's learning might not be as strict as that for adult learning. Consider setting a higher mispronunciation detection threshold for adult learning.
- **Consider the ability to account for miscues:** When the scenario involves reading long paragraphs, users are likely to find it hard to follow the reference text without making mistakes. These mistakes, including omission, insertion, and repetition, are counted as miscues. With **EnableMiscue** enabled, the pronounced words will be compared to the reference text, and will be marked with Omission, Insertion, Repetition based on the comparison.

Characteristics and limitations of Pronunciation Assessment

Article • 06/08/2022 • 3 minutes to read

As a part of the Azure Cognitive Services Speech service, pronunciation assessment empowers end-to-end education solutions for computer-assisted language learning. Pronunciation Assessment involves multiple criteria to assess learners' performance at multiple levels of detail, with perceptions similar to human judges.

How accurate is Pronunciation Assessment?

Pronunciation Assessment feature provides objective scores, like pronunciation accuracy and fluency degree, for language learners in [computer-assisted language learning ↗](#). The performance of pronunciation assessment depends on Azure Cognitive Services Speech-To-Text transcription accuracy with the use of a submitted transcription as reference, and [inter-rater agreement ↗](#) between the system and human judges. For a definition of Speech-To-Text accuracy, see [Characteristics and limitations for using speech-to-text ↗](#).

The following sections are designed to help you understand key concepts about accuracy as they apply to using Pronunciation Assessment.

The language of accuracy

The accuracy of Speech-To-Text affects pronunciation assessment. [Word error rate ↗](#) (WER) is used to measure Speech-To-Text accuracy as the industry standard. WER counts the number of incorrect words identified during recognition and then divides by the total number of words provided in the correct transcript, which is often created by human labeling.

Comparing Pronunciation Assessment to Human Judges

The [Pearson correlation coefficient ↗](#) is used to measure the correlation between pronunciation assessment API generated scores and scores generated by human judges. The Pearson correlation coefficient is a measure of linear correlation for two given sequences. It's widely used to measure the difference between automatically generated machine results and human-annotated labels. This coefficient assigns a value between –

1 to 1, where 0 is no correlation, negative value means the prediction is opposite to the target, and positive value means how prediction is aligned with the target.

The proposed guidelines for a Pearson correlation coefficient interpretation are shown in the following table. The strength signifies the relationship correlation between two variables and reflects how consistently the machine result aligns with human labels. Values that are close to 1 indicate a stronger correlation.

Strength of Association	Coefficient Value	Detail
Low	0.1 to 0.3	The autogenerated scores from an automatic system aren't significantly aligned with the perception of humans.
Medium	0.3 to 0.5	The autogenerated scores from an automatic system are aligned with the perception of humans, but differences still exist, and people might not agree with the result.
High	0.5 to 1.0	The autogenerated scores from an automatic system are aligned with the perception of humans, and people are willing to agree with the system results.

In our evaluations, Microsoft Pronunciation Assessment has performed >0.5 Pearson correlation with human judges' results, which indicates the autogenerated results are highly consistent with the judgment of human experts.

System limitations and best practices to improve system accuracy

- Pronunciation Assessment works better with higher-quality audio input. We recommend an input quality of 16 kHz or higher.
- Pronunciation Assessment quality is also affected by the distance of the speaker from the microphone. Recordings should be made with the speaker close to the microphone, and not over a remote connection.
- Pronunciation Assessment doesn't support a mixed lingual assessment scenario.
- Pronunciation Assessment supports released en-US language and [other languages](#) in preview.
- Pronunciation Assessment doesn't support a multi-speaker assessment scenario. The audio should include only one speaker for each assessment.
- Pronunciation Assessment compares the submitted audio to native speakers in general conditions. The speaker should maintain a normal speaking speed and volume, and avoid shouting or otherwise raising their voice.

- Pronunciation assessment performs better in an environment with little background noise. Current Speech-To-Text models accommodate noise in general conditions. Noisy environments or multiple people speaking at the same time might lead to lower confidence of the evaluation. To handle difficult cases better, you can suggest that the speaker should repeat a pronunciation if they score below a certain threshold.

Evaluating Pronunciation Assessment in your applications

Pronunciation Assessment's performance will vary depending on the real-world uses that customers implement. In order to ensure optimal performance in their scenarios, customers should conduct their own evaluations of the solutions they implement using Pronunciation Assessment.

- Before using Pronunciation Assessment in your applications, consider whether this product performs well in your scenario. Collect real-life data from the target scenario, test how Pronunciation Assessment performs, and make sure Speech-To-Text and Pronunciation Assessment can deliver the accuracy you need, see [Evaluate and improve Azure Cognitive Services Custom Speech accuracy](#).
- Select suitable thresholds per the target scenario. Pronunciation Assessment provides accuracy scores at different levels and you may need to consider the threshold employed in real-use. For example, the grading method for children's learning might not be as strict as that for adult learning. Consider setting a higher mispronunciation detection threshold for adult learning.

Transparency note and use cases for Custom Neural Voice

Article • 10/07/2022 • 4 minutes to read

This Transparency Note discusses Custom Neural Voice and the key considerations for making use of this technology responsibly.

What is a Transparency note?

An AI system includes not only the technology, but also the people who will use it, the people who will be affected by it, and the environment in which it is deployed. Creating a system that is fit for its intended purpose requires an understanding of how the technology works, its capabilities and limitations, and how to achieve the best performance. Microsoft's Transparency Notes are intended to help you understand how our AI technology works, the choices system owners can make that influence system performance and behavior, and the importance of thinking about the whole system, including the technology, the people, and the environment. You can use Transparency Notes when developing or deploying your own system, or share them with the people who will use or be affected by your system.

Microsoft's Transparency notes are part of a broader effort at Microsoft to put our AI principles into practice. To find out more, see [Responsible AI principles from Microsoft](#).

Introduction to Custom Neural Voice

Custom Neural Voice is a [Text-to-Speech](#) (TTS) feature, part of Speech Service in Azure Cognitive Services, that allows customers to create a one-of-a-kind customized synthetic voice for their applications by providing their own audio data of their selected voice talents. For more information on Custom Neural Voice, see [Overview of Custom Neural Voice](#).

Limited Access to Custom Neural Voice

Custom Neural Voice is a Limited Access service, and registration is required for access to some features. To learn more about Microsoft's Limited Access policy visit aka.ms/limitedaccesscogservices. Certain features are only available to Microsoft

managed customers and partners, and only for certain use cases selected at the time of registration.

Approved use cases

The following use cases are approved for customers:

- **Media: Educational or interactive learning:** For use to create a fictional brand or character voice for reading or speaking educational materials, online learning, interactive lesson plans, simulation learning, standardized testing, or guided museum tours.
- **Media: Entertainment:** For use to create a fictional brand or character voice for reading or speaking entertainment content for video games, movies, TV, recorded music, podcasts, audio books, or augmented or virtual reality.
- **Media: Journalistic or news:** For use to create voices for reading news or journalistic content that must be accompanied by a published, text version of the same content.
- **Media: Marketing:** For use to create a fictional brand or character voice for reading or speaking marketing and product or service media, product introductions, business promotion, or advertisements.
- **Media: Self-authored content:** For use to create a voice for reading content authored by the voice talent except where the voice is used to enhance the authority or credibility of the content in connection with financial, health, legal, political, or spiritual matters.
- **Accessibility Features:** For use in audio description systems, narration, or to facilitate communication by speaking impaired individuals.
- **Interactive Voice Response (IVR) Systems:** For use to create voices for call center operations, telephony systems, or responses for phone interactions.
- **Public Service Announcement:** For use to create a fictional brand or character voice for announcements for public venues.
- **Translation and Localization:** For use in real-time translation applications for translating conversations in different languages or translating audio media.
- **Virtual Assistant or Chatbot:** For use to create a fictional brand or character voice for smart assistants in or for virtual web assistants, appliances, cars, home

appliances, toys, control of IoT devices, navigation systems, reading out personal messages, virtual companions, or customer service scenarios.

Considerations when using Custom Neural Voice

The ability to produce synthetic media generatively, rapidly, and at scale offers unique opportunities for augmenting personal and creative expression, but also poses unprecedented challenges to public safety by making it easier to misappropriate, misinform, mislead, propagandize, or libel; while simultaneously undermining the believability of legitimate recordings and other digital artifacts. For this reason, Microsoft has established the following [Code of Conduct](#) that prohibits certain uses of Custom Neural Voice.

In addition to reviewing the Code of Conduct when choosing a use case to use Custom Neural Voice, take the following considerations into account:

- **Avoid photo realistic avatars with synthetic voices to represent real people** - One of the key principles of the responsible use of Custom Neural Voice is to ensure that our consumers understand and expect the content they are interacting with is synthetic. Pairing a photo realistic avatar with the custom neural voice of a real person could potentially create an illusion to consumers that they are interacting with a real and known person. This would erode trust in the application and potentially cause harm to consumers.
- **Carefully consider using a synthetic voice with contents without editorial control**
 - Synthetic voice can sound like a human, which could amplify the effect of fake or misleading content.

See also

- [Introduction to Custom Neural Voice](#)
- [Characteristics and limitations for using Custom Neural Voice](#)
- [Application process for Custom Neural Voice](#)
- [Data, privacy, and security for Custom Neural Voice](#)

Characteristics and limitations for using Custom Neural Voice

Article • 10/07/2022 • 9 minutes to read

Custom Neural Voice enables you to create brand voices for your apps. To create a Neural TTS voice that sounds like a specific person or style for your apps, the recordings of the voice talent embodying that personality and style are used. The audio data is then uploaded to the [Custom Voice platform](#) where a machine learning model is built using transfer learning technology and our multi-lingual, multi-speaker base model. While the custom voice models can speak in a voice that sounds like the voice talent, the similarity and naturalness of the voice depends on a number of factors, including the size of the training data, the quality of the recording, the accuracy of the transcript file, how well the recorded voice in the training data matches the personality of the designed voice for your intended use case, as well as other factors.

Fairness

At Microsoft, we strive to empower every person on the planet to do more. An essential part of this goal is working to create technologies and products that are fair and inclusive. Fairness is a multi-dimensional, socio-technical topic and impacts many different aspects of our product development. You can learn more about Microsoft's approach to fairness [here](#).

One dimension we need to consider is how well the system performs for different groups of people. Research has shown that without conscious effort focused on improving performance for all groups, it is often possible for the performance of an AI system to vary across groups based on factors such as race, ethnicity, gender, and age.

Custom Neural Voice models are trained using transfer learning technology based on our multi-lingual, multi-speaker base model, with the recording samples of human voices that you provide. The richer the base model, the more powerful the transfer learning, the less training data is required. While our base model is built using a wide range of speech data that includes different speaking accents, age groups, genders, across more than 50 languages (see list of supported languages [here](#)), it's possible that certain demographic groups are not well represented in some languages. For example, we cover less speech data from children than from adults. To accommodate this limitation, we require at least 300 lines of recordings (or, around 30 minutes of speech) to be prepared as training data for Custom Neural Voice, and we recommend 2,000 lines of recordings (2-3 hours of speech) to create a voice for production use. Each line of

recording (a.k.a. “utterance”) consists of a normal sentence or a short phrase that is read by your chosen voice talent. With 2,000 utterances, our system can learn the target voice characteristics well even if the base model doesn’t include a similar speaker. The evaluation is done using SMOS measurement as described below. We recommend lower score for SMOS measurement across languages as judges need to compare the recording from primary language with voice in secondary language.

Each application is different, and our base model may not match your context or cover all scenarios required for your use case. We encourage developers to thoroughly evaluate synthesized voice quality for the service with real-world data that reflects your use case, including testing with users from different demographic groups and with different speech characteristics. Please see the [Quality of the voice model trained](#) section for best practices for building high quality voice models.

In addition to ensuring similar performance, it is important to consider how to minimize risks of stereotyping and erasure that may result from synthetic voices. For example, if you are creating a custom neural voice for a smart voice assistant, carefully consider what voice is appropriate to create, and seek diverse perspectives from a variety of backgrounds. When building and evaluating your system, seek diverse input.

