Contents

- Speech service documentation
- Overview
  - Speech service overview
  - Language and voice support
  - Region support
  - Pricing
  - Quotas and limits
- Speech-to-text
  - Speech-to-text documentation
  - Speech-to-text overview
  - Speech-to-text quickstart
  - How-to guides
    - Change speech recognition source language
    - Use compressed audio input formats
    - Select a specific microphone input device
    - Language identification
    - Use language identification
    - Use the batch transcription API
    - Migrate from v2.0 to v3.0
    - Use Pronunciation Assessment
    - Control and monitor connection status
    - Customization
      - What is Custom Speech?
      - Prepare data for Custom Speech
      - Inspect Custom Speech data
      - Evaluate and improve Custom Speech accuracy
      - Train and deploy a Custom Speech model
      - CI/CD for Custom Speech
      - Customize Pronunciation

- Create human-labeled transcriptions
- Model and Endpoint Lifecycle
- Responsible use of AI
  - Speech-to-text
    - Transparency note
      - Use cases for Speech-to-text
      - Characteristics and limitations
    - Integration and responsible use
    - Data, privacy, and security
  - Pronunciation Assessment
    - Transparency Note
      - Use cases for Pronunciation Assessment
      - Characteristics and limitations
- Tutorials
  - Improve accuracy with tenant models
- Reference
  - C++
  - C#
  - Go
  - Java
  - JavaScript
  - Objective-C
  - Python
  - Release notes
  - Speech-to-text REST API
- Resources
  - Language support
  - Region support
  - Ingestion Client
  - Universal Phone Set
  - Speech-to-text FAQ
Text-to-speech
- Text-to-speech documentation

Text-to-speech overview
Text-to-speech quickstart
How-to guides
  Improve synthesis with SSML
  Lower speech synthesis latency using Speech SDK
  Synthesize speech with Long Audio API
  Get facial pose events for lip-sync
  Customization
    Custom Neural Voice
      What is Custom Neural Voice?
      Get started with Custom Neural Voice
      Prepare data for training
      Create and use your voice model
      How to record voice samples
    Audio Content Creation
      Audio Content Creation
Responsible use of AI
  Custom Neural Voice
    Transparency Note
      Use cases for Custom Neural Voice
      Characteristics and limitations
    Integration and responsible use
      Limited access
      Responsible deployment of synthetic speech
      Disclosure for voice talent
      Disclosure design guidelines
      Disclosure design patterns
      Code of conduct
    Data, privacy and security
Reference
  C++
  C#

- Java
- JavaScript
- Objective-C
- Python
- Release notes
- Text-to-speech REST API
- Resources
  - Neural voice support
  - Region support
  - SSML phonetic sets
- Speech translation
  - Speech translation documentation
  - Speech translation overview
  - Speech translation quickstart
  - How-to guides
    - Translate and synthesize text with REST APIs
  - Reference
    - C++
    - C#
    - Java
    - JavaScript
    - Objective-C
    - Python
    - Release notes
  - Resources
    - Language support
    - Region support
    - Speech translation FAQ
- Intent recognition
  - Intent recognition documentation
  - Intent recognition overview
  - Intent recognition quickstart

How-to guides
Recognize speech intents with LUIS, C#
Recognize speech intents with simple pattern matching, C++
Reference
Speech SDK
C++
C#
Java
JavaScript
Objective-C
Python
Release notes
LUIS service
LUIS developer resources
Resources
Language support
Region support
Speaker Recognition
Speaker Recognition overview
Speaker Recognition quickstart
Responsible use of AI
Transparency note
Use cases for Speaker Recognition
Characteristics and limitations
Integration and responsible use
Limited access
General guidelines
Data, privacy, and security
Reference
Speaker Recognition REST API
C++
C#

- Java
- JavaScript
- Objective-C
- Python
- Release notes
- Keyword recognition
  - Keyword recognition overview
  - Custom Keyword quickstart
  - Recommendations and guidelines
  - Region support
- Scenario deep-dives
  - Conversation scenarios
    - Conversation scenarios documentation
    - Conversation transcription
      - Conversation Transcription overview
      - Real-time Conversation Transcription quickstart
      - How-to guides
        - Handle call center scenarios
        - Asynchronous Conversation transcription
      - Reference
        - Speech SDK
          - C++
          - C#
          - Java
          - JavaScript
          - Objective-C
          - Python
          - Release notes
        - REST APIs
          - Conversation transcription REST APIs
    - Multi-device conversation
      - Multi-device conversation overview

- Multi-device conversation quickstart
- Reference
  - Speech SDK
    - C++
    - C#
    - Java
- Voice assistants
  - Voice assistants documentation
  - Voice assistants overview
  - Custom Commands
    - What is Custom Commands?
    - Quickstarts
      - Create a Custom Commands app
    - How-to guides
      - Develop and customize Custom Commands apps
      - Integrate with client applications
        - Integrate with a client app using Speech SDK
        - Send activity to client application
        - Set up web endpoints
        - Export your Custom Commands app as a skill
        - Update a command from the client app
        - Update a command from a web endpoint
      - Continuous Deployment with Azure DevOps
      - Test Custom Commands applications
      - Debug errors when authoring the app
      - Debug errors when running the app
    - Resources
      - Custom Commands concepts
      - Custom Commands Samples (GitHub)
      - Voice Assistants Samples (GitHub)
      - Voice assistants FAQ
      - Custom Commands encryption of data at rest

- Direct Line Speech
  - What is Direct Line Speech?
  - Quickstarts
    - Create a custom voice assistant
  - Tutorials
    - Voice enable your bot with Speech SDK
  - Resources
    - Voice assistants FAQ
- Voice assistants on Windows
  - Overview of voice assistants on Windows
  - How-to guides
    - Start developing your assistant with Windows
    - Enable voice activation
  - Resources
    - Privacy requirements and guidelines
    - Best practices for assistant experiences
    - Samples and frequently asked questions
- How-to guides
  - Enable logging
  - Track memory usage
  - Enable private endpoint
  - Enable VNet service endpoint
  - Configure OpenSSL for Linux
  - Configure RHEL/CentOS 7
  - Use containers
    - Install and run Speech containers
    - Configure speech containers
    - Batch processing with speech containers
    - Speech containers with Kubernetes and Helm
    - Speech containers on Azure container instances
    - Containers FAQ
    - Cognitive Services containers documentation

- Migration
  - From Bing Speech
  - From the Translator Speech API
- Enterprise readiness
  - Set up Virtual Networks
  - Use customer-managed keys
  - How to backup and recover speech customization resources
- Devices
  - Overview
    - Audio processing overview
    - Speech SDK integration
  - Hardware
    - Microphone array recommendations
  - Reference
    - Speech devices SDK documentation
    - What is the Speech devices SDK?
    - Get the Speech devices SDK
    - Use Speech devices SDK
    - Speech devices SDK samples (GitHub)
    - Troubleshoot the Speech devices SDK
    - Speech devices SDK release notes
    - How to configure a Microphone Array
    - Roobo v1 details
    - Roobo v2 details
- Resources
  - Speech SDK
    - Speech SDK overview
    - Setup Speech SDK
    - Troubleshoot the Speech SDK
    - Speech SDK Release notes
    - Reference
      - C++

- C#
- Java
- JavaScript
- Objective-C
- Python
- Go
- Speech CLI
  - Speech CLI overview
  - Speech CLI quickstart
  - How-to guides
    - Speech CLI configuration options
    - Speech CLI batch operations
- Speech Studio
  - Speech Studio overview
  - Role-based access control in Speech Studio
- Enterprise readiness
  - Region support
  - Compliance and certification
- Support and help options
- Swagger documentation
- Sovereign Clouds
- Pricing calculator
- Azure roadmap
- Privacy and cookies

What is the Speech service?

11/23/2021 • 7 minutes to read • Edit Online

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

> **IMPORTANT**
>
> The Speech service has replaced Bing Speech API and Translator Speech. See the *Migration* section for migration instructions.

The following features are part of the Speech service. Use the links in this table to learn more about common use-cases for each feature, or browse the API reference.

SERVICE	FEATURE	DESCRIPTION	SDK	REST
Speech-to-Text	Real-time Speech-to-text	Speech-to-text transcribes or translates audio streams or local files to text in real time that your applications, tools, or devices can consume or display. Use speech-to-text with Language Understanding (LUIS) to derive user intents from transcribed speech and act on voice commands.	Yes	Yes
	Batch Speech-to-Text	Batch Speech-to-text enables asynchronous speech-to-text transcription of large volumes of speech audio data stored in Azure Blob Storage. In addition to converting speech audio to text, Batch Speech-to-text also allows for diarization and sentiment-analysis.	No	Yes

SERVICE	FEATURE	DESCRIPTION	SDK	REST
	Multi-device Conversation	Connect multiple devices or clients in a conversation to send speech- or text-based messages, with easy support for transcription and translation	Yes	No
	Conversation Transcription	Enables real-time speech recognition, speaker identification, and diarization. It's perfect for transcribing in-person meetings with the ability to distinguish speakers.	Yes	No
	Create Custom Speech Models	If you are using speech-to-text for recognition and transcription in a unique environment, you can create and train custom acoustic, language, and pronunciation models to address ambient noise or industry-specific vocabulary.	No	Yes
	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.	Yes	Yes

SERVICE	FEATURE	DESCRIPTION	SDK	REST
Text-to-Speech	Text-to-speech	Text-to-speech converts input text into human-like synthesized speech using Speech Synthesis Markup Language (SSML). Use neural voices, which are human-like voices powered by deep neural networks. See Language support.	Yes	Yes
	Create Custom Voices	Create custom voice fonts unique to your brand or product.	No	Yes
Speech Translation	Speech translation	Speech translation enables real-time, multi-language translation of speech to your applications, tools, and devices. Use this service for speech-to-speech and speech-to-text translation.	Yes	No
Voice assistants	Voice assistants	Voice assistants using the Speech service empower developers to create natural, human-like conversational interfaces for their applications and experiences. The voice assistant service provides fast, reliable interaction between a device and an assistant implementation that uses the Bot Framework's Direct Line Speech channel or the integrated Custom Commands service for task completion.	Yes	No

SERVICE	FEATURE	DESCRIPTION	SDK	REST
Speaker Recognition	Speaker verification & identification	The 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?".	Yes	Yes

Try the Speech service for free

For the following steps, you need both a Microsoft account and an Azure account. If you do not have a Microsoft account, you can sign up for one free of charge at the Microsoft account portal. Select **Sign in with Microsoft** and then, when asked to sign in, select **Create a Microsoft account**. Follow the steps to create and verify your new Microsoft account.

Once you have a Microsoft account, go to the Azure sign-up page, select **Start free**, and create a new Azure account using a Microsoft account. Here is a video of how to sign up for Azure free account.

> **NOTE**
>
> When you sign up for a free Azure account, it comes with $200 in service credit that you can apply toward a paid Speech service subscription, valid for up to 30 days. Your Azure services are disabled when your credit runs out or expires at the end of the 30 days. To continue using Azure services, you must upgrade your account. For more information, see How to upgrade your Azure free account.
>
> The Speech service has two service tiers: free(f0) and subscription(s0), which have different limitations and benefits. If you use the free, low-volume Speech service tier you can keep this free subscription even after your free trial or service credit expires. For more information, see Cognitive Services pricing - Speech service.

Create the Azure resource

To add a Speech service resource (free or paid tier) to your Azure account:

1. Sign in to the Azure portal using your Microsoft account.
2. Select **Create a resource** at the top left of the portal. If you do not see **Create a resource**, you can always find it by selecting the collapsed menu in the upper left corner of the screen.
3. In the **New** window, type "speech" in the search box and press ENTER.
4. In the search results, select **Speech**.