Quality of the voice model trained

You can measure the quality of the voice model produced by Custom Neural Voice through listening to the samples generated by the service. A quantitative approach is to use [MOS](#) (Mean Opinion Score) to measure naturalness. In MOS, independent human judges score the naturalness of the voice samples they are presented with using a scale of 1 to 5, with 1 being the worst quality and 5 being the best quality. An average score is then calculated for the report. When selecting judges, it is recommended to include people with a variety of backgrounds, including judges from different demographic groups.

In custom voice quality evaluations, MOS gap is usually used to compare the quality of the TTS voice model against a human recording. The quality of a voice model created by Custom Neural Voice compared to that of a human recording is expected to be close, with a gap of no more than 0.5 in the MOS score.

In addition, you can use Similarity MOS (SMOS) to measure how similar the custom voice sounds compared to the original human recordings. With SMOS studies, judges are asked to listen to a set of paired audios, one generated using the custom voice, the other from the original human recordings in the training data, and rate if the two audios in each pair are spoken by the same person, using a five-point scale (1 being the lowest,

5 the highest). The average score is reported as the SMOS score. We recommend that a good custom neural voice should achieve an SMOS higher than 4.0.

Besides measuring naturalness with MOS and SMOS, you can also assess the intelligibility of the voice model by checking the pronunciation accuracy of the generated speech. This is done by having the judges listen to a set of testing samples, determine whether they can understand the meaning and indicate any words that were unintelligible to them. Intelligibility rate is calculated using the percentage of the correctly intelligible words among the total number of words tested (i.e., the number of intelligible words/the total number of words tested * 100%). Normally a usable TTS engine needs to reach a score of > 98% for intelligibility.

Measurement	Definition	How it is calculated	Recommended text size	Recommended score
MOS	Mean Opinion Score of the quality of the audios	Average of the rating scores of each judge on each audio	> 30 generated audios > 20 judges on each audio	> 4.0 (normally requires the MOS of the human recording is higher than 4.5)
MOS gap	The MOS score difference between human recordings and the generated audios	The MOS score on the human recordings minus the MOS score on the generated audios	> 10 human recordings > 30 generated audios > 20 judges on each audio	<0.5
SMOS	The similarity of the generated audios to the human recordings	Average of the rating scores of the similarity level on each pair of audios	> 40 pairs > 20 judges on each pair	>4.0 >3.5 (secondary language)
Intelligibility	The pronunciation accuracy of the generated speech at the word level	Percentage of the correctly intelligible words among the total number of words tested	> 60 generated audios > 10 judges on each audio	>98%

Best practices to improve the quality of the voice model

Creating a great custom voice requires careful quality control in each step, from voice design, data preparation, to the deployment of the voice model to your system.

Persona design

Before building a custom neural voice, particularly if it is for a fictitious voice, it is a good practice to design a persona of the voice that would represent your brand, fit well with the user scenarios as well as resonate with your intended audience. This can be specified in a persona brief: a document that describes the features of the voice and the fictitious or real person behind the voice. This persona brief document will help guide the process of creating a custom voice model including the recording scripts, the voice talent selection, and the training and tuning of the voice.

Script selection

Your recording script defines the training dataset for your voice model and is the starting point of any custom voice recording session. Your recording script must be carefully selected to represent the user scenarios for your voice. For example, if you are going to use the voice model for your customer service bot, you may want to use the phrases from your bot conversations to create the recording script. To create a voice for reading stories, you can use a relevant story script for your recordings.

Follow the [guidance here](#) to prepare your script in more detail.

Note

When preparing your recording script, make sure you include a statement sentence to acquire the voice talent acknowledgement for using their voice data to create a TTS voice model and generate synthetic speech. You can find the statement in multiple languages [here ↗](#). The language of the verbal statement must be the same as your recording.

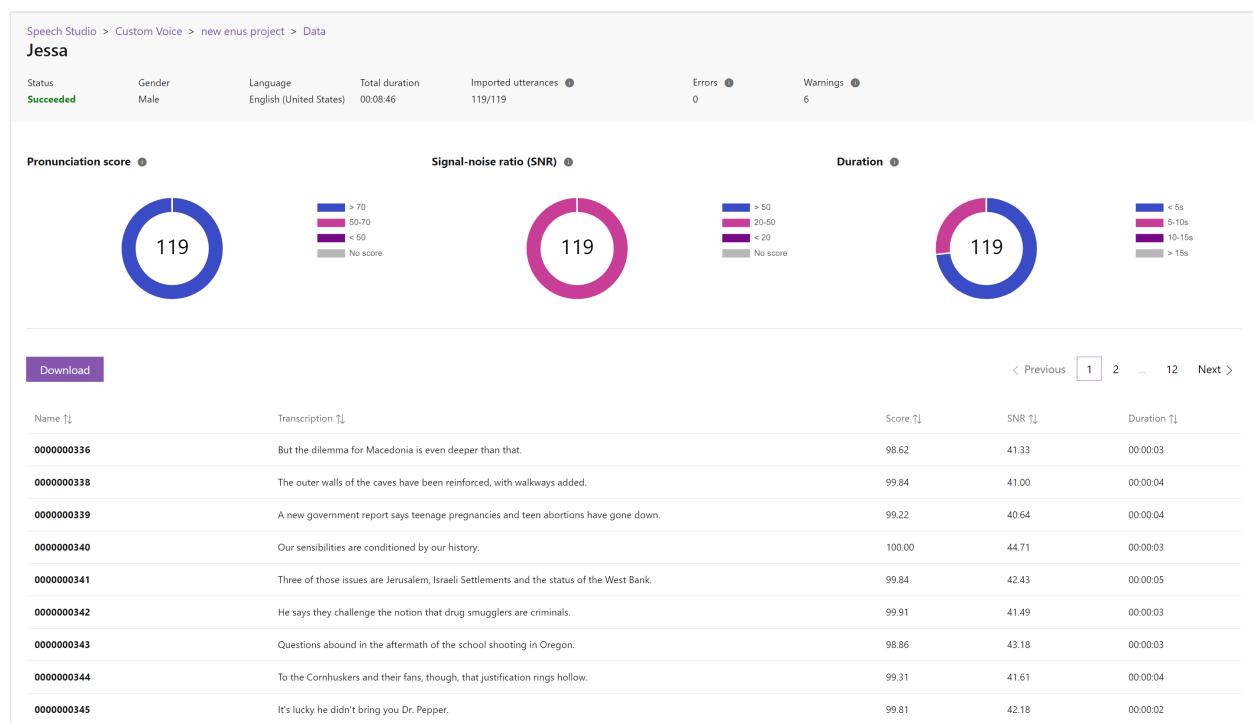
As a technical safeguard intended to prevent misuse of Custom Neural Voice, Microsoft will use this recording to verify that the voice talent's voice in the script matches the voice provided in the training data through **Speaker Verification**. Read more about this process in the **Data and Privacy section** [here ↗](#).

Preparing training data

We recommend that the audio recordings should be captured in a professional quality recording studio so that high signal-to-noise ratio is achieved without distortion, and other defects such as RF interference and plosives are minimized. Follow the [recording guidance here](#).

The quality of the voice model trained heavily depends on the recorded voice used for training. Consistent volume, speaking rate, speaking pitch, and consistency in expressive mannerisms of speech are essential to create a great custom neural voice. Custom Neural Voice creates a synthetic voice model that mimics the voice in the training data. The quality of the recording of the voice talent is the upper bound of the quality of the Custom Neural Voice model. The voice model will capture the styles, accents, and other characteristics of the voice, including defects such as noises and mispronunciations.

Unexpected errors such as mismatching of the transcript to the recordings can introduce mistakes in the pronunciation labeling to the training system. The [Speech Studio](#) provides capabilities for users to evaluate the pronunciation accuracy, identify noises, check the audio length for each utterance in the training dataset, and filter out unqualified recordings. For example, in the dataset detail view, you can check the pronunciation score, the signal-to-noise ratio and the audio length for your training data.



It is nevertheless not possible to provide 100% accurate data review results automatically. As the developer creating the synthetic voice, you are responsible for reviewing and ensuring the audio quality of the training data is sufficient for voice model building.

Tuning and adjustment

The style and the characteristics of the trained voice model depend on the style and the quality of the recordings from the voice talent used for training. However, several adjustments can be made using [SSML \(Speech Synthesis Markup Language\)](#) when you make the API calls to your voice model to create synthetic speech. SSML is the markup language used to communicate with the Speech service to convert text into audio. The adjustments include change of pitch, rate, intonation, and pronunciation correction. If the voice model is built with multiple styles, SSML can also be used to switch the styles.

All of the SSML markups mentioned above can be passed directly to the API. We also provide an online tool, [Audio Content Creation](#), that allows customers to tune using a friendly UI.

See also

- [Introduction to Custom Neural Voice](#)
- [Transparency note and use cases for Custom Neural Voice](#)
- [Application process for Custom Neural Voice](#)
- [Data, privacy, and security for Custom Neural Voice](#)

Limited Access to Custom Neural Voice

Article • 10/07/2022 • 3 minutes to read

As part of Microsoft's commitment to responsible AI, Custom Neural Voice is designed with the intention of protecting the rights of individuals and society, fostering transparent human-computer interaction, and counteracting the proliferation of harmful deepfakes and misleading content. For this reason, Custom Neural Voice is a Limited Access feature available by registration only, and only for certain use cases.

Note

Approved registrants for Custom Neural Voice will also get access to deploy and use a Custom Neural Voice Lite voice model. Custom Neural Voice Lite is a project type in public preview that allows you to record 20-50 voice samples on Speech Studio and create a lightweight custom voice model for demonstration and evaluation purposes. Both the recording script and the testing script are pre-defined by Microsoft. The synthetic voice model created using the Custom Neural Voice Lite project could be deployed and used at your discretion, after you register and full access to Custom Neural Voice is granted.

Registration process

As a Limited Access feature, Custom Neural Voice requires registration. Only customers managed by Microsoft, meaning those who are working directly with Microsoft account teams, are eligible for access. Customers who wish to use this feature are required to register by [submitting a registration form](#). The use of Custom Neural Voice is limited to the use case selected at the time of registration. Microsoft may require customers to re-verify this information.

The Custom Neural Voice capability is made available to customers under the terms governing their subscription to Microsoft Azure Services (including the [Service Specific Terms](#)). Please review these terms carefully as they contain important conditions and obligations governing your use of Custom Neural Voice. For example, these terms include, but are not limited to, the following obligations:

- 1. Voice talent and approved use cases.** Customer will need to warrant that it has obtained explicit written permission from voice talent prior to creating a voice model, which includes sharing the [Disclosure for Voice Talent](#) with the voice talent

and may only use the custom voice model for use cases selected during registration.

2. **Implementation requirements.** As outlined in our Code of Conduct [here](#), customer must not use its custom voice model for certain prohibited uses and must also agree that when deploying the custom voice model, its implementation will [disclose the synthetic nature](#) of the service to users and support a feedback channel that allows users of the service to report issues and share details with Microsoft.
3. **Microsoft's additional processing and use of voice talent data.** Before Customer can train a custom voice model, we may require customers to upload an audio file to the Speech Studio with a pre-defined statement from the voice talent acknowledging customer's use of Voice Talent's voice to create a synthetic voice. As a technical safeguard intended to help prevent misuse of this service, Microsoft reserves the right to use Microsoft's speaker recognition biometric identification technology on this recorded statement and verify it against the training audio data in order to provide some assurance that the voices come from the same speaker. Microsoft will also continue to retain this audio file and the custom voice model to protect the security and integrity of our services. ***Customer is responsible for ensuring all necessary permissions are received from its voice talent for these purposes.***

Learn more about how we process this data in the [Data and Privacy section](#). Learn more about the legal terms that apply to this feature [here](#).

Help and support

FAQ about Limited Access features can be found [here](#).

If you need help with Custom Neural Voice, find support [here](#).

Report abuse of Custom Neural Voice [here](#).

Next steps

- [Introduction to Custom Neural Voice](#)
- [Transparency note and use cases for Custom Neural Voice](#)
- [Responsible deployment of synthetic speech](#)
- [Disclosure for voice talent](#)

Guidelines for responsible deployment of synthetic voice technology

Article • 06/20/2022 • 3 minutes to read

In this article, you learn about Microsoft's general design guidelines for using synthetic voice technology. These guidelines were developed in studies that Microsoft conducted with voice talent, consumers, and individuals with speech disorders to guide the responsible development of synthetic voices.

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](#).