5. Select **Create**, then:

   - Give a unique name for your new resource. The name helps you distinguish among multiple subscriptions tied to the same service.
   - Choose the Azure subscription that the new resource is associated with to determine how the fees are billed. Here is the introduction for how to create an Azure subscription in the Azure portal.
   - Choose the region where the resource will be used. Azure is a global cloud platform that is generally available in many regions worldwide. To get the best performance, select a region that's closest to you or where your application runs. The Speech service availabilities vary from different regions. Make sure that you create your resource in a supported region. See region support for Speech services.
   - Choose either a free (F0) or paid (S0) pricing tier. For complete information about pricing and usage quotas for each tier, select **View full pricing details** or see speech services pricing. For limits on resources, see Azure Cognitive Services Limits.
   - Create a new resource group for this Speech subscription or assign the subscription to an existing resource group. Resource groups help you keep your various Azure subscriptions organized.
   - Select **Create**. This will take you to the deployment overview and display deployment progress messages.

It takes a few moments to deploy your new Speech resource.

Find keys and location/region

To find the keys and location/region of a completed deployment, follow these steps:

1. Sign in to the Azure portal using your Microsoft account.

2. Select **All resources**, and select the name of your Cognitive Services resource.

3. On the left pane, under **RESOURCE MANAGEMENT**, select **Keys and Endpoint**.

Each subscription has two keys; you can use either key in your application. To copy/paste a key to your code editor or other location, select the copy button next to each key, switch windows to paste the clipboard contents to the desired location.

Additionally, copy the `LOCATION` value, which is your region ID (ex. `westus`, `westeurope`) for SDK calls.

> **IMPORTANT**
> These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely– for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

Complete a quickstart

We offer quickstarts in most popular programming languages, each 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
- Intent recognition quickstart
- Speaker recognition quickstart

After you've had a chance to get started with the Speech service, try our tutorials that show you how to solve various scenarios.

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

Get sample code

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

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

Customize your speech experience

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

Other products offer speech models tuned for specific purposes like healthcare or insurance, but are available to everyone equally. Customization in Azure Speech becomes part of *your unique* competitive advantage that is unavailable to any other user or customer. In other words, your models are private and custom-tuned for your use-case only.

SPEECH SERVICE	PLATFORM	DESCRIPTION

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

Deploy on premises using Docker containers

Use Speech service containers to deploy API features on-premises. These Docker containers enable you to bring the service closer to your data for compliance, security or other operational reasons. The Speech service offers the following containers:

- Standard Speech-to-text
- Custom Speech-to-text
- Standard Text-to-speech
- Neural Text-to-speech
- Custom Text-to-speech (preview)
- Speech Language Identification (preview)

Reference docs

- Speech SDK
- Speech Devices SDK
- REST API: Speech-to-text
- REST API: Text-to-speech
- REST API: Batch transcription and customization

Next steps

Get started with speech-to-text Get started with text-to-speech

Language and voice support for the Speech service

11/23/2021 • 22 minutes to read • Edit Online

Language support varies by Speech service functionality. The following tables summarize language support for Speech-to-text, Text-to-speech, Speech translation and Speaker Recognition service offerings.

Speech-to-text

Both the Microsoft Speech SDK and the REST API support the following languages (locales).

To improve accuracy, customization is offered for a subset of the languages through uploading **Audio + Human-labeled Transcripts** or **Related Text: Sentences**. Support for customization of the acoustic model with **Audio + Human-labeled Transcripts** is limited to the specific base models listed below. Other base models and languages will only use the text of the transcripts to train custom models just like with **Related Text: Sentences**. To learn more about customization, see Get started with Custom Speech.

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Algeria)	`ar-DZ`	Text		
Arabic (Bahrain), modern standard	`ar-BH`	Text		
Arabic (Egypt)	`ar-EG`	Text	Yes	
Arabic (Iraq)	`ar-IQ`	Text		
Arabic (Israel)	`ar-IL`	Text		
Arabic (Jordan)	`ar-JO`	Text		
Arabic (Kuwait)	`ar-KW`	Text		
Arabic (Lebanon)	`ar-LB`	Text		
Arabic (Libya)	`ar-LY`	Text		
Arabic (Morocco)	`ar-MA`	Text		
Arabic (Oman)	`ar-OM`	Text		
Arabic (Qatar)	`ar-QA`	Text		
Arabic (Saudi Arabia)	`ar-SA`	Text		
Arabic (Palestinian Authority)	`ar-PS`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Syria)	`ar-SY`	Text		
Arabic (Tunisia)	`ar-TN`	Text		
Arabic (United Arab Emirates)	`ar-AE`	Text		
Arabic (Yemen)	`ar-YE`	Text		
Bulgarian (Bulgaria)	`bg-BG`	Text		
Catalan (Spain)	`ca-ES`	Text Pronunciation	Yes	
Chinese (Cantonese, Traditional)	`zh-HK`	Audio (20201015) Text	Yes	
Chinese (Mandarin, Simplified)	`zh-CN`	Audio (20200910) Text	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Audio (20190701, 20201015) Text	Yes	
Croatian (Croatia)	`hr-HR`	Text Pronunciation		
Czech (Czech)	`cs-CZ`	Text Pronunciation		
Danish (Denmark)	`da-DK`	Text Pronunciation	Yes	
Dutch (Netherlands)	`nl-NL`	Audio (20201015) Text Pronunciation	Yes	
English (Australia)	`en-AU`	Audio (20201019) Text Pronunciation	Yes	
English (Canada)	`en-CA`	Audio (20201019) Text Pronunciation	Yes	
English (Ghana)	`en-GH`	Text Pronunciation		
English (Hong Kong)	`en-HK`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
English (India)	`en-IN`	Audio (20200923) Text Pronunciation		
English (Ireland)	`en-IE`	Text Pronunciation		
English (Kenya)	`en-KE`	Text Pronunciation		
English (New Zealand)	`en-NZ`	Audio (20201019) Text Pronunciation		
English (Nigeria)	`en-NG`	Text Pronunciation		
English (Philippines)	`en-PH`	Text Pronunciation		
English (Singapore)	`en-SG`	Text Pronunciation		
English (South Africa)	`en-ZA`	Text Pronunciation		
English (Tanzania)	`en-TZ`	Text Pronunciation		
English (United Kingdom)	`en-GB`	Audio (20201019) Text Pronunciation	Yes	Yes
English (United States)	`en-US`	Audio (20201019, 20210223) Text Pronunciation	Yes	Yes
Estonian(Estonia)	`et-EE`	Text Pronunciation		
Filipino (Philippines)	`fil-PH`	Text Pronunciation		
Finnish (Finland)	`fi-FI`	Text Pronunciation	Yes	
French (Canada)	`fr-CA`	Audio (20201015) Text Pronunciation	Yes	
French (France)	`fr-FR`	Audio (20201015) Text Pronunciation	Yes	

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
French (Switzerland)	`fr-CH`	Text Pronunciation		
German (Austria)	`de-AT`	Text Pronunciation		
German (Switzerland)	`de-CH`	Text Pronunciation		
German (Germany)	`de-DE`	Audio (20190701, 20200619, 20201127) Text Pronunciation	Yes	
Greek (Greece)	`el-GR`	Text	Yes	
Gujarati (Indian)	`gu-IN`	Text		
Hebrew (Israel)	`he-IL`	Text		
Hindi (India)	`hi-IN`	Audio (20200701) Text	Yes	
Hungarian (Hungary)	`hu-HU`	Text Pronunciation		
Indonesian (Indonesia)	`id-ID`	Text Pronunciation		
Irish (Ireland)	`ga-IE`	Text Pronunciation		
Italian (Italy)	`it-IT`	Audio (20201016) Text Pronunciation	Yes	
Japanese (Japan)	`ja-JP`	Text	Yes	
Kannada (India)	`kn-IN`	Text		
Korean (Korea)	`ko-KR`	Audio (20201015) Text	Yes	
Latvian (Latvia)	`lv-LV`	Text Pronunciation		
Lithuanian (Lithuania)	`lt-LT`	Text Pronunciation		
Malay (Malaysia)	`ms-MY`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Maltese (Malta)	`mt-MT`	Text		
Marathi (India)	`mr-IN`	Text		
Norwegian (Bokmål, Norway)	`nb-NO`	Text	Yes	
Persian (Iran)	`fa-IR`	Text		
Polish (Poland)	`pl-PL`	Text Pronunciation	Yes	
Portuguese (Brazil)	`pt-BR`	Audio (20190620, 20201015) Text Pronunciation	Yes	
Portuguese (Portugal)	`pt-PT`	Text Pronunciation	Yes	
Romanian (Romania)	`ro-RO`	Text Pronunciation	Yes	
Russian (Russia)	`ru-RU`	Audio (20200907) Text	Yes	
Slovak (Slovakia)	`sk-SK`	Text Pronunciation		
Slovenian (Slovenia)	`sl-SI`	Text Pronunciation		
Spanish (Argentina)	`es-AR`	Text Pronunciation		
Spanish (Bolivia)	`es-BO`	Text Pronunciation		
Spanish (Chile)	`es-CL`	Text Pronunciation		
Spanish (Colombia)	`es-CO`	Text Pronunciation		
Spanish (Costa Rica)	`es-CR`	Text Pronunciation		
Spanish (Cuba)	`es-CU`	Text Pronunciation		
Spanish (Dominican Republic)	`es-DO`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Spanish (Ecuador)	`es-EC`	Text Pronunciation		
Spanish (El Salvador)	`es-SV`	Text Pronunciation		
Spanish (Equatorial Guinea)	`es-GQ`	Text		
Spanish (Guatemala)	`es-GT`	Text Pronunciation		
Spanish (Honduras)	`es-HN`	Text Pronunciation		
Spanish (Mexico)	`es-MX`	Audio (20200907) Text Pronunciation	Yes	
Spanish (Nicaragua)	`es-NI`	Text Pronunciation		
Spanish (Panama)	`es-PA`	Text Pronunciation		
Spanish (Paraguay)	`es-PY`	Text Pronunciation		
Spanish (Peru)	`es-PE`	Text Pronunciation		
Spanish (Puerto Rico)	`es-PR`	Text Pronunciation		
Spanish (Spain)	`es-ES`	Audio (20201015) Text Pronunciation	Yes	
Spanish (Uruguay)	`es-UY`	Text Pronunciation		
Spanish (USA)	`es-US`	Text Pronunciation		
Spanish (Venezuela)	`es-VE`	Text Pronunciation		
Swahili (Kenya)	`sw-KE`	Text		
Swedish (Sweden)	`sv-SE`	Text Pronunciation	Yes	
Tamil (India)	`ta-IN`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Telugu (India)	`te-IN`	Text		
Thai (Thailand)	`th-TH`	Text	Yes	
Turkish (Turkey)	`tr-TR`	Text		
Vietnamese (Vietnam)	`vi-VN`	Text		

Text-to-speech

Both the Microsoft Speech SDK and REST APIs support these voices, each of which supports a specific language and dialect, identified by locale. You can also get a full list of languages and voices supported for each specific region/endpoint through the voices list API.

> **IMPORTANT**
> Pricing varies for standard, custom and neural voices. Please visit the Pricing page for additional information.