Microsoft requires its customers to disclose the synthetic nature of custom neural voice to its users.

- Make sure to provide adequate disclosure to audiences, especially when using voice of a well-known person - People make their judgment on information based on the person who delivers it, whether they do it consciously or unconsciously. For example, disclosure could be verbally shared at the start of a broadcast. For more information visit the [disclosure patterns](#).
- Consider proper disclosure to parents or other parties with use cases that are designed for minors and children. If your use case is intended for minors or children, you'll need to ensure that the parents or legal guardians can understand the disclosure about the use of synthetic media and make the right decision for the minors or children on whether to use the experience.

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 read by a synthetic voice, some users prefer a more empathetic and human-like tone, while others prefer an 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. 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.

Voice talents should have control over their voice model (how and where it will be used) and be compensated for its use. Microsoft requires custom voice customers to obtain explicit written permission from their voice talent to create a synthetic voice and its agreement with voice talents contemplate the duration, use and any content limitations. If you're creating a synthetic voice of a well-known person, you should provide a way for the person behind the voice to edit or approve the contents.

Some voice talents are unaware of the potential malicious uses of the technology and should be educated by system owners about the capabilities of the technology.

Microsoft requires Customers to share Microsoft's [Disclosure for Voice Talent](#) with Voice Talent directly or through Voice Talent's authorized representative that describes how synthetic voices are developed and operate in conjunction with text to speech services.

Considerations for people 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 transfer of voice font ownership after death to family members, if permission has been given.

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 may wish to update 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.

See also

- [Disclosure for Voice Talent](#)
- [How to Disclose](#)
- [Disclosure Design Patterns](#)

Disclosure for voice talent

Article • 10/07/2022 • 10 minutes to read

The goal of this article is to help voice talent understand the technology behind the text-to-speech (TTS) capabilities that their voices help create. It also contains important privacy disclosures for voice talent about how Microsoft may process, use and retain audio files containing voice talent's recorded statements and Custom Neural Voice voice models to help Microsoft prevent, and/or respond to complaints of, misuse of Cognitive Services or Custom Neural Voice services.

Microsoft is committed to designing AI responsibly. We hope this note will foster a greater shared understanding among tech builders, voice talent, and the general public about the intended and beneficial uses of this technology.

Key TTS terms

Voice model: A text-to-speech computer model that can mimic unique vocal characteristics of a target speaker. A voice model is also called as voice font or synthetic voice. A voice model is a set of parameters in binary format that is not human readable and does not contain audio recordings. It cannot be reverse engineered to derive or construct the audio recordings of a human being speaking.

Voice talent: Individuals or target speakers whose voices are recorded and used to create voice models that are intended to sound like the voice talent's voice.

Two broad categories of text-to-speech (TTS)

Following are the two broad categories for text-to-speech (TTS).

Standard TTS

How it works: The standard, or "traditional," method of TTS breaks down spoken language into phonetic snippets that can be remixed and matched using classical programming or statistical methods.

What to know about it: Standard TTS requires a large volume of voice data—in the range of 10,000 lines or more—to produce a more human-like voice model. With fewer recorded lines, a standard TTS voice model will tend to sound more obviously robotic.

Examples of how Microsoft uses it:

- **Platform Voice** is a feature of the Speech Service on Azure that offers "off-the-shelf" voice models for public use. Platform Voices are also used in several Microsoft products including the Edge Browser, Narrator, Office, and Teams.
- **Custom Voice** is a feature of the Speech Service on Azure that allows you to build a synthetic voice model using recordings from a voice talent to represent a specific persona for a corporation/enterprise.
- **Microsoft and/or Windows system voices** are included in the Windows operating system. They are also used in several applications such as Narrator, Cortana, Edge Read Aloud, and Teams.

What to expect when recording: Contributing at least 6,000 lines to produce a good quality voice font.

Neural TTS

How it works: Neural TTS synthesizes speech using deep neural networks that have "learned" the way phonetics are combined in natural human speech rather than using classical programming or statistical methods. In addition to the recordings of a target voice talent, neural TTS uses a source library that contains voice recordings from many different speakers.

What to know about it: Because of the way it synthesizes voices, neural TTS can produce styles of speech that weren't part of the original recordings, such as changes in tone of voice and affectation. Neural TTS voices sound fluid and are good at replicating the natural pauses, idiosyncrasies, and hesitancy that people express when they're speaking. Those who hear synthetic voices made via neural TTS tend to rate them closer to human speech than standard TTS voices.

Examples of how Microsoft uses it:

- **Platform Voice** is a feature of the Speech Service on Azure that offers "off-the-shelf" voice models for public use. Platform Voices are also used in several Microsoft products including the Edge Browser, Narrator, Office, and Teams.
- **Custom Neural Voice** is a feature of the Speech Service on Azure that allows you to create a one-of-a-kind custom synthetic voice model for your brand. The following capabilities are used to produce Custom Neural Voices:
 - **Language transfer** can express in a language different from the original voice recordings.
 - **Style transfer** can express in a style of speaking different from the original voice recordings. For example, a newscaster voice.
 - **Voice transformation** can express in a manner different from the original voice recordings. For example, modifying tone or pitch to create different character

voices.

- Other voices used in Microsoft's products and services, such as Cortana.

What to expect when recording: Contributing at least 300 lines for a proof of concept voice model and about 2,000 lines to produce a new voice model for production use.

Voice talent and synthetic voices: an evolving relationship

Recognizing the integral relationship between voice talent and synthetic voices, Microsoft interviewed voice talent to better understand their perspectives on new developments in the technology. Research we conducted in 2019 showed that voice talent saw potential benefit from the capabilities introduced by neural TTS, such as saving studio time to complete recording jobs, and adding capacity to complete more voice acting assignments. At the same time, there were varying degrees of awareness about how developments in TTS technology could potentially impact their profession.

Overall, voice talent expressed a desire for transparency and clarity about:

- Limits on what their voice likeness could and could not be used to express.
- The duration of allowable use of their voice likeness.
- Potential impact on future recording opportunities.
- The persona that would be associated with their voice likeness.

Synthetic voice in wider use

Traditionally, TTS systems were somewhat limited in adoption due to their robotic sound. Most were used to support accessibility—for example as a screen reader for people who are Blind or have low vision. TTS has also been used by people with a speech impairment. For instance, the late Stephen Hawking used a TTS-generated voice.

Now, with increasingly realistic-sounding synthetic voices and the uptick in more familiar, everyday interactions between machines and humans, the uses of this technology have proliferated and expanded. TTS systems power voice assistants across an array of devices and applications. They read out news, search results, public service announcements, educational content, and much more.

Microsoft's approach to responsible use of TTS

Every day, people find new ways to apply TTS technology, and not all are for the good of individuals or society. If misused, believably human-sounding TTS voices, especially a

custom voice that mimics a real person, could cause harm. For example, a misinformation campaign could become much more potent if it used the voice of a well-known public figure.

We recognize that there's no perfect way to prevent media from being modified or to unequivocally prove where it came from. Therefore, our approach to responsible use has focused on being transparent about neural TTS, evaluating appropriate use, and demonstrating our values through action.

Requirements and tips for meaningful consent from voice talent

To use Custom Neural Voice, we contractually require you to do the following:

- Obtain explicit written permission from voice talent to use that person's voice for the purpose of creating a custom voice.
- Provide this document to voice talent so they can understand how TTS works, and how it may be used once they complete the audio recording process.
- Get necessary permissions from voice talent for Microsoft's processing, use and retention of voice talent's audio files to perform speaker verification against training data and our use and retention of voice models as described below.

We also recommend that you do the following:

- Share the intended contexts of use with voice talent so they are aware of who will hear their voice, in what scenarios, and whether/how people will be able to interact with it.
- Ensure voice talent are aware that a voice model made from their recordings can say things they didn't specifically record in the studio.
- Discuss whether there's anything they'd be uncomfortable with the voice model being used to say.

Microsoft's processing, use and retention of voice talent data

Microsoft's use of Voice talent audio files for Speaker Verification

You must obtain permission from voice talents for use of their voice to create custom voice models for a synthetic voice. This technical safeguard is intended to help prevent

misuse of our service, by, for example, preventing someone from training voice models with audio recordings and using it to spoof a voice without the speaker's knowledge or consent.

In [Speech Studio](#), you must upload an audio file with a recorded consent statement from the voice talent. Microsoft reserves the right to use Microsoft's speaker recognition technology on this recorded statement and verify it against the training audio data in order to provide some assurance that the voices came from the same speaker or as otherwise necessary to investigate misuse of the services.

The speaker's voice signatures created from the recorded statement files and training audio data are used by Microsoft solely for the purposes stated above. Microsoft will retain the recorded statement file for as long as necessary in order to preserve the security and integrity of Microsoft's Azure Cognitive Services. Learn more about how we process, use and retain this data in the [Data and Privacy section](#).

Microsoft's use of Custom Neural Voice models

While you maintain the exclusive usage rights to your Custom Neural Voice model, Microsoft may independently retain a copy of Custom Neural Voice models for as long as necessary. Microsoft may use your Custom Neural Voice model for the sole purpose of protecting the security and integrity of Microsoft Azure Cognitive Services.

Microsoft will secure and store a copy of Voice Talent's recorded statement and Custom Neural Voice models with the same high level security that it uses for its other Azure Services. Learn more at [Microsoft Trust Center](#).

We will continue to identify and be explicit about the intentional, beneficial, and intended uses of TTS that are based upon existing social norms and expectations people have around media when they believe it to be real or fake. In line with Microsoft's trust principles, Microsoft does not actively monitor or moderate the audio content generated by your use of Custom Neural Voice. You are solely responsible for ensuring that usage complies with all applicable laws and regulations and in accordance with the terms of its agreement with voice talent.

Microsoft's use of Voice Talent data with Custom Neural Voice Lite

Custom Neural Voice Lite is a project type in public preview that allows you to record 20-50 voice samples on Speech Studio and create a lightweight custom voice model for demonstration and evaluation purposes. Both the recording script and the testing script

are pre-defined by Microsoft. The synthetic voice model created using the Custom Neural Voice Lite project could be deployed and used at your discretion, after you apply and full access to Custom Neural Voice is granted.

The synthetic voice and the related audio recording submitted via the Speech Studio will automatically be deleted within 90 days from the Speech Studio portal unless you decide to deploy the synthetic voice, in which case, you will control the duration of its retention. If the Voice Talent would like to have the synthetic voice and the related audio recordings deleted before 90 days, they can delete them on the portal directly, or contact their enterprise to do so.

Before you can deploy the Synthetic Voice model created using a Custom Neural Voice Lite project, it's required that the Voice Talent provide an additional audio recording of the Voice Talent acknowledging the synthetic voice will be used by their enterprise outside of the demonstration and evaluation purpose.

Guidelines for responsible deployment

Because TTS is an adaptable technology, there are grey areas in determining how it should or shouldn't be used. To navigate these, we've formulated the following guidelines for using synthetic voice models:

- Protect owners of voices from misuse or identity theft.
- Prevent the proliferation of fake and misleading content.
- Encourage use in scenarios where consumers expect to be interacting with synthetic content.
- Encourage use in scenarios where consumers observe the generation of the synthetic content.

Examples of inappropriate use

TTS must not be used to:

- Deceive people and/or intentionally misinform;
- Claim to be from any person, company, government body, or entity without explicit permission to make that representation and/or impersonate to gain unauthorized information or privileges;
- Create, incite, or disguise hate speech, discrimination, defamation, terrorism, or acts of violence;
- Exploit or manipulate children;
- Make unsolicited phone calls, bulk communications, posts, or messages;
- Disguise policy positions or political ideologies;

- Disseminate unattributed content or misrepresent the source

Examples of appropriate use

Appropriate TTS use cases could include, but are not limited to:

- Virtual agents based on fictional personas. For example, on-demand web searching, IoT control, or customer support provided by a company's branded character.
- Entertainment media for use in fictional content. For example, movies, video games, tv, recorded music, or audio books.
- Accredited educational institutions or educational media. For example, interactive lesson plans or guided museum tours.
- Assistive technology and real-time translation. For example, ALS-afflicted individuals preserving their voices.
- Public service announcements using fictional personas. For example, airport or train terminal announcements.