Neural voices

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

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

> **NOTE**
> Neural voices are created from samples that use a 24 khz sample rate. All voices can upsample or downsample to other sample rates when synthesizing.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Afrikaans (South Africa)	`af-ZA`	Female	`af-ZA-AdriNeural` New	General
Afrikaans (South Africa)	`af-ZA`	Male	`af-ZA-WillemNeural` New	General
Amharic (Ethiopia)	`am-ET`	Female	`am-ET-MekdesNeural` New	General
Amharic (Ethiopia)	`am-ET`	Male	`am-ET-AmehaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Algeria)	`ar-DZ`	Female	`ar-DZ-AminaNeural` New	General
Arabic (Algeria)	`ar-DZ`	Male	`ar-DZ-IsmaelNeural` New	General
Arabic (Bahrain)	`ar-BH`	Female	`ar-BH-LailaNeural` New	General
Arabic (Bahrain)	`ar-BH`	Male	`ar-BH-AliNeural` New	General
Arabic (Egypt)	`ar-EG`	Female	`ar-EG-SalmaNeural`	General
Arabic (Egypt)	`ar-EG`	Male	`ar-EG-ShakirNeural`	General
Arabic (Iraq)	`ar-IQ`	Female	`ar-IQ-RanaNeural` New	General
Arabic (Iraq)	`ar-IQ`	Male	`ar-IQ-BasselNeural` New	General
Arabic (Jordan)	`ar-JO`	Female	`ar-JO-Sana Neural` New	General
Arabic (Jordan)	`ar-JO`	Male	`ar-JO-Taim Neural` New	General
Arabic (Kuwait)	`ar-KW`	Female	`ar-KW-NouraNeural` New	General
Arabic (Kuwait)	`ar-KW`	Male	`ar-KW-FahedNeural` New	General
Arabic (Libya)	`ar-LY`	Female	`ar-LY-ImanNeural` New	General
Arabic (Libya)	`ar-LY`	Male	`ar-LY-OmarNeural` New	General
Arabic (Morocco)	`ar-MA`	Female	`ar-MA-MounaNeural` New	General
Arabic (Morocco)	`ar-MA`	Male	`ar-MA-JamalNeural` New	General
Arabic (Qatar)	`ar-QA`	Female	`ar-QA-AmalNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Qatar)	`ar-QA`	Male	`ar-QA-MoazNeural` New	General
Arabic (Saudi Arabia)	`ar-SA`	Female	`ar-SA-ZariyahNeural`	General
Arabic (Saudi Arabia)	`ar-SA`	Male	`ar-SA-HamedNeural`	General
Arabic (Syria)	`ar-SY`	Female	`ar-SY-AmanyNeural` New	General
Arabic (Syria)	`ar-SY`	Male	`ar-SY-LaithNeural` New	General
Arabic (Tunisia)	`ar-TN`	Female	`ar-TN-ReemNeural` New	General
Arabic (Tunisia)	`ar-TN`	Male	`ar-TN-HediNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Female	`ar-AE-FatimaNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Male	`ar-AE-HamdanNeural` New	General
Arabic (Yemen)	`ar-YE`	Female	`ar-YE-MaryamNeural` New	General
Arabic (Yemen)	`ar-YE`	Male	`ar-YE-SalehNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Female	`bn-BD-NabanitaNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Male	`bn-BD-PradeepNeural` New	General
Bulgarian (Bulgaria)	`bg-BG`	Female	`bg-BG-KalinaNeural`	General
Bulgarian (Bulgaria)	`bg-BG`	Male	`bg-BG-BorislavNeural`	General
Burmese (Myanmar)	`my-MM`	Female	`my-MM-NilarNeural` New	General
Burmese (Myanmar)	`my-MM`	Male	`my-MM-ThihaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Catalan (Spain)	`ca-ES`	Female	`ca-ES-AlbaNeural`	General
Catalan (Spain)	`ca-ES`	Female	`ca-ES-JoanaNeural`	General
Catalan (Spain)	`ca-ES`	Male	`ca-ES-EnricNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuGaaiNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuMaanNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Male	`zh-HK-WanLungNeural`	General
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaohanNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaomoNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoruiNeural`	Senior voice, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxiaoNeural`	General, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxuanNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyouNeural`	Child voice, optimized for story narrating
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunxiNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyangNeural`	Optimized for news reading, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyeNeural`	Optimized for story narrating
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoChenNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoYuNeural`	General
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-YunJheNeural`	General
Croatian (Croatia)	`hr-HR`	Female	`hr-HR-GabrijelaNeural`	General
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-SreckoNeural`	General
Czech (Czech)	`cs-CZ`	Female	`cs-CZ-VlastaNeural`	General
Czech (Czech)	`cs-CZ`	Male	`cs-CZ-AntoninNeural`	General
Danish (Denmark)	`da-DK`	Female	`da-DK-ChristelNeural`	General
Danish (Denmark)	`da-DK`	Male	`da-DK-JeppeNeural`	General
Dutch (Belgium)	`nl-BE`	Female	`nl-BE-DenaNeural`	General
Dutch (Belgium)	`nl-BE`	Male	`nl-BE-ArnaudNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-ColetteNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-FennaNeural`	General
Dutch (Netherlands)	`nl-NL`	Male	`nl-NL-MaartenNeural`	General
English (Australia)	`en-AU`	Female	`en-AU-NatashaNeural`	General
English (Australia)	`en-AU`	Male	`en-AU-WilliamNeural`	General
English (Canada)	`en-CA`	Female	`en-CA-ClaraNeural`	General
English (Canada)	`en-CA`	Male	`en-CA-LiamNeural`	General
English (Hongkong)	`en-HK`	Female	`en-HK-YanNeural`	General
English (Hongkong)	`en-HK`	Male	`en-HK-SamNeural`	General
English (India)	`en-IN`	Female	`en-IN-NeerjaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (India)	`en-IN`	Male	`en-IN-PrabhatNeural`	General
English (Ireland)	`en-IE`	Female	`en-IE-EmilyNeural`	General
English (Ireland)	`en-IE`	Male	`en-IE-ConnorNeural`	General
English (Kenya)	`en-KE`	Female	`en-KE-AsiliaNeural` New	General
English (Kenya)	`en-KE`	Male	`en-KE-ChilembaNeural` New	General
English (New Zealand)	`en-NZ`	Female	`en-NZ-MollyNeural`	General
English (New Zealand)	`en-NZ`	Male	`en-NZ-MitchellNeural`	General
English (Nigeria)	`en-NG`	Female	`en-NG-EzinneNeural` New	General
English (Nigeria)	`en-NG`	Male	`en-NG-AbeoNeural` New	General
English (Philippines)	`en-PH`	Female	`en-PH-RosaNeural`	General
English (Philippines)	`en-PH`	Male	`en-PH-JamesNeural`	General
English (Singapore)	`en-SG`	Female	`en-SG-LunaNeural`	General
English (Singapore)	`en-SG`	Male	`en-SG-WayneNeural`	General
English (South Africa)	`en-ZA`	Female	`en-ZA-LeahNeural`	General
English (South Africa)	`en-ZA`	Male	`en-ZA-LukeNeural`	General
English (Tanzania)	`en-TZ`	Female	`en-TZ-ImaniNeural` New	General
English (Tanzania)	`en-TZ`	Male	`en-TZ-ElimuNeural` New	General
English (United Kingdom)	`en-GB`	Female	`en-GB-LibbyNeural`	General
English (United Kingdom)	`en-GB`	Female	`en-GB-SoniaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United Kingdom)	`en-GB`	Female	`en-GB-MiaNeural` Retired on 30 October 2021, see below	General
English (United Kingdom)	`en-GB`	Male	`en-GB-RyanNeural`	General
English (United States)	`en-US`	Female	`en-US-AmberNeural`	General
English (United States)	`en-US`	Female	`en-US-AriaNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-AshleyNeural`	General
English (United States)	`en-US`	Female	`en-US-CoraNeural`	General
English (United States)	`en-US`	Female	`en-US-ElizabethNeural`	General
English (United States)	`en-US`	Female	`en-US-JennyNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-MichelleNeural`	General
English (United States)	`en-US`	Female	`en-US-MonicaNeural`	General
English (United States)	`en-US`	Female	`en-US-SaraNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Kid	`en-US-AnaNeural`	General
English (United States)	`en-US`	Male	`en-US-BrandonNeural`	General
English (United States)	`en-US`	Male	`en-US-ChristopherNeural`	General
English (United States)	`en-US`	Male	`en-US-EricNeural`	General
English (United States)	`en-US`	Male	`en-US-GuyNeural`	General, multiple voice styles available using SSML