See also

- [Transparency note and use cases for Custom Neural Voice](#)
- [Responsible deployment of synthetic speech](#)
- [Disclosure design guidelines](#)
- [Disclosure design patterns](#)
- [Data, privacy, and security for Custom Neural Voice](#)

Disclosure design guidelines

Article • 06/20/2022 • 3 minutes to read

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.

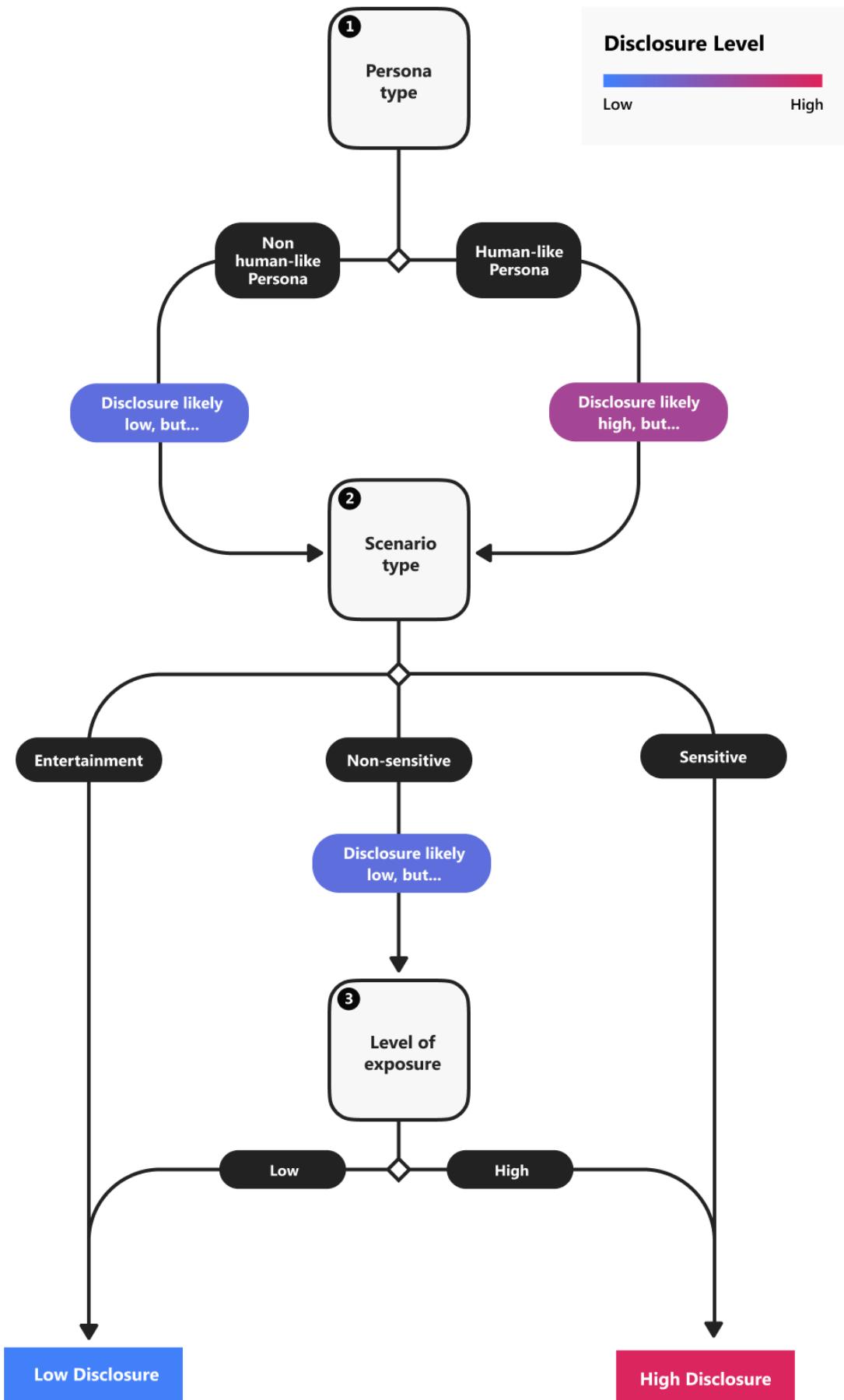
Context of use	Potential Risks & Challenges

	Context of use	Potential Risks & Challenges
1. Persona type	If any of the following apply, your persona fits under the 'Human-like Persona' category: <ul style="list-style-type: none"> • 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) • The synthetic voice is based on the voice of a widely recognizable real person (e.g., celebrity, political figure) 	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.
2. Scenario type	If any of the following apply, your voice experience fits under the 'Sensitive' category: <ul style="list-style-type: none"> • Obtains or displays personal information from the user • Broadcasts time sensitive news/information (e.g., emergency alert) • Aims to help real people communicate with each other (e.g., reads personal emails/texts) • Provides medical/health assistance 	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.

Context of use	Potential Risks & Challenges
<p>3. Level of exposure Your voice experience most likely fits under the 'High' category if:</p> <ul style="list-style-type: none"> • The user will hear or interact with the synthetic voice frequently or for a long duration of time 	<p>The importance of being transparent and building trust with users is even higher when establishing long-term relationships.</p>

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.



See also

- Disclosure design patterns
- Disclosure for Voice Talent
- Guidelines for Responsible Deployment of Synthetic Voice Technology

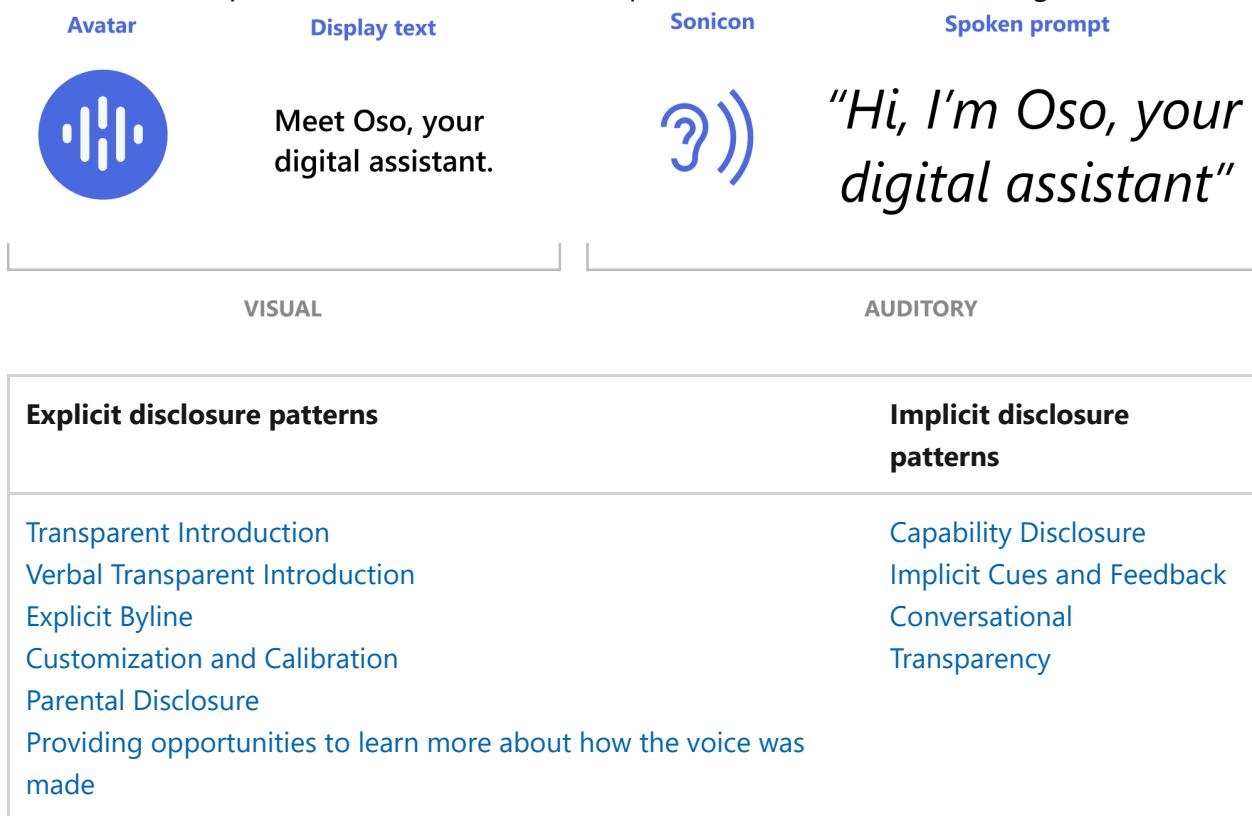
Disclosure design patterns

Article • 09/01/2022 • 8 minutes to read

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.



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

If your synthetic voice experience...	Recommendations	Design patterns
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
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

If your synthetic voice experience...	Recommendations	Design patterns
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

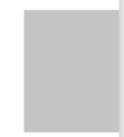
- Describe that the voice is artificial (e.g. "digital")
- Describe what the agent is capable of doing
- Explicitly state the voice's origins
- Offer an entry point to learn more about the synthetic voice

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.



Meet Oso, your new digital voice assistant



Search movies, tv shows, and more.



Meet your new voice assistant, Oso*. You can control your TV with Oso and get things done just by asking. Oso can help you search movies, shows, look up the weather, search the internet, and more.

*Oso uses synthetic voice technology.



Try saying...

Continue

Learn more

Set up later

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

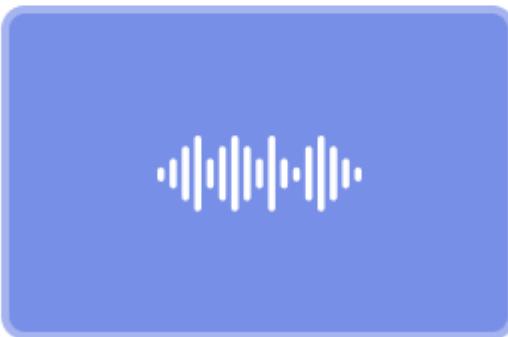
- Allow users to customize the voice (e.g., select language and voice type)
- Provide users a way to teach the system to respond to their unique voice (e.g., voice calibration, custom commands)
- Optimize for user-generated or contextual interactions (e.g., reminders)

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:

$$7 \times 5 = \boxed{}$$

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.

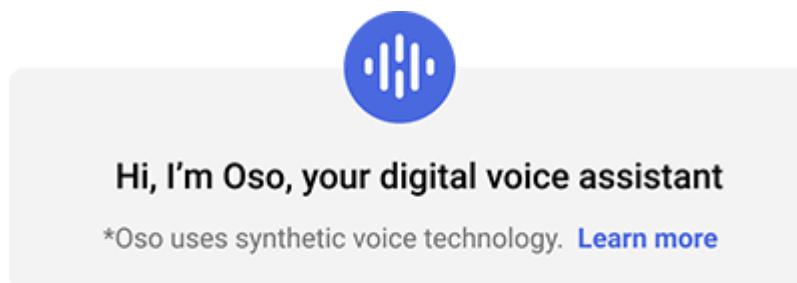
Recommendations

- 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.



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](#)
[FAQ](#)
[Legal](#)
[Contact us](#)

Features ▶

[Synthetic voice technology ▾](#)

X
X
X

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	Sonicon (e.g., a brief distinctive sound, series of musical notes)	Vibration
Responsive real-time cues (e.g., animations)		
Non-screen cues (e.g., lights and patterns on a device)		

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.



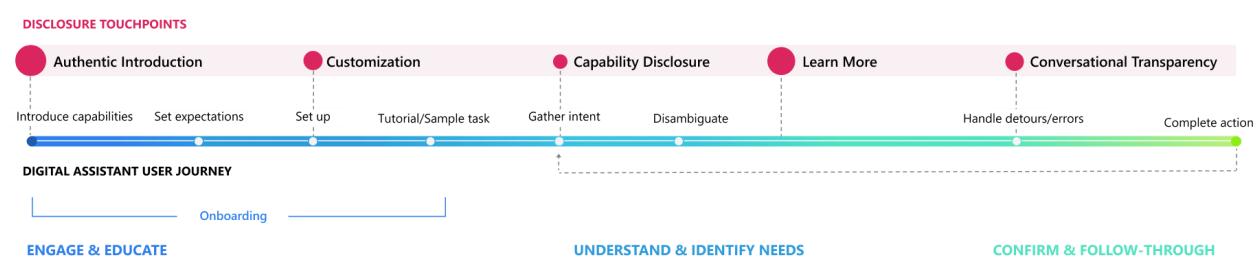
Do you have a soul?