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Male	`en-US-JacobNeural`	General
Estonian (Estonia)	`et-EE`	Female	`et-EE-AnuNeural`	General
Estonian (Estonia)	`et-EE`	Male	`et-EE-KertNeural`	General
Filipino (Philippines)	`fil-PH`	Female	`fil-PH-BlessicaNeural` New	General
Filipino (Philippines)	`fil-PH`	Male	`fil-PH-AngeloNeural` New	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-NooraNeural`	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-SelmaNeural`	General
Finnish (Finland)	`fi-FI`	Male	`fi-FI-HarriNeural`	General
French (Belgium)	`fr-BE`	Female	`fr-BE-CharlineNeural`	General
French (Belgium)	`fr-BE`	Male	`fr-BE-GerardNeural`	General
French (Canada)	`fr-CA`	Female	`fr-CA-SylvieNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-AntoineNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-JeanNeural`	General
French (France)	`fr-FR`	Female	`fr-FR-DeniseNeural`	General
French (France)	`fr-FR`	Male	`fr-FR-HenriNeural`	General
French (Switzerland)	`fr-CH`	Female	`fr-CH-ArianeNeural`	General
French (Switzerland)	`fr-CH`	Male	`fr-CH-FabriceNeural`	General
Galician (Spain)	`gl-ES`	Female	`gl-ES-SabelaNeural` New	General
Galician (Spain)	`gl-ES`	Male	`gl-ES-RoiNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
German (Austria)	`de-AT`	Female	`de-AT-IngridNeural`	General
German (Austria)	`de-AT`	Male	`de-AT-JonasNeural`	General
German (Germany)	`de-DE`	Female	`de-DE-KatjaNeural`	General
German (Germany)	`de-DE`	Male	`de-DE-ConradNeural`	General
German (Switzerland)	`de-CH`	Female	`de-CH-LeniNeural`	General
German (Switzerland)	`de-CH`	Male	`de-CH-JanNeural`	General
Greek (Greece)	`el-GR`	Female	`el-GR-AthinaNeural`	General
Greek (Greece)	`el-GR`	Male	`el-GR-NestorasNeural`	General
Gujarati (India)	`gu-IN`	Female	`gu-IN-DhwaniNeural`	General
Gujarati (India)	`gu-IN`	Male	`gu-IN-NiranjanNeural`	General
Hebrew (Israel)	`he-IL`	Female	`he-IL-HilaNeural`	General
Hebrew (Israel)	`he-IL`	Male	`he-IL-AvriNeural`	General
Hindi (India)	`hi-IN`	Female	`hi-IN-SwaraNeural`	General
Hindi (India)	`hi-IN`	Male	`hi-IN-MadhurNeural`	General
Hungarian (Hungary)	`hu-HU`	Female	`hu-HU-NoemiNeural`	General
Hungarian (Hungary)	`hu-HU`	Male	`hu-HU-TamasNeural`	General
Indonesian (Indonesia)	`id-ID`	Female	`id-ID-GadisNeural`	General
Indonesian (Indonesia)	`id-ID`	Male	`id-ID-ArdiNeural`	General
Irish (Ireland)	`ga-IE`	Female	`ga-IE-OrlaNeural`	General
Irish (Ireland)	`ga-IE`	Male	`ga-IE-ColmNeural`	General
Italian (Italy)	`it-IT`	Female	`it-IT-ElsaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Italian (Italy)	`it-IT`	Female	`it-IT-IsabellaNeural`	General
Italian (Italy)	`it-IT`	Male	`it-IT-DiegoNeural`	General
Japanese (Japan)	`ja-JP`	Female	`ja-JP-NanamiNeural`	General
Japanese (Japan)	`ja-JP`	Male	`ja-JP-KeitaNeural`	General
Javanese (Indonesia)	`jv-ID`	Female	`jv-ID-SitiNeural` New	General
Javanese (Indonesia)	`jv-ID`	Male	`jv-ID-DimasNeural` New	General
Khmer (Cambodia)	`km-KH`	Female	`km-KH-SreymomNeural` New	General
Khmer (Cambodia)	`km-KH`	Male	`km-KH-PisethNeural` New	General
Korean (Korea)	`ko-KR`	Female	`ko-KR-SunHiNeural`	General
Korean (Korea)	`ko-KR`	Male	`ko-KR-InJoonNeural`	General
Latvian (Latvia)	`lv-LV`	Female	`lv-LV-EveritaNeural`	General
Latvian (Latvia)	`lv-LV`	Male	`lv-LV-NilsNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Female	`lt-LT-OnaNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Male	`lt-LT-LeonasNeural`	General
Malay (Malaysia)	`ms-MY`	Female	`ms-MY-YasminNeural`	General
Malay (Malaysia)	`ms-MY`	Male	`ms-MY-OsmanNeural`	General
Maltese (Malta)	`mt-MT`	Female	`mt-MT-GraceNeural`	General
Maltese (Malta)	`mt-MT`	Male	`mt-MT-JosephNeural`	General
Marathi (India)	`mr-IN`	Female	`mr-IN-AarohiNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Marathi (India)	`mr-IN`	Male	`mr-IN-ManoharNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-IselinNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-PernilleNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Male	`nb-NO-FinnNeural`	General
Persian (Iran)	`fa-IR`	Female	`fa-IR-DilaraNeural` New	General
Persian (Iran)	`fa-IR`	Male	`fa-IR-FaridNeural` New	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-AgnieszkaNeural`	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-ZofiaNeural`	General
Polish (Poland)	`pl-PL`	Male	`pl-PL-MarekNeural`	General
Portuguese (Brazil)	`pt-BR`	Female	`pt-BR-FranciscaNeural`	General, multiple voice styles available using SSML
Portuguese (Brazil)	`pt-BR`	Male	`pt-BR-AntonioNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-FernandaNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-RaquelNeural`	General
Portuguese (Portugal)	`pt-PT`	Male	`pt-PT-DuarteNeural`	General
Romanian (Romania)	`ro-RO`	Female	`ro-RO-AlinaNeural`	General
Romanian (Romania)	`ro-RO`	Male	`ro-RO-EmilNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-DariyaNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-SvetlanaNeural`	General
Russian (Russia)	`ru-RU`	Male	`ru-RU-DmitryNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Slovak (Slovakia)	`sk-SK`	Female	`sk-SK-ViktoriaNeural`	General
Slovak (Slovakia)	`sk-SK`	Male	`sk-SK-LukasNeural`	General
Slovenian (Slovenia)	`sl-SI`	Female	`sl-SI-PetraNeural`	General
Slovenian (Slovenia)	`sl-SI`	Male	`sl-SI-RokNeural`	General
Somali (Somalia)	`so-SO`	Female	`so-SO-UbaxNeural` New	General
Somali (Somalia)	`so-SO`	Male	`so-SO-MuuseNeural` New	General
Spanish (Argentina)	`es-AR`	Female	`es-AR-ElenaNeural`	General
Spanish (Argentina)	`es-AR`	Male	`es-AR-TomasNeural`	General
Spanish (Bolivia)	`es-BO`	Female	`es-BO-SofiaNeural` New	General
Spanish (Bolivia)	`es-BO`	Male	`es-BO-MarceloNeural` New	General
Spanish (Chile)	`es-CL`	Female	`es-CL-CatalinaNeural` New	General
Spanish (Chile)	`es-CL`	Male	`es-CL-LorenzoNeural` New	General
Spanish (Colombia)	`es-CO`	Female	`es-CO-SalomeNeural`	General
Spanish (Colombia)	`es-CO`	Male	`es-CO-GonzaloNeural`	General
Spanish (Costa Rica)	`es-CR`	Female	`es-CR-MariaNeural` New	General
Spanish (Costa Rica)	`es-CR`	Male	`es-CR-JuanNeural` New	General
Spanish (Cuba)	`es-CU`	Female	`es-CU-BelkysNeural` New	General
Spanish (Cuba)	`es-CU`	Male	`es-CU-ManuelNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Dominican Republic)	`es-DO`	Female	`es-DO-RamonaNeural` New	General
Spanish (Dominican Republic)	`es-DO`	Male	`es-DO-EmilioNeural` New	General
Spanish (Ecuador)	`es-EC`	Female	`es-EC-AndreaNeural` New	General
Spanish (Ecuador)	`es-EC`	Male	`es-EC-LuisNeural` New	General
Spanish (El Salvador)	`es-SV`	Female	`es-SV-LorenaNeural` New	General
Spanish (El Salvador)	`es-SV`	Male	`es-SV-RodrigoNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Female	`es-GQ-TeresaNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Male	`es-GQ-JavierNeural` New	General
Spanish (Guatemala)	`es-GT`	Female	`es-GT-MartaNeural` New	General
Spanish (Guatemala)	`es-GT`	Male	`es-GT-AndresNeural` New	General
Spanish (Honduras)	`es-HN`	Female	`es-HN-KarlaNeural` New	General
Spanish (Honduras)	`es-HN`	Male	`es-HN-CarlosNeural` New	General
Spanish (Mexico)	`es-MX`	Female	`es-MX-DaliaNeural`	General
Spanish (Mexico)	`es-MX`	Male	`es-MX-JorgeNeural`	General
Spanish (Nicaragua)	`es-NI`	Female	`es-NI-YolandaNeural` New	General
Spanish (Nicaragua)	`es-NI`	Male	`es-NI-FedericoNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Panama)	`es-PA`	Female	`es-PA-MargaritaNeural` New	General
Spanish (Panama)	`es-PA`	Male	`es-PA-RobertoNeural` New	General
Spanish (Paraguay)	`es-PY`	Female	`es-PY-TaniaNeural` New	General
Spanish (Paraguay)	`es-PY`	Male	`es-PY-MarioNeural` New	General
Spanish (Peru)	`es-PE`	Female	`es-PE-CamilaNeural` New	General
Spanish (Peru)	`es-PE`	Male	`es-PE-AlexNeural` New	General
Spanish (Puerto Rico)	`es-PR`	Female	`es-PR-Karina Neural` New	General
Spanish (Puerto Rico)	`es-PR`	Male	`es-PR-Victor Neural` New	General
Spanish (Spain)	`es-ES`	Female	`es-ES-ElviraNeural`	General
Spanish (Spain)	`es-ES`	Male	`es-ES-AlvaroNeural`	General
Spanish (Uruguay)	`es-UY`	Female	`es-UY-ValentinaNeural` New	General
Spanish (Uruguay)	`es-UY`	Male	`es-UY-MateoNeural` New	General
Spanish (US)	`es-US`	Female	`es-US-PalomaNeural`	General
Spanish (US)	`es-US`	Male	`es-US-AlonsoNeural`	General
Spanish (Venezuela)	`es-VE`	Female	`es-VE-PaolaNeural` New	General
Spanish (Venezuela)	`es-VE`	Male	`es-VE-SebastianNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Sundanese (Indonesia)	`su-ID`	Female	`su-ID-TutiNeural` New	General
Sundanese (Indonesia)	`su-ID`	Male	`su-ID-JajangNeural` New	General
Swahili (Kenya)	`sw-KE`	Female	`sw-KE-ZuriNeural`	General
Swahili (Kenya)	`sw-KE`	Male	`sw-KE-RafikiNeural`	General
Swahili (Tanzania)	`sw-TZ`	Female	`sw-TZ-RehemaNeural` New	General
Swahili (Tanzania)	`sw-TZ`	Male	`sw-TZ-DaudiNeural` New	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-HilleviNeural`	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-SofieNeural`	General
Swedish (Sweden)	`sv-SE`	Male	`sv-SE-MattiasNeural`	General
Tamil (India)	`ta-IN`	Female	`ta-IN-PallaviNeural`	General
Tamil (India)	`ta-IN`	Male	`ta-IN-ValluvarNeural`	General
Tamil (Singapore)	`ta-SG`	Female	`ta-SG-VenbaNeural` New	General
Tamil (Singapore)	`ta-SG`	Male	`ta-SG-AnbuNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Female	`ta-LK-SaranyaNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Male	`ta-LK-KumarNeural` New	General
Telugu (India)	`te-IN`	Female	`te-IN-ShrutiNeural`	General
Telugu (India)	`te-IN`	Male	`te-IN-MohanNeural`	General
Thai (Thailand)	`th-TH`	Female	`th-TH-AcharaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Thai (Thailand)	th-TH	Female	th-TH-PremwadeeNeural	General
Thai (Thailand)	th-TH	Male	th-TH-NiwatNeural	General
Turkish (Turkey)	tr-TR	Female	tr-TR-EmelNeural	General
Turkish (Turkey)	tr-TR	Male	tr-TR-AhmetNeural	General
Ukrainian (Ukraine)	uk-UA	Female	uk-UA-PolinaNeural	General
Ukrainian (Ukraine)	uk-UA	Male	uk-UA-OstapNeural	General
Urdu (India)	ur-IN	Female	ur-IN-GulNeural New	General
Urdu (India)	ur-IN	Male	ur-IN-SalmanNeural New	General
Urdu (Pakistan)	ur-PK	Female	ur-PK-UzmaNeural	General
Urdu (Pakistan)	ur-PK	Male	ur-PK-AsadNeural	General
Uzbek (Uzbekistan)	uz-UZ	Female	uz-UZ-MadinaNeural New	General
Uzbek (Uzbekistan)	uz-UZ	Male	uz-UZ-SardorNeural New	General
Vietnamese (Vietnam)	vi-VN	Female	vi-VN-HoaiMyNeural	General
Vietnamese (Vietnam)	vi-VN	Male	vi-VN-NamMinhNeural	General
Welsh (United Kingdom)	cy-GB	Female	cy-GB-NiaNeural	General
Welsh (United Kingdom)	cy-GB	Male	cy-GB-AledNeural	General
Zulu (South Africa)	zu-ZA	Female	zu-ZA-ThandoNeural New	General
Zulu (South Africa)	zu-ZA	Male	zu-ZA-ThembaNeural New	General

> **IMPORTANT**
>
> The English (United Kingdom) voice `en-GB-MiaNeural` retired on **30 October 2021**. All service requests to `en-GB-MiaNeural` now will be re-directed to `en-GB-SoniaNeural` automatically since **30 October 2021**. If you are using container Neural TTS, please download and deploy the latest version, starting from **30 October 2021**, all requests with previous versions will be rejected.

Neural voices in preview

Below neural voices are in public preview.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Female	`en-US-JennyMultilingualNeural` New	General, multi-lingual capabilities available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaochenNeural` New	Optimized for spontaneous conversation
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyanNeural` New	Optimized for customer service
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoshuangNeural` New	Child voice, optimized for child story and chat; multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoqiuNeural` New	Optimized for narrating

> **IMPORTANT**
>
> Voices in public preview are only available in 3 service regions: East US, West Europe and Southeast Asia.

> **TIP**
>
> `en-US-JennyNeuralMultilingual` supports multiple languages. Check the voices list API for supported languages list.

For more information about regional availability, see regions.

To learn how you can configure and adjust neural voices, such as Speaking Styles, see Speech Synthesis Markup Language.

> **IMPORTANT**
>
> The `en-US-JessaNeural` voice has changed to `en-US-AriaNeural`. If you were using "Jessa" before, convert over to "Aria".

> **TIP**
> You can continue to use the full service name mapping like "Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)" in your speech synthesis requests.

Standard voices

More than 75 standard voices are available in over 45 languages and locales, which allow you to convert text into synthesized speech. For more information about regional availability, see regions.

> **IMPORTANT**
> We are retiring the standard voices on **31st August 2024** and they will no longer be supported after that date.We announced this in emails sent to all existing Speech subscriptions created before **31st August 2021**. During the retiring period (**31st August 2021 - 31st August 2024**), existing standard voice users can continue to use standard voices, but all new users/new speech resources must choose neural voices.

> **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 (Egypt)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-Zhiwei`
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-Matej`
Czech (Czech)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
French (France)	`fr-FR`	Female	`fr-FR-Julie`
French (France)	`fr-FR`	Male	`fr-FR-Paul`
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-HeloisaRUS`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-HeliaRUS`
Romanian (Romania)	`ro-RO`	Male	`ro-RO-Andrei`
Russian (Russia)	`ru-RU`	Female	`ru-RU-EkaterinaRUS`
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".

TIP

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.

Customization

Custom Voice is available in the neural tier (a.k.a, Custom Neural Voice). Based on the Neural TTS technology and the multi-lingual multi-speaker universal model, Custom Neural Voice lets you create synthetic voices that

are rich in speaking styles, or adaptable cross languages. Check below for the languages supported.

> **IMPORTANT**
> The standard tier including the statistical parametric and the concatenative training methods of custom voice is being deprecated and will be retired on 2/29/2024. If you are using non-neural/standard Custom Voice, migrate to Custom Neural Voice immediately to enjoy the better quality and deploy the voices responsibly.

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Arabic (Egypt)	`ar-EG`	Yes	No
Bulgarian (Bulgaria)	`bg-BG`	Yes	No
Chinese (Mandarin, Simplified)	`zh-CN`	Yes	Yes
Chinese (Mandarin, Simplified), English bilingual	`zh-CN` bilingual	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Yes	No
Czech (Czech)	`cs-CZ`	Yes	No
Dutch (Netherlands)	`nl-NL`	Yes	No
English (Australia)	`en-AU`	Yes	Yes
English (Canada)	`en-CA`	Yes	No
English (India)	`en-IN`	Yes	No
English (Ireland)	`en-IE`	Yes	No
English (United Kingdom)	`en-GB`	Yes	Yes
English (United States)	`en-US`	Yes	Yes
French (Canada)	`fr-CA`	Yes	Yes
French (France)	`fr-FR`	Yes	Yes
German (Austria)	`de-AT`	Yes	No
German (Germany)	`de-DE`	Yes	Yes
Hungarian (Hungary)	`hu-HU`	Yes	No
Italian (Italy)	`it-IT`	Yes	Yes

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Japanese (Japan)	`ja-JP`	Yes	Yes
Korean (Korea)	`ko-KR`	Yes	Yes
Norwegian (Bokmål, Norway)	`nb-NO`	Yes	No
Portuguese (Brazil)	`pt-BR`	Yes	Yes
Portuguese (Portugal)	`pt-PT`	Yes	No
Russian (Russia)	`ru-RU`	Yes	Yes
Slovak (Slovakia)	`sk-SK`	Yes	No
Spanish (Mexico)	`es-MX`	Yes	Yes
Spanish (Spain)	`es-ES`	Yes	Yes
Turkish (Turkey)	`tr-TR`	Yes	No
Vietnamese (Vietnam)	`vi-VN`	Yes	No

Select the right locale that matches the training data you have to train a custom voice model. For example, if the recording data you have is spoken in English with a British accent, select `en-GB`.

> **NOTE**
> We do not support bi-lingual model training in Custom Voice, except for the Chinese-English bi-lingual. Select "Chinese-English bilingual" if you want to train a Chinese voice that can speak English as well. Chinese-English bilingual model training using the standard method is available in North Europe and North Central US only. Custom Neural Voice training is available in UK South and East US.

Speech translation

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

Text languages

TEXT LANGUAGE	LANGUAGE CODE
Afrikaans	`af`
Albanian	`sq`
Amharic	`am`

TEXT LANGUAGE	LANGUAGE CODE
Arabic	`ar`
Armenian	`hy`
Assamese	`as`
Azerbaijani	`az`
Bangla	`bn`
Bosnian (Latin)	`bs`
Bulgarian	`bg`
Cantonese (Traditional)	`yue`
Catalan	`ca`
Chinese (Literary)	`lzh`
Chinese Simplified	`zh-Hans`
Chinese Traditional	`zh-Hant`
Croatian	`hr`
Czech	`cs`
Danish	`da`
Dari	`prs`
Dutch	`nl`
English	`en`
Estonian	`et`
Fijian	`fj`
Filipino	`fil`
Finnish	`fi`
French	`fr`
French (Canada)	`fr-ca`

TEXT LANGUAGE	LANGUAGE CODE
German	`de`
Greek	`el`
Gujarati	`gu`
Haitian Creole	`ht`
Hebrew	`he`
Hindi	`hi`
Hmong Daw	`mww`
Hungarian	`hu`
Icelandic	`is`
Indonesian	`id`
Inuktitut	`iu`
Irish	`ga`
Italian	`it`
Japanese	`ja`
Kannada	`kn`
Kazakh	`kk`
Khmer	`km`
Klingon	`tlh-Latn`
Klingon (plqaD)	`tlh-Piqd`
Korean	`ko`
Kurdish (Central)	`ku`
Kurdish (Northern)	`kmr`
Lao	`lo`
Latvian	`lv`

TEXT LANGUAGE	LANGUAGE CODE
Lithuanian	`lt`
Malagasy	`mg`
Malay	`ms`
Malayalam	`ml`
Maltese	`mt`
Maori	`mi`
Marathi	`mr`
Myanmar	`my`
Nepali	`ne`
Norwegian	`nb`
Odia	`or`
Pashto	`ps`
Persian	`fa`
Polish	`pl`
Portuguese (Brazil)	`pt`
Portuguese (Portugal)	`pt-pt`
Punjabi	`pa`
Queretaro Otomi	`otq`
Romanian	`ro`
Russian	`ru`
Samoan	`sm`
Serbian (Cyrillic)	`sr-Cyrl`
Serbian (Latin)	`sr-Latn`
Slovak	`sk`

TEXT LANGUAGE	LANGUAGE CODE
Slovenian	`sl`
Spanish	`es`
Swahili	`sw`
Swedish	`sv`
Tahitian	`ty`
Tamil	`ta`
Telugu	`te`
Thai	`th`
Tigrinya	`ti`
Tongan	`to`
Turkish	`tr`
Ukrainian	`uk`
Urdu	`ur`
Vietnamese	`vi`
Welsh	`cy`
Yucatec Maya	`yua`

Speaker Recognition

Speaker recognition is mostly language agnostic. We built a universal model for text-independent speaker recognition by combining various data sources from multiple languages. We have tuned and evaluated the model on the languages and locales that appear in the following table. See the overview for additional information on Speaker Recognition.

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (US)	`en-US`	yes	yes	yes
Chinese (Mandarin, simplified)	`zh-CN`	n/a	yes	yes
English (Australia)	`en-AU`	n/a	yes	yes

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (Canada)	`en-CA`	n/a	yes	yes
English (India)	`en-IN`	n/a	yes	yes
English (UK)	`en-GB`	n/a	yes	yes
French (Canada)	`fr-CA`	n/a	yes	yes
French (France)	`fr-FR`	n/a	yes	yes
German (Germany)	`de-DE`	n/a	yes	yes
Italian	`it-IT`	n/a	yes	yes
Japanese	`ja-JP`	n/a	yes	yes
Portuguese (Brazil)	`pt-BR`	n/a	yes	yes
Spanish (Mexico)	`es-MX`	n/a	yes	yes
Spanish (Spain)	`es-ES`	n/a	yes	yes

Custom Keyword and Keyword Verification

The following table outlines supported languages for Custom Keyword and Keyword Verification.

LANGUAGE	LOCALE (BCP-47)	CUSTOM KEYWORD	KEYWORD VERIFICATION
Chinese (Mandarin, Simplified)	zh-CN	Yes	Yes
English (United States)	en-US	Yes	Yes
Japanese (Japan)	ja-JP	No	Yes
Portuguese (Brazil)	pt-BR	No	Yes

Next steps

- Create a free Azure account
- See how to recognize speech in C#

Speech service supported regions

11/23/2021 • 5 minutes to read • Edit Online

The Speech service allows your application to convert audio to text, perform speech translation, and covert text to speech. The service is available in multiple regions with unique endpoints for the Speech SDK and REST APIs.

The Speech portal, where you can perform custom configurations to your speech experience for all regions, is available at speech.microsoft.com.

Keep in mind the following points when considering regions:

- If your application uses a Speech SDK, you provide the region identifier, such as `westus`, when creating a speech configuration. Make sure the region matches the region of your subscription.
- If your application uses one of the Speech service's REST APIs, the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

> **NOTE**
>
> Speech Services doesn't store/process customer data outside the region the customer deploys the service instance in.

Speech SDK

In the Speech SDK, the region is specified as a parameter (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

Speech-to-text, text-to-speech, and translation

The Speech service is available in these regions for **speech recognition**, **text-to-speech**, and **translation**:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`

GEOGRAPHY	REGION	REGION IDENTIFIER
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

If you plan to train a custom model with audio data, use one of the regions with dedicated hardware for faster training. You can use the REST API to copy the fully trained model to another region later.

Intent recognition

Available regions for **intent recognition** via the Speech SDK are the following:

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 in these regions:

GLOBAL REGION	REGION	REGION IDENTIFIER
North America	West US	`westus`
North America	West US 2	`westus2`
North America	East US	`eastus`
North America	East US 2	`eastus2`
North America	West Central US	`westcentralus`
North America	South Central US	`southcentralus`
Europe	West Europe	`westeurope`
Europe	North Europe	`northeurope`
Asia	East Asia	`eastasia`
Asia	Southeast Asia	`southeastasia`
India	Central India	`centralindia`

Speaker Recognition

Available regions for **Speaker Recognition** are the following:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Central India	`centralindia`
Australia	Australia East	`australiaeast`
Europe	North Europe	`northeurope`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	West Europe	`westeurope`
Europe	UK South	`uksouth`

REST APIs

The Speech service also exposes REST endpoints for speech-to-text, text-to-speech and speaker recognition requests.

Speech-to-text

For speech-to-text reference documentation, see Speech-to-text REST API.