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](#)

See also

Code of Conduct for Text-to-Speech integrations

Article • 07/18/2022 • 2 minutes to read

The following code of conduct defines the requirements that all Text-to-Speech ("TTS") implementations must adhere to in good faith. This code of conduct is in addition to the Acceptable Use Policy in the [Microsoft Online Services Terms](#).

Content requirements

- All content released through your TTS integration and associated metadata must be originally created by the publisher, appropriately licensed from the third-party rights holder, used as permitted by the rights holder, or used as otherwise permitted by law.
- This service must not be used to simulate the voice of politicians or government officials, even with their consent.

Your TTS feature must:

- disclose the synthetic nature of the voice to users such that they are not likely to be deceived or duped—or able to prank others—into believing they are interacting with a real person;
- support a feedback channel that allows users of the service to report issues with the service

Your TTS feature must not be used to:

- deceive or intentionally misinform people;
- claim to be from any person, company, government body, or entity without explicit permission to make that representation and/or impersonate to gain unauthorized information or privileges;
- create, incite, or disguise hate speech, discrimination, defamation, terrorism, or acts of violence;
- exploit or manipulate children;
- make unsolicited phone calls, bulk communications, posts, or messages;
- disguise policy positions or political ideologies;
- disseminate unattributed content or misrepresent sources

Report abuse

If you suspect that Custom Neural Voice is being used in manner that is abusive or illegal, or infringes on your rights or the rights of other people, you can report it at the [Report Abuse Portal](#).

The [Azure Online Service Terms \(OST\)](#) prohibits customers from using any Azure service, including Custom Neural Voice, to violate the law. Customers who violate terms of service will lose access to those services.

Examples that would violate the policy include using Custom Voice to intentionally deceive or misinform; claim to be from any person, company, government body, or entity without explicit permission to make that representation and/or impersonate to gain unauthorized information or privileges; create, incite, or disguise hate speech, discrimination, defamation, terrorism, or acts of violence; exploit or manipulate children; make unsolicited phone calls, bulk communications, posts, or messages; disguise policy positions or political ideologies; disseminate unattributed content or misrepresent the source.

See also

- [Transparency note and use cases for Custom Neural Voice](#)
- [Responsible deployment of synthetic speech](#)
- [Disclosure design guidelines](#)
- [Disclosure design patterns](#)
- [Data, privacy, and security for Custom Neural Voice](#)

Data, privacy, and security for Custom Neural Voice

Article • 10/07/2022 • 8 minutes to read

This article provides details regarding how Custom Neural Voice data provided by you is processed, used and stored. As an important reminder, you are responsible for your use and the implementation of this technology and are required to obtain all necessary permissions from voice talents for the processing of their voice data to develop a synthetic voice as well as any licenses, permissions or other proprietary rights required for the content you input into the text-to-speech ("TTS") service, part of Speech in Azure Cognitive Services, to generate audio content in the synthetic voice. Some jurisdictions may impose special legal requirements for the collection, processing and storage of certain categories of data, such as biometric data and mandate disclosing the use of synthetic voices to users. Before using Custom Neural Voice and the TTS service for the processing and storage of data and creation of synthetic speech, you must ensure compliance with any such legal requirements that may apply to you.

What data does Custom Neural Voice and TTS process?

Custom Neural Voice processes the following types of data:

- **Recorded statement file of voice talent.** When using the [Speech Studio](#), customers are required to upload a recorded statement of the voice talent that acknowledges that their voice will be used by customer to create synthetic voice(s).

ⓘ Note

When preparing your recording script, make sure you include the statement sentence to acquire the voice talent acknowledgement. You can find the statement in multiple languages [here](#). The language of the verbal statement must be the same as your recording.

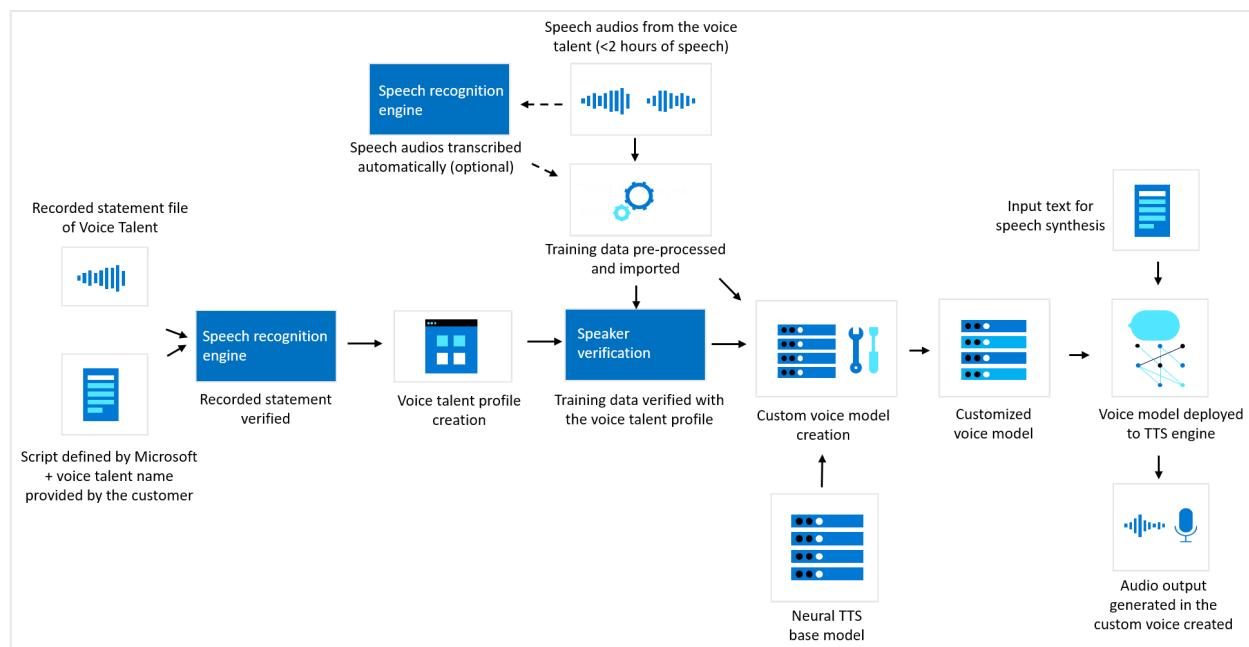
- **Training data (including audio files and related text transcripts).** This includes audio recordings from the voice talent who has agreed to use their voice for the model training and the related text transcripts. You can provide their own text transcriptions of audio or use the automated speech recognition transcription feature available within the Speech Studio to generate a text transcription of the

audio. Both the audio recordings and the text transcription files will be used as the voice model Training Data.

- **Text as the test script.** You can upload your own text-based scripts to evaluate and test the quality of the custom voice model by generating speech synthesis audio samples.
- **Text input for speech synthesis.** This is the text you select and send to TTS to generate audio content using your custom neural voice.

How does Custom Neural Voice and TTS process data?

The diagram below illustrates how your data is processed. This diagram covers three different types of processing: how Microsoft verifies voice files of the voice talent prior to the custom neural voice model training, how Microsoft creates a custom neural voice model with your training data, and how TTS processes your text input to generate audio content.



Voice file verification

Microsoft requires customers to upload an audio file with a recorded statement from its voice talent acknowledging Customer's use of their voice to the Speech Studio.

Microsoft may use Microsoft's [speech-to-text/speech recognition](#) technology on this recorded statement to transcribe it to text and verify the content in the recording matches the pre-defined script provided by Microsoft. This audio statement, along with the description information you provide with the audio is used to create a voice talent

profile. You must associate training data with the relevant voice talent profile when initiating custom neural voice training.

Microsoft may process biometric voice signatures from the recorded voice statement file of the voice talent and from randomized audios from the training datasets in order to ascertain that the voice signature in each of the audio recordings matches the same speaker with reasonable confidence using the [Speaker Verification](#) feature of Speech, in Azure Cognitive Services. A voice signature may also be called a “voice template” or “voiceprint” and it is a numeric vector that represents an individual’s voice characteristics that is extracted from audio recordings of a person speaking. This technical safeguard is intended to help prevent misuse of Custom Neural Voice, by, for example, preventing customers from training voice models with audio recordings and using it to spoof a voice without a speaker’s knowledge or consent.

The voice signatures are used by Microsoft solely for the purposes of speaker verification or as otherwise necessary to investigate misuse of the services

The [Online Services Data Protection Addendum](#) (“DPA”) sets forth customers and Microsoft’s obligations with respect to the processing and security of Customer Data and Personal Data in connection with Azure and is incorporated by reference into customers enterprise agreement for Azure services. Microsoft’s data processing in this section is governed under the Legitimate Interest Business operations section of the Data Protection Addendum.

Training a custom neural voice model

The training data (speech audios) submitted to the Speech Studio is pre-processed using automated tools for quality checking including data format check, pronunciation scoring, noise detection, script mapping, etc.. The training data is then imported to the model training component of the custom voice platform. During the training process, the training data (both voice audio and text transcriptions) are decomposed into fine-grained mappings of voice acoustics and text, such as a sequence of phonemes. Through further complex machine learning modeling, this is built into a voice model, which then can be used to generate voice that sounds like the voice talent, and can be in different languages from the recording. The voice model is a text-to-speech computer model that can mimic unique vocal characteristics of a target speaker. It represents a set of parameters in binary format that is not human readable and does not contain audio recordings.

Customer’s training data is only used to develop customer’s custom voice model and is not used by Microsoft to train or improve any Microsoft TTS voice models.

Speech synthesis/audio content generation

Once the voice model is created, you can use it to create audio content through the TTS service with two different options.

For real time speech synthesis, you send the input text to the TTS service via the [TTS SDK](#) or [RESTful API](#). TTS processes the input text and returns output audio content files in real time to the your application that made the request.

For asynchronous synthesis of long audio (batch synthesis), you submit the input text files to the TTS batch service via the [Long Audio API](#) to asynchronously create audios longer than 10 minutes (for example audio books or lectures). Unlike synthesis performed using the text-to-speech API, responses aren't returned in real time with the Long Audio API. Audios are created asynchronously, and you can access and download the synthesized audios when it is made available from the batch synthesis service.

You can also use your custom voice to generate audio content through a no-code [Audio Content Creation tool](#), and choose to save your text input or output audio content with the tool in Azure storage.

Data processing for Custom Neural Voice Lite

Custom Neural Voice Lite is a project type in public preview that allows you to record 20-50 voice samples on Speech Studio and create a lightweight custom voice model for demonstration and evaluation purposes. Both the recording script and the testing script are pre-defined by Microsoft. The synthetic voice model created using the Custom Neural Voice Lite project could be deployed and used at your discretion, after you apply and full access to Custom Neural Voice is granted.

The synthetic voice and the related audio recording submitted via the Speech Studio will automatically be deleted within 90 days from the Speech Studio portal unless you decide to deploy the synthetic voice, in which case, you will control the duration of its retention. If the Voice Talent would like to have the synthetic voice and the related audio recordings deleted before 90 days, they can delete them on the portal directly, or contact their enterprise to do so.

Before you can deploy the Synthetic Voice model created using a Custom Neural Voice Lite project, it's required that the Voice Talent provide an additional audio recording of the Voice Talent acknowledging the synthetic voice will be used by their enterprise outside of the demonstration and evaluation purpose.

Data storage and retention

Recorded statement and Speaker Verification data: The voice signatures are used by Microsoft solely for the purposes of speaker verification or as otherwise necessary to investigate misuse of the services. The voice signatures will be retained only for the time duration necessary to perform such speaker verification, which may occur from time to time. Microsoft may require this verification before allowing you to train or retrain custom voice models in the Speech Studio, or as otherwise necessary. Microsoft will retain the recorded statement file and voice talent profile data for as long as necessary in order to preserve the security and integrity of Speech in Azure Cognitive Services.

Custom Neural Voice models: While you maintain the exclusive usage rights to your Custom Neural Voice model, Microsoft may independently retain a copy of Custom Neural Voice models for as long as necessary. Microsoft may use your Custom Neural Voice model for the sole purpose of protecting the security and integrity of Microsoft Azure Cognitive Services.