The endpoint for the REST API has this format:

```
https://<REGION_IDENTIFIER>.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

> **NOTE**
>
> The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is:
>
> ```
> https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
> ```
>
> .

Text-to-speech

For text-to-speech reference documentation, see Text-to-speech REST API.

Neural and standard voices

Use this table to determine **availability of neural and standard voices** by region/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`
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/v`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`

REGION	ENDPOINT
South Central US	`https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v`
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`

> **TIP**
>
> Voices in preview are only available in these 3 regions: East US, West Europe and Southeast Asia.

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	`https://australiaeast.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Brazil South	`https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
Canada Central	`https://canadacentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Central US	`https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East Asia	`https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US	`https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US 2	`https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
France Central	`https://francecentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
India Central	`https://centralindia.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
Japan East	`https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
Japan West	`https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

REGION	ENDPOINT
Korea Central	`https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
North Central US	`https://northcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
North Europe	`https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
South Central US	`https://southcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
Southeast Asia	`https://southeastasia.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
UK South	`https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West Europe	`https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US	`https://westus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US 2	`https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

Custom Neural Voice

The following table details regional support for Custom Neural Voice features.

FEATURE	SUPPORTED REGIONS
Voice model hosting	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Realtime Characters	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Long Audio Characters	East US, West Europe, Southeast Asia, UK South, India Central
Custom Neural Training	East US, Southeast Asia, UK South

Long audio API

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.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`

Speaker Recognition

For speaker recognition reference documentation, see Speaker Recognition REST API. Available regions are the same as Speaker Recognition SDK.

Speech service Quotas and Limits

11/23/2021 • 10 minutes to read • Edit Online

This article contains a quick reference and the **detailed description** of Azure Cognitive Speech service Quotas and Limits for all pricing tiers. It also contains some best practices to avoid request throttling.

Quotas and Limits quick reference

Jump to Text-to-Speech Quotas and limits

Speech-to-Text Quotas and Limits per Speech resource

In the tables below Parameters without "Adjustable" row are **not** adjustable for all price tiers.

Online Transcription

For the usage with Speech SDK and/or 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 V2.0 and v3.0 limit	Batch transcription is not available for F0	300 requests per minute
Max audio input file size	N/A	1 GB
Max input blob size (may contain more than one file, for example, in a zip archive; ensure to note the file size limit above)	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 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 using `text` parameter in Create Model API request	200 KB	500 KB

[1] For **Free (F0)** pricing tier see also 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 tables below Parameters without "Adjustable" row are **not** adjustable for all price tiers.

General

QUOTA	FREE (F0)[3]	STANDARD (S0)
Max number of Transactions per Second (TPS) per Speech resource		
Real-time API. Standard, Neural, Custom, and Custom Neural voices	200[4]	200[4]
Adjustable	No[4]	Yes[4]
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

Long Audio API

QUOTA	FREE (F0)[3]	STANDARD (S0)
Min text length	N/A	400 characters for plain text; 400 billable characters for SSML
Max text length	N/A	10000 paragraphs
Start time	N/A	10 tasks or 10000 characters accumulated

Custom Neural Voice and Custom Voice[6]

QUOTA	FREE (F0)[3]	STANDARD (S0)
Max number of Transactions per Second (TPS) per Speech resource	See above	See above
Max number of data sets per Speech resource	10	500
Max number of simultaneous dataset upload per Speech resource	2	5
Max data file size for data import per dataset	2 GB	2 GB
Upload of long audios or audios without script	No	Yes
Max number of simultaneous model trainings per Speech resource	1 (Custom Voice[6] only)	3
Max number of custom endpoints per Speech resource	1 (Custom Voice[6] only)	50
Concurrent Request limit for Custom Neural voice		
Default value	N/A	10
Adjustable	N/A	Yes[5]
Concurrent Request limit for Custom voice[6]		
Default value	10	10
Adjustable	No[5]	Yes[5]

[3] For **Free (F0)** pricing tier see also monthly allowances at the pricing page.

[4] See additional explanations and best practices.

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

[6] Custom Voice is being deprecated and not available for newly created Speech resources. See additional information.

Detailed description, Quota adjustment, and best practices

Before requesting a quota increase (where applicable), ensure that it is necessary. Speech service is using autoscaling technologies to bring the required computational resources in "on-demand" mode and at the same time to keep the customer costs low by not maintaining an excessive amount of hardware capacity. Every time your application receives a Response Code 429 ("Too many requests") while your workload is within the defined limits (see Quotas and Limits quick reference) the most likely explanation is that the Service is scaling up to your demand and did not reach the required scale yet, thus does not immediately have enough resources to serve the request. This state is usually transient and should not last long.

General best practices to mitigate throttling during autoscaling

To minimize issues related to throttling (Response Code 429), we recommend using the following techniques:

- Implement retry logic in your application
- Avoid sharp changes in the workload. Increase the workload gradually
  Example. Your application is using Text-to-Speech and your current workload is 5 TPS (transactions per second). The next second you increase the load to 20 TPS (that is four times more). The Service immediately starts scaling up to fulfill the new load, but likely it will not be able to do it within a second, so some of the requests will get Response Code 429.
- Test different load increase patterns
  - See Workload pattern example
- Create additional Speech resources in the same or different Regions and distribute the workload among them using "Round Robin" technique. This is especially important for **Text-to-Speech TPS (transactions per second)** parameter, which is set as 200 per Speech Resource and cannot be adjusted

The next sections describe specific cases of adjusting quotas.
Jump to Text-to-speech: increasing concurrent request limit for Custom Neural and Custom Voices

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

By default the number of concurrent requests is limited to 100 per Speech resource (Base model) and to 100 per Custom endpoint (Custom model). For the Standard pricing tier, this amount can be increased. Before submitting the request, ensure you are familiar with the material in this section and aware of these best practices.

> **NOTE**
> If you use custom models, please be aware, that one Speech resource may be associated with many custom endpoints hosting many custom model deployments. Each Custom endpoint has the default number of concurrent request limit (100) set by creation. If you need to adjust it, you need to make the adjustment of each custom endpoint **separately**. Please also note, that the value of the number of concurrent request limit for the base model of a Speech resource has **no** effect to the custom endpoints associated with this resource.

Increasing the Concurrent Request limit does **not** directly affect your costs. Speech service uses "Pay only for what you use" model. The limit defines how high the Service may scale before it starts throttle your requests.

Concurrent Request limits for **Base** and **Custom** models need to be adjusted **separately**.

Existing value of Concurrent Request limit parameter is **not** visible via Azure portal, Command-Line tools, or API requests. To verify the existing value, create an Azure Support Request.

> **NOTE**
> Speech containers do not require increases of Concurrent Request limit, as containers are constrained only by the CPUs of the hardware they are hosted on. However Speech containers have their own capacity limitations that should be taken into account. See the question *"Could you help with capacity planning and cost estimation of on-prem Speech-to-text containers?"* from the Speech containers FAQ.

Have the required information ready:

- For **Base model**:
  - Speech Resource ID
  - Region
- For **Custom model**:
  - Region
  - Custom Endpoint ID
- **How to get information (Base model)**:
  - Go to Azure portal
  - Select the Speech Resource for which you would like to increase the Concurrency Request limit
  - Select *Properties* (*Resource Management* group)
  - Copy and save the values of the following fields:
    - **Resource ID**
    - **Location** (your endpoint Region)
- **How to get information (Custom Model)**:
  - Go to Speech Studio portal
  - Sign in if necessary
  - Go to Custom Speech
  - Select your project
  - Go to *Deployment*
  - Select the required Endpoint
  - Copy and save the values of the following fields:
    - **Service Region** (your endpoint Region)
    - **Endpoint ID**

Create and submit support request

Initiate the increase of Concurrent Request limit for your resource or if necessary check the today's limit by submitting the Support Request:

- Ensure you have the required information
- Go to Azure portal
- Select the Speech Resource for which you would like to increase (or to check) the Concurrency Request limit
- Select *New support request* (*Support + troubleshooting* group)
- A new window will appear with auto-populated information about your Azure Subscription and Azure Resource
- Enter *Summary* (like "Increase STT Concurrency Request limit")
- In *Problem type* select "Quota or Subscription issues"
- In appeared *Problem subtype* select:
  - "Quota or concurrent requests increase" - for an increase request
  - "Quota or usage validation" to check existing limit

- Click *Next: Solutions*
- Proceed further with the request creation
- When in *Details* tab enter in the *Description* field:
  - a note, that the request is about **Speech-to-Text** quota
  - **Base** or **Custom** model
  - Azure resource information you collected before
  - Complete entering the required information and click *Create* button in *Review + create* tab
  - Note the support request number in Azure portal notifications. You will be contacted shortly for further processing

Example of a workload pattern best practice

This example presents the approach we recommend following to mitigate possible request throttling due to Autoscaling being in progress. It is not an "exact recipe", but merely a template we invite to follow and adjust as necessary.

Let us suppose that a Speech 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 1.5-2 minutes. Control the Service responses and implement the logic that falls back (reduces the load) if you get too many Response Codes 429. Then retry in 1-2-4-4 minute pattern. (That is retry the load increase in 1 min, if still does not work, then in 2 min, and so on)

Generally, it is highly recommended to test the workload and the workload patterns before going to production.

Text-to-speech: increasing concurrent request limit for Custom Neural and Custom Voices

By default the number of concurrent requests for Custom Neural Voice and Custom Voice endpoints is limited to 10. For the Standard pricing tier, this amount can be increased. Before submitting the request, ensure you are familiar with the material in this section and aware of these best practices.

Increasing the Concurrent Request limit does **not** directly affect your costs. Speech service uses "Pay only for what you use" model. The limit defines how high the Service may scale before it starts throttle your requests.

Existing value of Concurrent Request limit parameter is **not** visible via Azure portal, Command-Line tools, or API requests. To verify the existing value, create an Azure Support Request.

> **NOTE**
>
> Speech containers do not require increases of Concurrent Request limit, as containers are constrained only by the CPUs of the hardware they are hosted on.

Prepare the required information:

To create an increase request, you will need to provide your Deployment Region and the Custom Endpoint ID. To get it, perform the following actions:

- Go to Speech Studio portal
- Sign in if necessary
- Go to *Custom Voice*
- Select your project
- Go to *Deployment*
- Select the required Endpoint
- Copy and save the values of the following fields:
  - **Service Region** (your endpoint Region)
  - **Endpoint ID**

Create and submit support request

Initiate the increase of Concurrent Request limit for your resource or if necessary check the today's limit by submitting the Support Request:

- Ensure you have the required information
- Go to Azure portal
- Select the Speech Resource for which you would like to increase (or to check) the Concurrency Request limit
- Select *New support request* (*Support + troubleshooting* group)
- A new window will appear with auto-populated information about your Azure Subscription and Azure Resource
- Enter *Summary* (like "Increase TTS Custom Endpoint Concurrency Request limit")
- In *Problem type* select "Quota or Subscription issues"
- In appeared *Problem subtype* select:
  - "Quota or concurrent requests increase" - for an increase request
  - "Quota or usage validation" to check existing limit
- Click *Next: Solutions*
- Proceed further with the request creation
- When in *Details* tab enter in the *Description* field:
  - a note, that the request is about **Text-to-Speech** quota
  - Azure resource information you collected before
  - Complete entering the required information and click *Create* button in *Review + create* tab
  - Note the support request number in Azure portal notifications. You will be contacted shortly for further processing

What is speech-to-text?

11/23/2021 • 2 minutes to read • Edit Online

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

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

This documentation contains the following article types:

- **Quickstarts** are getting-started instructions to guide you through making requests to the service.
- **How-to guides** contain instructions for using the service in more specific or customized ways.
- **Concepts** provide in-depth explanations of the service functionality and features.
- **Tutorials** are longer guides that show you how to use the service as a component in broader business solutions.

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

Get started

See the quickstart to get started with speech-to-text. The service is available via the Speech SDK, the REST API, and the Speech CLI.

Sample code

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

- Speech-to-text samples (SDK)
- Batch transcription samples (REST)
- Pronunciation assessment samples (REST)

Customization

In addition to the standard Speech service model, you can create custom models. Customization helps to overcome speech recognition barriers such as speaking style, vocabulary and background noise, see Custom Speech. Customization options vary by language/locale, see supported languages to verify support.

Batch transcription

Batch transcription is a set of 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. See the how-to for more information on how to use the batch transcription API.

Reference docs

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

Speech SDK reference docs

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

- C# SDK
- C++ SDK
- Java SDK
- Python SDK
- JavaScript SDK
- Objective-C SDK

> **TIP**
>
> The Speech service SDK is actively maintained and updated. To track changes, updates and feature additions refer to the Speech SDK release notes.

Speech Devices SDK reference docs

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

REST API references

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

- REST API: Speech-to-text
- REST API: Pronunciation assessment
- REST API: Text-to-speech
- REST API: Batch transcription and customization

Next steps

- Get a Speech service subscription key for free
- Get the Speech SDK

Get started with speech-to-text

11/23/2021 • 46 minutes to read • Edit Online

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech SDK in your apps and products to perform high-quality speech-to-text conversion.

Skip to samples on GitHub

If you want to skip straight to sample code, see the C# quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

If you just want the package name to get started, run `Install-Package Microsoft.CognitiveServices.Speech` in the NuGet console.

For platform-specific installation instructions, see the following links:

- .NET Framework
- .NET Core
- Unity
- UWP
- Xamarin

Create a speech configuration

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

```
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("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    }
}
```

There are a few other ways that you can initialize a `SpeechConfig`:

- 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 from microphone

To recognize speech using your device microphone, create an `AudioConfig` using `FromDefaultMicrophoneInput()`. Then initialize a `SpeechRecognizer`, passing your `audioConfig` and `speechConfig`.

```
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("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
        await FromMic(speechConfig);
    }
}
```

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

Recognize from file

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

```
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("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
        await FromFile(speechConfig);
    }
}
```

Recognize from in-memory stream

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

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

```
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("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
        await FromStream(speechConfig);
    }
}
```

Using a push stream as input assumes that the audio data is a raw PCM, e.g. skipping 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 the `byte[]` starts at the *start of the audio data*.

Error handling

The previous examples simply get the recognized text from `result.text`, but to handle errors and other responses, you'll 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, inform the user: `ResultReason.NoMatch`
- If an error is encountered, print the error message: `ResultReason.Canceled`

```
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 update the speech key and location/region info?");
        }
        break;
}
```

Continuous recognition

The previous examples use at-start 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, continuous recognition is used 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 a `SpeechRecognizer`:

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

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

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

Next, subscribe to the events sent from the `SpeechRecognizer`.

- `Recognizing`: Signal for events containing intermediate recognition results.
- `Recognized`: Signal for events containing final recognition results (indicating a successful recognition attempt).
- `SessionStopped`: Signal for events indicating the end of a recognition session (operation).
- `Canceled`: Signal for events containing canceled recognition results (indicating a recognition attempt that was canceled as a result or a direct cancellation request or, alternatively, a transport or protocol failure).

```
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 update the speech key and location/region info?");
    }

    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.

```
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();
```

Dictation mode

When using continuous recognition, you can enable dictation processing by using the corresponding "enable dictation" function. This mode will cause the speech config instance to interpret word descriptions of sentence structures such as punctuation. For example, the utterance "Do you live in town question mark" would be interpreted as the text "Do you live in town?".

To enable dictation mode, use the `EnableDictation` method on your `SpeechConfig`.

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

Change source language

A common task for speech recognition is specifying the input (or source) language. Let's take a look at how you

would change the input language to Italian. In your code, find your `SpeechConfig`, then add this line directly below it.

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

The `SpeechRecognitionLanguage` property expects a language-locale format string. You can provide any value in the **Locale** column in the list of supported locales/languages.

Improve recognition accuracy

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

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

Single words or complete phrases can be added to a Phrase List. During recognition, an entry in a phrase list is used to boost recognition of the words and phrases in the list even when the entries appear in the middle of the utterance.

> **IMPORTANT**
>
> The Phrase List feature is available in the following languages: en-US, de-DE, en-AU, en-CA, en-GB, en-IN, es-ES, fr-FR, it-IT, ja-JP, pt-BR, zh-CN
>
> The Phrase List feature should be used with no more than a few hundred phrases. If you have a larger list or for languages that are not currently supported, training a custom model will likely be the better choice to improve accuracy.
>
> The Phrase List feature is not supported with custom endpoints. Do not use it with custom endpoints. Instead, train a custom model that includes the phrases.

To use a phrase list, first create a `PhraseListGrammar` object, then add specific words and phrases with `AddPhrase`.

Any changes to `PhraseListGrammar` take effect on the next recognition or after a reconnection to the Speech service.

```
var phraseList = PhraseListGrammar.FromRecognizer(recognizer);
phraseList.AddPhrase("Supercalifragilisticexpialidocious");
```

If you need to clear your phrase list:

```
phraseList.Clear();
```

Other options to improve recognition accuracy

Phrase lists are only one option to improve recognition accuracy. You can also:

- Improve accuracy with Custom Speech
- Improve accuracy with tenant models

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech SDK in your apps and products to perform high-quality speech-to-text conversion.

Skip to samples on GitHub

If you want to skip straight to sample code, see the C++ quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, use the following instructions:

- Linux
- macOS
- Windows

Create a speech configuration

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

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

auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
```

There are a few other ways that you can initialize a `SpeechConfig`:

- 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.

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

Recognize from microphone

To recognize speech using your device microphone, create an `AudioConfig` using `FromDefaultMicrophoneInput()`. Then initialize a `SpeechRecognizer`, passing your `audioConfig` and `config`.

```
using namespace Microsoft::CognitiveServices::Speech::Audio;

auto audioConfig = AudioConfig::FromDefaultMicrophoneInput();
auto recognizer = SpeechRecognizer::FromConfig(config, audioConfig);

cout << "Speak into your microphone." << std::endl;
auto result = recognizer->RecognizeOnceAsync().get();
cout << "RECOGNIZED: Text=" << result->Text << std::endl;
```

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

Recognize from file

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

```
using namespace Microsoft::CognitiveServices::Speech::Audio;

auto audioInput = AudioConfig::FromWavFileInput("YourAudioFile.wav");
auto recognizer = SpeechRecognizer::FromConfig(config, audioInput);

auto result = recognizer->RecognizeOnceAsync().get();
cout << "RECOGNIZED: Text=" << result->Text << std::endl;
```

Recognize speech

The Recognizer class for the Speech SDK for C++ exposes a few methods that you can use for speech recognition.

At-start recognition

At-start recognition asynchronously 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. Here's an example of asynchronous at-start recognition using `RecognizeOnceAsync`:

```
auto result = recognizer->RecognizeOnceAsync().get();
```

You'll need to write some code to handle the result. This sample evaluates the `result->Reason`:

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

```
switch (result->Reason)
{
    case ResultReason::RecognizedSpeech:
        cout << "We recognized: " << result->Text << std::endl;
        break;
    case ResultReason::NoMatch:
        cout << "NOMATCH: Speech could not be recognized." << std::endl;
        break;
    case ResultReason::Canceled:
        {
            auto cancellation = CancellationDetails::FromResult(result);
            cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

            if (cancellation->Reason == CancellationReason::Error) {
                cout << "CANCELED: ErrorCode= " << (int)cancellation->ErrorCode << std::endl;
                cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
                cout << "CANCELED: Did you update the speech key and location/region info?" << std::endl;
            }
        }
        break;
    default:
        break;
}
```

Continuous recognition

Continuous recognition is a bit more involved than at-start recognition. 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.

Let's start by defining the input and initializing a `SpeechRecognizer`:

```
auto audioInput = AudioConfig::FromWavFileInput("YourAudioFile.wav");
auto recognizer = SpeechRecognizer::FromConfig(config, audioInput);
```

Next, let's create a variable to manage the state of speech recognition. To start, we'll declare a `promise<void>`, since at the start of recognition we can safely assume that it's not finished.

```
promise<void> recognitionEnd;
```

We'll subscribe to the events sent from the `SpeechRecognizer`.

- `Recognizing`: Signal for events containing intermediate recognition results.
- `Recognized`: Signal for events containing final recognition results (indicating a successful recognition attempt).
- `SessionStopped`: Signal for events indicating the end of a recognition session (operation).
- `Canceled`: Signal for events containing canceled recognition results (indicating a recognition attempt that was canceled as a result or a direct cancellation request or, alternatively, a transport or protocol failure).

```
recognizer->Recognizing.Connect([](const SpeechRecognitionEventArgs& e)
    {
        cout << "Recognizing:" << e.Result->Text << std::endl;
    });

recognizer->Recognized.Connect([](const SpeechRecognitionEventArgs& e)
    {
        if (e.Result->Reason == ResultReason::RecognizedSpeech)
        {
            cout << "RECOGNIZED: Text=" << e.Result->Text
                 << " (text could not be translated)" << std::endl;
        }
        else if (e.Result->Reason == ResultReason::NoMatch)
        {
            cout << "NOMATCH: Speech could not be recognized." << std::endl;
        }
    });

recognizer->Canceled.Connect([&recognitionEnd](const SpeechRecognitionCanceledEventArgs& e)
    {
        cout << "CANCELED: Reason=" << (int)e.Reason << std::endl;
        if (e.Reason == CancellationReason::Error)
        {
            cout << "CANCELED: ErrorCode=" << (int)e.ErrorCode << "\n"
                 << "CANCELED: ErrorDetails=" << e.ErrorDetails << "\n"
                 << "CANCELED: Did you update the speech key and location/region info?" << std::endl;

            recognitionEnd.set_value(); // Notify to stop recognition.
        }
    });

recognizer->SessionStopped.Connect([&recognitionEnd](const SessionEventArgs& e)
    {
        cout << "Session stopped.";
        recognitionEnd.set_value(); // Notify to stop recognition.
    });
```

With everything set up, we can call `StopContinuousRecognitionAsync`.

```
// Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
recognizer->StartContinuousRecognitionAsync().get();

// Waits for recognition end.
recognitionEnd.get_future().get();

// Stops recognition.
recognizer->StopContinuousRecognitionAsync().get();
```

Dictation mode

When using continuous recognition, you can enable dictation processing by using the corresponding "enable dictation" function. This mode will cause the speech config instance to interpret word descriptions of sentence structures such as punctuation. For example, the utterance "Do you live in town question mark" would be interpreted as the text "Do you live in town?".

To enable dictation mode, use the `EnableDictation` method on your `SpeechConfig`.

```
config->EnableDictation();
```

Change source language

A common task for speech recognition is specifying the input (or source) language. Let's take a look at how you would change the input language to German. In your code, find your `SpeechConfig`, then add this line directly below it.

```
config->SetSpeechRecognitionLanguage("de-DE");
```

`SetSpeechRecognitionLanguage` is a parameter that takes a string as an argument. You can provide any value in the list of supported locales/languages.

Improve recognition accuracy

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

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

Single words or complete phrases can be added to a Phrase List. During recognition, an entry in a phrase list is used to boost recognition of the words and phrases in the list even when the entries appear in the middle of the utterance.

> **IMPORTANT**
>
> The Phrase List feature is available in the following languages: en-US, de-DE, en-AU, en-CA, en-GB, en-IN, es-ES, fr-FR, it-IT, ja-JP, pt-BR, zh-CN
>
> The Phrase List feature should be used with no more than a few hundred phrases. If you have a larger list or for languages that are not currently supported, training a custom model will likely be the better choice to improve accuracy.
>
> The Phrase List feature is not supported with custom endpoints. Do not use it with custom endpoints. Instead, train a custom model that includes the phrases.

To use a phrase list, first create a `PhraseListGrammar` object, then add specific words and phrases with `AddPhrase`.

Any changes to `PhraseListGrammar` take effect on the next recognition or after a reconnection to the Speech service.

```
auto phraseListGrammar = PhraseListGrammar::FromRecognizer(recognizer);
phraseListGrammar->AddPhrase("Supercalifragilisticexpialidocious");
```

If you need to clear your phrase list:

```
phraseListGrammar->Clear();
```

Other options to improve recognition accuracy

Phrase lists are only one option to improve recognition accuracy. You can also:

- Improve accuracy with Custom Speech
- Improve accuracy with tenant models

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech SDK in your apps and products

to perform high-quality speech-to-text conversion.

Skip to samples on GitHub

If you want to skip straight to sample code, see the Go quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK for Go.

Speech-to-text from microphone

Use the following code sample to run speech recognition from your default device microphone. Replace the variables `subscription` and `region` with your speech key and location/region. See the Find keys and location/region page to find your key-location/region pair. Running the script will start a recognition session on your default microphone and output text.

```
package main

import (
 "bufio"
 "fmt"
 "os"

 "github.com/Microsoft/cognitive-services-speech-sdk-go/audio"
 "github.com/Microsoft/cognitive-services-speech-sdk-go/speech"
)

func sessionStartedHandler(event speech.SessionEventArgs) {
 defer event.Close()
 fmt.Println("Session Started (ID=", event.SessionID, ")")
}

func sessionStoppedHandler(event speech.SessionEventArgs) {
 defer event.Close()
 fmt.Println("Session Stopped (ID=", event.SessionID, ")")
}

func recognizingHandler(event speech.SpeechRecognitionEventArgs) {
 defer event.Close()
 fmt.Println("Recognizing:", event.Result.Text)
}

func recognizedHandler(event speech.SpeechRecognitionEventArgs) {
 defer event.Close()
 fmt.Println("Recognized:", event.Result.Text)
}

func cancelledHandler(event speech.SpeechRecognitionCanceledEventArgs) {
 defer event.Close()
 fmt.Println("Received a cancellation: ", event.ErrorDetails)
}

func main() {
    subscription :=  "<paste-your-speech-key-here>"
    region := "<paste-your-speech-location/region-here>"

 audioConfig, err := audio.NewAudioConfigFromDefaultMicrophoneInput()
```

```
  audioConfig, err := audio.NewAudioConfigFromDefaultMicrophoneInput()
  if err != nil {
   fmt.Println("Got an error: ", err)
   return
  }
  defer audioConfig.Close()
  config, err := speech.NewSpeechConfigFromSubscription(subscription, region)
  if err != nil {
   fmt.Println("Got an error: ", err)
   return
  }
  defer config.Close()
  speechRecognizer, err := speech.NewSpeechRecognizerFromConfig(config, audioConfig)
  if err != nil {
   fmt.Println("Got an error: ", err)
   return
  }
  defer speechRecognizer.Close()
  speechRecognizer.SessionStarted(sessionStartedHandler)
  speechRecognizer.SessionStopped(sessionStoppedHandler)
  speechRecognizer.Recognizing(recognizingHandler)
  speechRecognizer.Recognized(recognizedHandler)
  speechRecognizer.Canceled(cancelledHandler)
  speechRecognizer.StartContinuousRecognitionAsync()
  defer speechRecognizer.StopContinuousRecognitionAsync()
  bufio.NewReader(os.Stdin).ReadBytes('\n')
 }
```

Run the following commands to create a go.mod file that links to components hosted on GitHub.