Microsoft will secure and store a copy of Voice Talent's recorded statement and Custom Neural Voice models with the same high level security that it uses for its other Azure Services. Learn more at [Microsoft Trust Center](#).

Training data: You submit voice training data to generate voice models via [Speech Studio](#), and it will be retained and stored by default in Azure storage (See [Azure Storage encryption for data at REST](#) for details). You can access and delete any of the training data used to build the voice models via Speech Studio.

You can manage storage of your training data via [BYOS \(Bring Your Own Storage\)](#). With this storage method, training data may be accessed only for the purposes of voice model training and will otherwise be stored via BYOS.

Text input for speech synthesis: Microsoft does not retain or store the the text that you provide with the real-time synthesis TTS API. Scripts provided via the [Long Audio API](#) for TTS are stored in Azure storage to process the batch synthesis request. The input text can be deleted via the [delete](#) API at any time.

Output audio content: Microsoft does not store the audio content that are generated with the real-time synthesis API. If you are using the [Long Audio API](#), the output audio content is stored in Azure storage. Thse audios can be removed at any time via the [delete](#) operation.

To learn more about Microsoft's privacy and security commitments visit the [Microsoft Trust Center](#).

Use cases for Speaker Recognition

Article • 12/06/2022 • 6 minutes to read

What is a Transparency Note?

An AI system includes not only the technology, but also the people who will use it, the people who will be affected by it, and the environment in which it's deployed. Creating a system that is fit for its intended purpose requires an understanding of how the technology works, its capabilities and limitations, and how to achieve the best performance.

Microsoft provides *Transparency Notes* to help you understand how our AI technology works. They include the choices system owners can make that influence system performance and behavior, and the importance of thinking about the whole system, including the technology, the people, and the environment. You can use Transparency Notes when developing or deploying your own system, or share them with the people who will use or be affected by your system.

Transparency Notes are part of a broader effort at Microsoft to put our AI principles into practice. To find out more, see [Microsoft's AI principles](#).

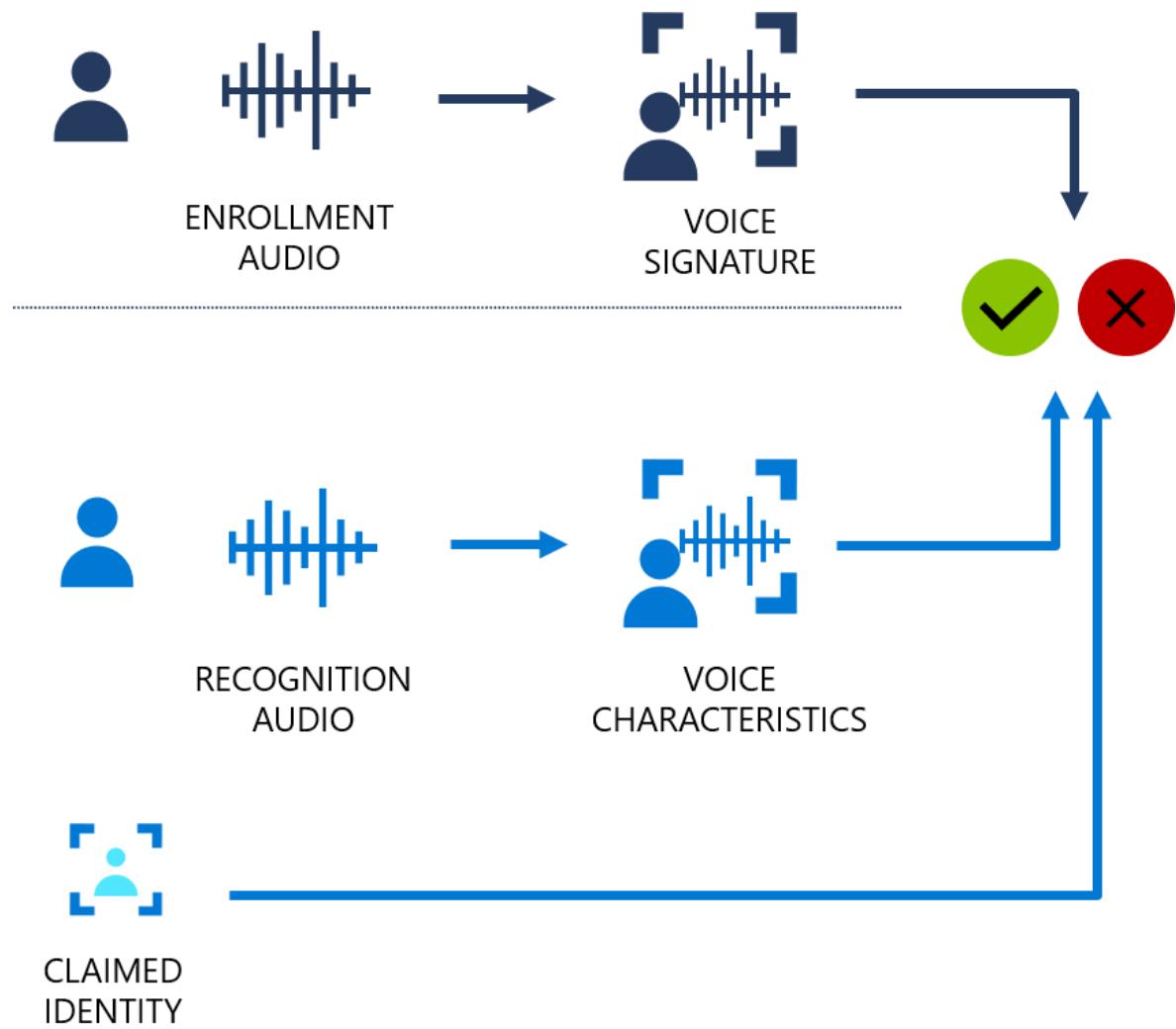
Introduction to Speaker Recognition

Speaker Recognition is an AI feature that can identify an individual speaking in an audio clip. The human voice has unique characteristics that can be associated with an individual. Speaker Recognition can recognize speakers by comparing the unique voice characteristics of incoming speech with registered voice signatures. For more information, see [Speaker Recognition](#).

The basics of Speaker Recognition

Speaker Recognition capabilities are provided through two APIs:

- *Speaker Verification* allows you to determine scenarios like “Is this Anna speaking?”. It verifies an individual’s identity by comparing the voice characteristics of their speech with the registered voice signature of the claimed identity.



- *Speaker Identification* allows you to determine scenarios like "Who is speaking, Anna, Isha, or Jing?". It attributes speech to individual speakers within a group of enrolled individuals.

Term	Definition
Voice signature	Also called <i>template</i> or <i>voiceprint</i> . It's a numeric vector that represents an individual's voice characteristics, extracted from audio recordings of a person speaking. The original audio recordings can't be interpreted or reconstructed based on a voice signature. Voice signature quality is a key determinant of how accurate your results are.
Enrollment	Enrollment is the process of creating voice signatures from the audio files of individuals' speech, so they can be recognized at a later time. When a person is enrolled to a recognition system, that person's template is also associated with a primary identifier ¹ that will be used to determine which voice signature to compare with the recognition speech input.
Recognition	During Recognition, audio of a person speaking is compared against one or more voice signatures. The process is called <i>verification</i> if the audio is compared with one specific voice signature. It's called <i>identification</i> if the audio is compared with more than one voice signature to identify the speaker.

Term	Definition
Text-dependent speaker verification	Also called <i>active verification</i> . The speaker chooses a specific passphrase (set of words) to be spoken during both enrollment and verification phases. During verification, the system recognizes the passphrase text and compares it with the enrollment passphrase. The result is based on both voice signature match and passphrase match.
Passphrase signature	In the enrollment audio of text-dependent APIs, the chosen passphrase is recognized to text. Then both the voice signature and the passphrase text are stored. The unique passphrase, such as "My voice is my passport verify me," is called a <i>passphrase signature</i> . The passphrase signature is also compared with the text of speech audio input during recognition.
Text-independent speaker verification	Also called <i>passive verification</i> . Speakers aren't required to speak pre-defined words, instead speakers can use any phrase. The voice signature is used during verification, but the speech content isn't considered. During recognition, speakers don't necessarily need to use the same phrase they did during enrollment. Longer audio recordings are recommended during enrollment to achieve reliable performance.
Activation phrase	It is a predefined phrase that the speaker has to read in the beginning of the enrollment when using text-independent APIs when active enrollment is enabled. Although speakers can use any phrases during the recognition process in text-independent speaker verification or identification, with active enrollment enabled, Microsoft requires the speaker to read this activation phrase first. After the activation step, the speaker can continue enrollment by using any phrases.

¹ Developers can associate the GUID (globally unique identifiers) generated by Microsoft with an individual's primary identifier to support verification of that individual. Speaker Recognition doesn't store primary identifiers, such as customer IDs, with voice signatures. Instead, Microsoft associates stored voice signatures with random GUIDs.

Limited Access to Speaker Recognition

Speaker Recognition is a Limited Access service, and registration is required for access to some features. To learn more about Microsoft's Limited Access policy visit aka.ms/limitedaccesscogservices. Certain features are only available to Microsoft managed customers and partners, and only for certain use cases selected at the time of registration.

Approved use cases

The following use cases are approved for customers:

- **Customer identity verification:** Call center or interactive voice response systems can use speaker verification to help verify a customer's identity when a caller seeks to access the customer's information or to take action with respect to the customer's account.
- **Multi-factor authentication:** Verify identity by matching voice characteristics against registered voice signature as one factor to enhance security.
- **Smart device personalization:** Voice-enabled interaction devices, such as smart vehicles or smart speakers, can use Speaker Recognition to provide personalized content. For example, you can play different types of movies or music in response to voice commands in a household by using the text-independent Speaker Verification API.
- **Speaker identification for meetings:** Identify individual speakers from a meeting transcription or in captions.
- **[Public Sector Only] Speaker identification or verification to:** (a) assist law enforcement or court officials in the prosecution or defense of a serious crime or to identify a missing person, in all cases only to the extent specifically authorized by a court order issued in a jurisdiction that maintains a fair and independent judiciary, and provided that the person sought to be identified or verified is not a minor; OR (b) assist officials of duly empowered international organizations in the prosecution of abuses of international criminal law, international human rights law, or international humanitarian law, provided that the person sought to be identified or verified is not a minor.

Considerations when using Speaker Recognition

- **Avoid using for recognizing multiple speakers in a speech input:** Speaker Recognition can't recognize more than one person in a single speech input. Speaker Recognition is intended to take in one person's speech input and compare it to one or more voice signatures.
- **Avoid using as a sole factor in authentication where security is important:** Speaker Recognition is not designed to differentiate a synthesized voice or recordings of a voice from a live human speaker. Carefully consider scenarios with a risk of spoofing. Speaker Recognition shouldn't be used as the sole factor in authenticating a user in applications where security is the goal, such as access to financial information or physical security.
- **Actively enroll users:** Voice signatures contain speakers' biometric voiceprint characteristics. To help prevent misuse of Speaker Recognition, Microsoft provides an active enrollment feature for users of text-independent APIs through an activation step. The activation step indicates the speakers' active participation in

creating their voice signatures and is intended to help avoid the scenario in which speakers are enrolled without their awareness. Be advised that this activation step does not alleviate customer's legal obligations to ensure it has received all necessary permissions and consents from its users for the purposes of its processing, retention, and intended uses of speaker signatures created.

- **Limit the number of candidates for speaker identification:** Speaker Identification API can only take up to 50 candidates to compare the speech input against in an API call.

Next steps

- [Speaker Recognition overview](#)
- [Limited access for Speaker Recognition](#)
- [Transparency Note for Speaker Recognition](#)

Characteristics and limitations of Speaker Recognition

Article • 12/02/2022 • 11 minutes to read

In this section, we'll review what performance means for Speaker Recognition, best practices for improving performance, limitations, and fairness as it relates to the Speaker Recognition feature.

General performance guidelines

Because Speaker Recognition serves various uses, there's no universally applicable estimate of accuracy for every system. In some cases, such as meeting transcription, Speaker Recognition is a building block that you can combine with other components to create an end-to-end solution. The performance of Speaker Recognition is therefore affected by these other components.

Accuracy

The accuracy of both speaker verification and speaker identification can be measured by False Rejection Rate (FRR) and False Acceptance Rate (FAR). Developers must balance the trade-off between the rates of FRR and FAR in each specific scenario.

Let's take text-independent speaker verification as an example. The following table shows the possible outcomes where two individuals request access to a building or system that uses Speaker Recognition. These two individuals are *Speaker A* who is enrolled, and an imposter who is claiming to be *Speaker A*. Speaker Recognition tests the audio of speech input against the saved voice signature of *Speaker A*.

Outcome	Details
Correct accept or true positive	The system correctly accepts an access attempt by <i>Speaker A</i> .
Correct reject or true negative	The system correctly rejects an access attempt by the imposter.
False accept or false positive	The system incorrectly accepts an access attempt by the imposter.
False reject or false negative	The system incorrectly rejects an access attempt by <i>Speaker A</i> .

The measurement for Speaker Identification API is similar, except that a speech input is tested against one of multiple known speakers.

The consequences of a false positive or a false negative will vary depending upon the intended use of the speaker recognition system. The following examples illustrate this variation and the how choices you make in designing the system affect the people who are subject to it. The design of the whole system, including fallback mechanisms, determines the consequences for people when errors occur.

- **Signing into a banking app:** Speaker Recognition can provide an added layer of security in addition to a PIN. A false positive for this application reduces customer security because it results in an incorrect match, while a false negative could prevent the customer from accessing their account. Because the purpose of the banking system is security, system owners will likely want to ensure that false positives are minimized by requiring higher confidence level thresholds. This however, may also result in most errors being made as false negatives (account access fails). To address this limitation, system owners should provide an alternative mechanism for enabling its users to access their application, for example, offering alternative sign-in via an access code notification to the customer's phone. The customer's experience might be less convenient in this case, but account access isn't blocked while security is still prioritized.
- **Receiving personalized content on a smart device:** Personal devices can use Speaker Recognition to respond to the device owner's voice command for personalized content. A false positive increases unnecessary activation, while a false negative might result in no response to the user command. Here, if the purpose of the system is convenience and efficiency and not security, some false positives may be acceptable to the system provider. In this case, false negatives are usually minimized, while most errors will be the result of false positives. Conducting multiple enrollments of the device owner's voice can help Speaker Recognition perform more accurately for that voice over time.

Match scores, match thresholds, and matched conditions

System configuration influences system accuracy. By comparing the pre-enrolled speaker profile and input audio, Speaker Recognition outputs a match score to indicate the similarity of the comparison result and uses a threshold to decide whether to accept or reject the input as a match. It's important to understand the trade-off between the rates of false positives and false negatives. The table below gives more details description.

Term	Definition
Match score or similarity score	Match scores range from 0 to 1. High match scores indicate that it's more likely that the speech input comes from the same person with the enrolled voice signature.
Match threshold	A match threshold is a configurable value between 0 and 1 that determines the match score required to be considered a positive match. If the match threshold is set to 0, then the system will accept any match score and the false accept rate would be high; if the match threshold is set to 1, then it will only accept a voice input with a 1 (100%) match score and the false reject rate would be high. Speaker Recognition API has a default match threshold that you can change to suit your application.

Because the optimal threshold varies highly with use cases or scenarios, the Speaker Verification API decides whether to accept or reject based on a default threshold of 0.5. The threshold is a compromise between the requirements of high security applications and high convenience applications. Adjust the threshold for each scenario and validate the results by testing with your data.

Outcome	Details
Correct accept or true positive	When the real <i>Speaker A</i> requests access as <i>Speaker A</i> , the system returns a match score of 0.8, which is above the default threshold of 0.5. The system correctly accepts the access attempt.
Correct reject or true negative	When an imposter requests access as <i>Speaker A</i> , the system returns 0.2, which is below the default threshold of 0.5. The system correctly rejects the access attempt.
False accept or false positive	When an imposter requests access as <i>Speaker A</i> , the system returns 0.6, which is above the threshold of 0.5. The system incorrectly accepts the access attempt.
False reject or false negative	When the real <i>Speaker A</i> requests access as <i>Speaker A</i> , the system returns 0.4, which is below the threshold of 0.5. The system incorrectly rejects the access attempt.

How should a match threshold be selected?

The optimal threshold varies greatly according to different scenarios. If the accuracy of results isn't optimal for a particular scenario, you can adjust the default threshold and fine-tune it based on the results of testing on your own data. You can gather real-world

data to evaluate if audio samples are labeled with the correct speaker identities. This type of dataset is known as a ground truth evaluation dataset.

You can then feed the evaluation data into Speaker Verification API and keep the returned match scores. Compare ground truth labels to the output of the system. With this evaluation of system performance, you can establish the overall FRR and FAR on the threshold of interest, and the distribution of errors between false positives and false negatives. Ground truth evaluation data should include adequate sampling of diverse people who will be subject to recognition, so that you can understand performance differences between groups of people and take corrective action.

With your evaluation results, you can adjust the threshold to better suit the scenario. For example, because the customer identity verification scenario usually prefers high security over convenience, you might set the threshold higher than the default threshold to reduce false accept errors. By contrast, because the personalization scenario might prefer high convenience over security, you might set the threshold lower than the default to reduce the false reject errors. Based on each evaluation result, you can iteratively adjust the match threshold until the trade-off between false positives and false negatives meets your objectives.

Best practices to improve accuracy

Here are some specific actions you can take to ensure the best results from your speaker recognition system.

Plan for variations in subject and environment

You can improve system accuracy by having matching conditions between the enrollment audio and the recognition audio. Matching conditions means using the same device or microphone, or having a consistent acoustic environment, or a consistent speaking style of the speaker (for example, a reading style versus a conversational style).

Achieving matching conditions can be difficult. Although Speaker Recognition has been trained with data of various acoustic conditions, it's still better to support multiple enrollments to accommodate varying conditions. The APIs support multiple enrollments, so you can enroll the speaker over time under the range of conditions you expect the system to be used in. Speaker Recognition can create a more robust audio signature with multiple enrollments.

For text-independent verification or identification, the duration of the enrollment or recognition audio input also affects accuracy. Effective speech length measures the total length of speech, excluding silence and non-speech segments. We use a speech

detection module to count the total usable amount of audio. For example, a user might send 30 seconds of audio for text-independent enrollment, but the effective speech length might only be 15 seconds. In cases where the speech content of enrollment and recognition is different (text-independent systems), the longer the effective speech, the better the performance. When active enrollment is enabled, the length of the activation phrase at the beginning of the enrollment is counted in the total speech length of enrollment.

Meet specifications

The following specifications are important to be aware of:

- **Audio format:** The current system only supports mono channel 16 bit, 16 kHz PCM-encoded WAV. Any data in the 8-kHz sampling rate should be up sampled to 16kHz before being sent to the service.
- **Maximum number of enrolled voices to compare:** In Speaker Identification API, the speech input is compared to a specified list of enrolled voices. The fewer candidates there are to compare with, the more accurate the result will be. In the current Speaker Identification API, the limit is 50 enrolled voices to compare.

Design the system to support human judgment

We recommend using Speaker Recognition capabilities to support people making accurate and efficient judgments, rather than fully automating a process. Meaningful human review is important to:

- Detect and resolve cases of misidentification or other failures.
- Provide support to people who believe their results were incorrect.

For example, in call center scenarios, a legitimate customer can be rejected due to having a sore throat. In this case, a human agent can intervene and help the customer verify their identity by asking security questions.

Use multiple factors for authentication

It's always recommended to build multifactor authentication for scenarios that require high security. Using another security factor can help mitigate spoofing attacks.

Mitigate the risk of replay (or spoofing) attacks

Speaker Recognition isn't intended to determine whether the audio is from a live person speaking or is an audio recording of an enrolled speaker. For verification scenarios requiring high security, in addition to using speech as a secondary authentication factor, consider mitigations such as generating random phrases for the speaker to read at runtime. This can mitigate the risk of replay attacks or attacks that use a non-parameter-based, synthesized voice.

Your application can send separate requests to the text-independent Speaker Verification API and the speech-to-text API. By combining these, the application can help confirm the speaker's identity.

Fairness

At Microsoft, we strive to empower every person on the planet to achieve more. An essential part of this goal is working to create technologies and products that are fair and inclusive. Fairness is a multi-dimensional, sociotechnical topic and impacts many different aspects of our product development. You can learn more about our approach to fairness [here](#).

One dimension of this goal is to consider how well the system performs for different groups of people. Research has shown that without conscious effort focused on improving performance for all groups, it is often possible for the performance of a system to vary across groups based on factors such as race, ethnicity, region, gender, and age.

We test our systems on various factors, including demographic factors such as gender, ethnicity, and age. Each application is different, and our testing might not perfectly match your context or cover all scenarios required for your use case. We encourage developers to thoroughly evaluate error rates for the service with real-world data that reflects your use case. This should include testing with users from different demographic groups and with different speech characteristics.

For Speaker Recognition, there are some observable differences in performance across languages on some specific datasets. We have tuned and evaluated on eight languages (English, French, Spanish, Chinese, German, Italian, Japanese, and Portuguese) for text-independent verification and identification APIs. You can see more details on Speaker Recognition's language and locale support [here](#). Speaker Recognition hasn't been tested with data representing minors under the age of 18 or people with speech disorders.

Evaluating and integrating Speaker Recognition for your use

Before a large-scale deployment or rollout of any speaker recognition system, system owners should conduct an evaluation phase. Do this evaluation in the context where you'll use the system, and with people who will interact with the system. Work with your analytics and research teams to collect ground truth evaluation data to:

1. Establish baseline accuracy, false positive and false negative rates.
2. Choose an appropriate match threshold for your scenario.
3. Determine whether the error distribution is skewed towards specific groups of people.

Evaluation is likely to be an iterative process. For example, you can start with 50 speakers and 20 trials for each speaker. An evaluation should reflect your deployment environment and any variations in that environment, such as microphone channel and noise level. Use ground truth evaluation data that represents a diversity of people and speaker styles.

In addition to analyzing accuracy data, you can also analyze feedback from the people making judgments based on the system output. Further, you can analyze satisfaction data from the people who are subject to recognition, and feedback from existing customer voice channels, to help tune the system and ensure successful engagement.

Next steps

- [Speaker Recognition overview](#)
- [Limited access for Speaker Recognition](#)
- [Transparency Note for Speaker Recognition](#)

Limited Access to Speaker Recognition

Article • 08/17/2022 • 2 minutes to read

As part of Microsoft's commitment to responsible AI, Speaker Recognition is designed with the intention of protecting the rights of individuals and society and fostering transparent human-computer interaction. For this reason, Microsoft's Speaker Recognition feature is a Limited Access feature available by registration only, and only for certain use cases.

Registration process

As a Limited Access feature, Speaker Recognition requires registration. Only customers managed by Microsoft, meaning those who are working directly with Microsoft account teams, are eligible for access. Customers who wish to use this feature are required to register by [submitting a registration form](#). The use of Speaker Recognition is limited to the use case selected at the time of registration. Microsoft may require customers to re-verify this information.

The Speaker Recognition is made available to customers under the terms governing their subscription to Microsoft Azure Services (including the [Service Specific Terms](#)). Please review these terms carefully as they contain important conditions and obligations governing your use of Speaker Recognition.

Learn more about the legal terms that apply to this feature [here](#).

Help and support

FAQ about Limited Access features can be found [here](#).

If you need help with Speaker Recognition, find support [here](#).

Report abuse of Speaker Recognition [here](#).

Next steps

- [Speaker Recognition overview](#)
- [Limited Access with Cognitive Services](#)
- [Transparency Note for Speaker Recognition](#)

Guidance for integration and responsible use with Speaker Recognition

Article • 06/20/2022 • 3 minutes to read

Microsoft wants to help you responsibly develop and deploy solutions that use the Speaker Recognition feature. We're taking a principled approach to upholding personal agency and dignity by considering the AI systems' fairness, reliability and safety, privacy and security, inclusiveness, transparency, and human accountability. These considerations reflect our commitment to developing Responsible AI.

General guidelines

When you're getting ready to deploy Speaker Recognition, the following activities help to set you up for success:

- **Understand what it can do:** Fully assess the capabilities of any AI system you're using to understand its capabilities and limitations. Understand how it will perform in your particular scenario by testing it with real life conditions and diverse user data that reflect your context, including fairness considerations.
- **Respect an individual's right to privacy:** Only collect biometric data and information from individuals for lawful and justifiable purposes. Obtain meaningful consent for your collection and intended uses.
- **Legal review:** Obtain appropriate legal advice to review your biometric solution, particularly if you'll use it in sensitive or high-risk applications. In some jurisdictions, there are specific legal requirements that govern the collection, use, storage, and security of biometric data. You're responsible for compliance with all applicable laws and regulations that apply to your solution deployment.
- **Build trust with affected stakeholders:** Communicate the expected benefits and potential risks to affected stakeholders. Help people understand why the data is needed and how the use of the data will lead to their benefit. Describe data handling in an understandable way.
- **Customer feedback loop:** Provide a feedback channel that allows users and individuals to report issues with Speaker Recognition after it's been deployed. This mechanism should also allow for feedback about fairness. Monitor and improve the AI-powered product or feature on an ongoing basis. Be ready to implement

any feedback and suggestions for improvement. Establish channels to collect questions and concerns from affected stakeholders (people who might be directly or indirectly impacted by the system, including employees, speakers, and the general public). Potential feedback channels include features built into application experiences, or an easy-to-remember email address for feedback.

- **Train and support end users:** The people who use the output of your solution, or who decide whether the output is correct, might not have experience collaborating with AI systems. This can result in mismatched judgments or the introduction of unfair bias. You can empower these users by evaluating where mismatches might occur, and providing training and ongoing support.

Recommendations for preserving privacy

A successful privacy approach empowers individuals with information and provides controls and protection to preserve their privacy.

- Ensure that you've received the appropriate permissions from users to conduct speaker recognition, and be clear about your intended scope of use and its duration. Don't share data without explicit consent from affected stakeholders and data owners, and minimize the data that you do share.
- Text-dependent verification requires the speaker's active participation by picking and reading a specific passphrase at enrollment. Text-independent verification or identification allows everyday language for enrollment and recognition, so the speaker's active participation isn't ensured. Ask the speaker to read a specific text in the enrollment to ensure active participation by the speaker and increase their awareness of using Speaker Recognition.
- Speaker Recognition doesn't store primary identifiers (for example, customer IDs) alongside voice signatures or audio of a speaker sent to the Speech service for enrollment or recognition. Microsoft associates this data with random GUIDs (globally unique identifiers). It's up to you to manage the user identity mapping between these GUIDs and your users. You're responsible for ensuring that this data is securely stored and managed.
- Provide a mechanism to allow the affected stakeholders and data owners to unenroll from Speaker Recognition and delete their data at any time. Implement a data retention strategy and plan that only retains enrollment data from your users for as long as needed to provide the services. Delete user data after some period of time, such as upon user termination or a specified period inactivity.

Next steps

Data and privacy for Speaker Recognition

Article • 12/02/2022 • 5 minutes to read

ⓘ Note

This article is provided for informational purposes only and not for the purpose of providing legal advice. We strongly recommend seeking specialist legal advice when implementing Speaker Recognition.

While the Speaker Recognition feature was designed with compliance, privacy, and security in mind, you're responsible for its use and the implementation of this technology. Be aware that the laws governing biometric recognition technologies often vary internationally and domestically, including at the federal, state, and local levels. In addition to regulating allowed use cases, some jurisdictions impose special legal requirements for the permissions governing collection, transfer, online processing, and storage of biometric data, particularly when used for identification or verification. Before using Speaker Recognition and Azure for the collect, transfer, processing, and storage of any data subject's biometric data, you must ensure compliance with the relevant legal requirements that apply to your service application.

ⓘ Note

For our customer's convenience, please consider utilizing the following disclosure regarding Microsoft's role when you use Azure Cognitive Services Speaker Recognition with your end users:

[Company] uses Microsoft Speaker Recognition technology to process [Company's] users' biometric data as its service provider ("Processor"). Microsoft may process and store your enrollment audio and voice signatures for the purposes of providing speaker verification and/or identification services on [Company]'s behalf, and only as instructed by [Company]. Microsoft will store this data as long as [Company] requests, which shall be no longer than a limited grace period after the date when (i) [Company] ceases to have a relationship with Microsoft or (ii) when [Company] requests deletion.

What data does Speaker Recognition process?

Speaker Recognition processes the following types of data:

Enrollment audio: Before enrollment, customers request a random GUID from the service. During the enrollment phase, customers send a speaker's audio input and the GUID to generate a voice signature and a passphrase signature match.

Enrolled voice signature: This is the numeric vector that represents an individual speaker's voice characteristics, extracted from audio recordings.

Passphrase signature: This is a pre-defined phrase, for example, 'My voice is my profile'. During enrollment of a speaker, enrollment audio will be processed through Azure speech recognition service in order to confirm that the text from that audio matches the passphrase required.

Recognition audio: After the enrollment of the customer's speaker(s), the customer sends audio input along with the relevant GUIDs to be processed to the Speaker Recognition feature and voice signatures are processed to determine if the audio matches the enrolled speaker(s) voice signatures. If using text-dependent speaker verification, the passphrase signature is also transcribed by speech recognition to determine if there is a passphrase match.

How Does Speaker Recognition process this data?

- **Text-dependent speaker verification system:**
 - **Enrollment Phase:** The speaker's voice is enrolled by saying a passphrase from a set of predefined phrases. Speaker Recognition extracts voice features from these audio recordings to form a unique voice signature, and the text of the chosen passphrase is verified through speech recognition. Both the voice signature and the passphrase are stored for the recognition phase.
 - **Recognition Phase:** The speaker needs to speak the same passphrase from the enrollment phase. During the recognition phase, Speaker Recognition extracts the voice features from the audio and recognizes the passphrase in the audio. Then the service compares both the voice features and the passphrase against the voice signature and the passphrase signature in the record. Only when both the voice and passphrase match the enrolled voice signature and passphrase signature by achieving the match threshold, the response returns *Accept* and provides a similarity score.
- **Text-independent speaker verification or identification system:**
 - **Activation and Enrollment Phase:** Speaker Recognition extracts the voice features from a speaker's enrollment audio to form a unique voice signature.

There are two steps in the enrollment:

1. In the activation step when active enrollment is enabled, the speaker's voice is recorded saying an activation phrase defined by Microsoft. The text of the activation phrase is transcribed using speech recognition technology provided by Azure Speech Services, and then compared with the predefined activation phrase. If a match is found, the audio of the activation phrase is stored as the voice signature. Unlike the passphrase in the text-dependent system, the text of the activation phrase won't be stored.
 2. The speaker can continue enrolling by speaking in everyday language to improve the performance of future verification or identification. This step is optional but recommended. After the activation step, there's no restriction on what the speaker says in the audio. The speech content will not be processed, only the voice signature will be extracted and stored for the recognition phase.
- **Recognition Phase:** After the activation and enrollment phase, the customer may send audio to the Speaker Recognition feature. Speaker Recognition will extract the voice features from the audio. For speaker identification system, Speaker Recognition compares these voice features against a list of voice signatures associated with the specified GUIDs, one by one. The service then ranks the list of GUIDs based on their similarity scores and returns up to five GUIDs and their similarity scores. For speaker verification system, Speaker Recognition compares these voice features against the voice signature in the record. When the voice matches the enrolled voice signature by achieving the match threshold, the response returns *Accept* and provides a similarity score.

How is data retained, and what customer controls are available?

Enrollment audio, voice signatures, and passphrases are all securely stored in the customer's Azure tenancy. This data is associated with random GUIDs only and Microsoft is not able to connect any customer names, IDs (other than the random GUID), or other personal information to the enrollment audio, voice signatures, or passphrase signatures. Developers can create, update, and delete enrollment data (both the voice signature and related enrollment audio) for individual speaker through API calls. When the subscription is deleted, all the speaker enrollment data associated with the subscription is also deleted. The following table summarizes Speaker Recognition data retention and controls.

Data type	Retention	Customer controls
Enrollment audio	Microsoft assigns randomly generated GUIDs (globally unique identifiers) to the enrollment audio for each individual. This GUID is also associated with the voice signature.	You can manage and delete all data that is stored associated with any individual GUID or all GUIDs.
Enrolled voice signature	Microsoft assigns a randomly generated unique identifier to every enrolled voice signature.	You can manage and delete all data that is stored associated with any individual GUID or all GUIDs.
Passphrase signature	For text-dependent enrollment, the passphrase signature is stored as text, and it's stored with the voice signature.	You can manage and delete all data that is stored associated with any individual GUID or all GUIDs.
Recognition audio	Audio sent to the service aren't stored after they're analyzed against the voice signature.	There are no customer controls for this data type.

To learn more about privacy and security commitments, see the [Microsoft Trust Center](#).

Next steps

- [Speaker Recognition overview](#)
- [Limited access for Speaker Recognition](#)
- [Transparency Note for Speaker Recognition](#)

Azure Cognitive Services support and help options

Article • 07/22/2022 • 2 minutes to read

Are you just starting to explore the functionality of Azure Cognitive Services? Perhaps you are implementing a new feature in your application. Or after using the service, do you have suggestions on how to improve it? Here are options for where you can get support, stay up-to-date, give feedback, and report bugs for Cognitive Services.

Create an Azure support request

A

Explore the range of [Azure support options and choose the plan](#) that best fits, whether you're a developer just starting your cloud journey or a large organization deploying business-critical, strategic applications. Azure customers can create and manage support requests in the Azure portal.

- [Azure portal](#)
- [Azure portal for the United States government](#)

Post a question on Microsoft Q&A

For quick and reliable answers on your technical product questions from Microsoft Engineers, Azure Most Valuable Professionals (MVPs), or our expert community, engage with us on [Microsoft Q&A](#), Azure's preferred destination for community support.

If you can't find an answer to your problem using search, submit a new question to Microsoft Q&A. Use one of the following tags when you ask your question:

- [Cognitive Services](#)

Vision

- [Computer Vision](#)
- [Custom Vision](#)
- [Face](#)
- [Form Recognizer](#)
- [Video Indexer](#)

Language

- Immersive Reader
- Language Understanding (LUIS)
- QnA Maker
- Language service
- Translator

Speech

- Speech service

Decision

- Anomaly Detector
- Content Moderator
- Metrics Advisor
- Personalizer

Azure OpenAI

- Azure OpenAI

Post a question to Stack Overflow



For answers on your developer questions from the largest community developer ecosystem, ask your question on Stack Overflow.

If you do submit a new question to Stack Overflow, please use one or more of the following tags when you create the question:

- Cognitive Services ↗

Vision

- Computer Vision ↗
- Custom Vision ↗
- Face ↗
- Form Recognizer ↗
- Video Indexer ↗

Language

- Immersive Reader ↗
- Language Understanding (LUIS) ↗

- [QnA Maker ↗](#)
- [Language service ↗](#)
- [Translator ↗](#)

Speech

- [Speech service ↗](#)

Decision

- [Anomaly Detector ↗](#)
- [Content Moderator ↗](#)
- [Metrics Advisor ↗](#)
- [Personalizer ↗](#)

Azure OpenAI

- [Azure OpenAI ↗](#)

Submit feedback

To request new features, post them on <https://feedback.azure.com> ↗. Share your ideas for making Cognitive Services and its APIs work better for the applications you develop.

- [Cognitive Services ↗](#)

Vision

- [Computer Vision ↗](#)
- [Custom Vision ↗](#)
- [Face ↗](#)
- [Form Recognizer ↗](#)
- [Video Indexer ↗](#)

Language

- [Immersive Reader ↗](#)
- [Language Understanding \(LUIS\) ↗](#)
- [QnA Maker ↗](#)
- [Language service ↗](#)
- [Translator ↗](#)

Speech

- [Speech service ↗](#)

Decision

- [Anomaly Detector ↗](#)
- [Content Moderator ↗](#)
- [Metrics Advisor ↗](#)
- [Personalizer ↗](#)

Stay informed

Staying informed about features in a new release or news on the Azure blog can help you find the difference between a programming error, a service bug, or a feature not yet available in Cognitive Services.

- Learn more about product updates, roadmap, and announcements in [Azure Updates ↗](#).
- News about Cognitive Services is shared in the [Azure blog ↗](#).
- [Join the conversation on Reddit ↗](#) about Cognitive Services.

Next steps

[What are Azure Cognitive Services?](#)