```
go mod init quickstart
go get github.com/Microsoft/cognitive-services-speech-sdk-go
```

Now build and run the code.

```
go build
go run quickstart
```

See the reference docs for detailed information on the `SpeechConfig` and `SpeechRecognizer` classes.

Speech-to-text from audio file

Use the following sample to run speech recognition from an audio file. Replace the variables `subscription` and `region` with your speech key and location/region. See the Find keys and location/region page to find your key-location/region pair. Additionally, replace the variable `file` with a path to a .wav file. Running the script will recognize speech from the file, and output the text result.

```
package main

import (
 "fmt"
 "time"

 "github.com/Microsoft/cognitive-services-speech-sdk-go/audio"
 "github.com/Microsoft/cognitive-services-speech-sdk-go/speech"
)

func main() {
    subscription :=  "<paste-your-speech-key-here>"
    region := "<paste-your-speech-location/region-here>"
    file := "path/to/file.wav"

 audioConfig, err := audio.NewAudioConfigFromWavFileInput(file)
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer audioConfig.Close()
 config, err := speech.NewSpeechConfigFromSubscription(subscription, region)
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer config.Close()
 speechRecognizer, err := speech.NewSpeechRecognizerFromConfig(config, audioConfig)
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer speechRecognizer.Close()
 speechRecognizer.SessionStarted(func(event speech.SessionEventArgs) {
  defer event.Close()
  fmt.Println("Session Started (ID=", event.SessionID, ")")
 })
 speechRecognizer.SessionStopped(func(event speech.SessionEventArgs) {
  defer event.Close()
  fmt.Println("Session Stopped (ID=", event.SessionID, ")")
 })

 task := speechRecognizer.RecognizeOnceAsync()
 var outcome speech.SpeechRecognitionOutcome
 select {
 case outcome = <-task:
 case <-time.After(5 * time.Second):
  fmt.Println("Timed out")
  return
 }
 defer outcome.Close()
 if outcome.Error != nil {
  fmt.Println("Got an error: ", outcome.Error)
 }
 fmt.Println("Got a recognition!")
 fmt.Println(outcome.Result.Text)
}
```

Run the following commands to create a go.mod file that links to components hosted on GitHub.

```
go mod init quickstart
go get github.com/Microsoft/cognitive-services-speech-sdk-go
```

Now build and run the code.

```
go build
go run quickstart
```

See the reference docs for detailed information on the `SpeechConfig` and `SpeechRecognizer` classes.

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech SDK in your apps and products to perform high-quality speech-to-text conversion.

Skip to samples on GitHub

If you want to skip straight to sample code, see the Java quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, use the following instructions:

- Java Runtime
- Android

Create a speech configuration

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

```
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

public class Program {
    public static void main(String[] args) throws InterruptedException, ExecutionException {
        SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-subscription-key>", "<paste-
your-region>");
    }
}
```

There are a few other ways that you can initialize a `SpeechConfig`:

- 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.

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

Recognize from microphone

To recognize speech using your device microphone, create an `AudioConfig` using `fromDefaultMicrophoneInput()`. Then initialize a `SpeechRecognizer`, passing your `audioConfig` and `config`.

```
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

public class Program {
    public static void main(String[] args) throws InterruptedException, ExecutionException {
        SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-subscription-key>", "<paste-
your-region>");
        fromMic(speechConfig);
    }

    public static void fromMic(SpeechConfig speechConfig) throws InterruptedException, ExecutionException {
        AudioConfig audioConfig = AudioConfig.fromDefaultMicrophoneInput();
        SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, audioConfig);

        System.out.println("Speak into your microphone.");
        Future<SpeechRecognitionResult> task = recognizer.recognizeOnceAsync();
        SpeechRecognitionResult result = task.get();
        System.out.println("RECOGNIZED: Text=" + result.getText());
    }
}
```

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

Recognize from file

If you want to recognize speech from an audio file instead of using a microphone, you still need to create an `AudioConfig`. However, when you create the `AudioConfig`, instead of calling `fromDefaultMicrophoneInput()`, call `fromWavFileInput()` and pass the file path.

```
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

public class Program {
    public static void main(String[] args) throws InterruptedException, ExecutionException {
        SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-subscription-key>", "<paste-
your-region>");
        fromFile(speechConfig);
    }

    public static void fromFile(SpeechConfig speechConfig) throws InterruptedException, ExecutionException {
        AudioConfig audioConfig = AudioConfig.fromWavFileInput("YourAudioFile.wav");
        SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, audioConfig);

        Future<SpeechRecognitionResult> task = recognizer.recognizeOnceAsync();
        SpeechRecognitionResult result = task.get();
        System.out.println("RECOGNIZED: Text=" + result.getText());
    }
}
```

Error handling

The previous examples simply get the recognized text using `result.getText()`, but to handle errors and other responses, you'll need to write some code to handle the result. The following example evaluates `result.getReason()` and:

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

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

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

Continuous recognition

The previous examples use at-start 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, continuous recognition is used 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.

Let's start by defining the input and initializing a `SpeechRecognizer`:

```
AudioConfig audioConfig = AudioConfig.fromWavFileInput("YourAudioFile.wav");
SpeechRecognizer recognizer = new SpeechRecognizer(config, audioConfig);
```

Next, let's create a variable to manage the state of speech recognition. To start, we'll declare a `Semaphore` at the class scope.

```
private static Semaphore stopTranslationWithFileSemaphore;
```

We'll subscribe to the events sent from the `SpeechRecognizer`.

- `recognizing`: Signal for events containing intermediate recognition results.
- `recognized`: Signal for events containing final recognition results (indicating a successful recognition attempt).
- `sessionStopped`: Signal for events indicating the end of a recognition session (operation).
- `canceled`: Signal for events containing canceled recognition results (indicating a recognition attempt that was canceled as a result or a direct cancellation request or, alternatively, a transport or protocol failure).

```
// First initialize the semaphore.
stopTranslationWithFileSemaphore = new Semaphore(0);

recognizer.recognizing.addEventListener((s, e) -> {
    System.out.println("RECOGNIZING: Text=" + e.getResult().getText());
});

recognizer.recognized.addEventListener((s, e) -> {
    if (e.getResult().getReason() == ResultReason.RecognizedSpeech) {
        System.out.println("RECOGNIZED: Text=" + e.getResult().getText());
    }
    else if (e.getResult().getReason() == ResultReason.NoMatch) {
        System.out.println("NOMATCH: Speech could not be recognized.");
    }
});

recognizer.canceled.addEventListener((s, e) -> {
    System.out.println("CANCELED: Reason=" + e.getReason());

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

    stopTranslationWithFileSemaphore.release();
});

recognizer.sessionStopped.addEventListener((s, e) -> {
    System.out.println("\n    Session stopped event.");
    stopTranslationWithFileSemaphore.release();
});
```

With everything set up, we can call `startContinuousRecognitionAsync`.

```
// Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
recognizer.startContinuousRecognitionAsync().get();

// Waits for completion.
stopTranslationWithFileSemaphore.acquire();

// Stops recognition.
recognizer.stopContinuousRecognitionAsync().get();
```

Dictation mode

When using continuous recognition, you can enable dictation processing by using the corresponding "enable dictation" function. This mode will cause the speech config instance to interpret word descriptions of sentence structures such as punctuation. For example, the utterance "Do you live in town question mark" would be interpreted as the text "Do you live in town?".

To enable dictation mode, use the `enableDictation` method on your `SpeechConfig`.

```
config.enableDictation();
```

Change source language

A common task for speech recognition is specifying the input (or source) language. Let's take a look at how you would change the input language to French. In your code, find your `SpeechConfig`, then add this line directly below it.

```
config.setSpeechRecognitionLanguage("fr-FR");
```

`setSpeechRecognitionLanguage` is a parameter that takes a string as an argument. You can provide any value in the list of supported locales/languages.

Improve recognition accuracy

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

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

Single words or complete phrases can be added to a Phrase List. During recognition, an entry in a phrase list is used to boost recognition of the words and phrases in the list even when the entries appear in the middle of the utterance.

> **IMPORTANT**
>
> The Phrase List feature is available in the following languages: en-US, de-DE, en-AU, en-CA, en-GB, en-IN, es-ES, fr-FR, it-IT, ja-JP, pt-BR, zh-CN
>
> The Phrase List feature should be used with no more than a few hundred phrases. If you have a larger list or for languages that are not currently supported, training a custom model will likely be the better choice to improve accuracy.
>
> The Phrase List feature is not supported with custom endpoints. Do not use it with custom endpoints. Instead, train a custom model that includes the phrases.

To use a phrase list, first create a `PhraseListGrammar` object, then add specific words and phrases with `AddPhrase`.

Any changes to `PhraseListGrammar` take effect on the next recognition or after a reconnection to the Speech service.

```
PhraseListGrammar phraseList = PhraseListGrammar.fromRecognizer(recognizer);
phraseList.addPhrase("Supercalifragilisticexpialidocious");
```

If you need to clear your phrase list:

```
phraseList.clear();
```

Other options to improve recognition accuracy

Phrase lists are only one option to improve recognition accuracy. You can also:

- Improve accuracy with Custom Speech
- Improve accuracy with tenant models

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech SDK in your apps and products to perform high-quality speech-to-text conversion.

Skip to samples on GitHub

If you want to skip straight to sample code, see the JavaScript quickstart samples on GitHub.

Alternatively, see the React sample to learn how to use the Speech SDK in a browser-based environment.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you need to install the Speech SDK for Node.js. If you just want the package name to install, run `npm install microsoft-cognitiveservices-speech-sdk`. For guided installation instructions, see the get started article.

Use the following `require` statement to import the SDK.

```
const sdk = require("microsoft-cognitiveservices-speech-sdk");
```

For more information on `require`, see the require documentation.

Create a speech configuration

To call the Speech service using the Speech SDK, you need to create a `SpeechConfig`. This class includes information about your subscription, like your key and associated location/region, endpoint, host, or authorization token. Create a `SpeechConfig` using your key and location/region. See the Find keys and location/region page to find your key-location/region pair.

```
const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-location/region-here>");
```

There are a few other ways that you can initialize a `SpeechConfig`:

- 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 location/region.

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

Recognize from microphone (Browser only)

Recognizing speech from a microphone is **not supported in Node.js**, and is only supported in a browser-based JavaScript environment. See the React sample on GitHub to see the speech-to-text from microphone implementation.

> **NOTE**
> If you want to use a *specific* audio input device, you need to specify the device ID in the `AudioConfig`. Learn how to get the device ID for your audio input device.

Recognize from file

To recognize speech from an audio file, create an `AudioConfig` using `fromWavFileInput()` which accepts a `Buffer` object. Then initialize a `SpeechRecognizer`, passing your `audioConfig` and `speechConfig`.

```
const fs = require('fs');
const sdk = require("microsoft-cognitiveservices-speech-sdk");
const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-location/region-here>");

function fromFile() {
    let audioConfig = sdk.AudioConfig.fromWavFileInput(fs.readFileSync("YourAudioFile.wav"));
    let recognizer = new sdk.SpeechRecognizer(speechConfig, audioConfig);

    recognizer.recognizeOnceAsync(result => {
        console.log(`RECOGNIZED: Text=${result.text}`);
        recognizer.close();
    });
}
fromFile();
```

Recognize from in-memory stream

For many use-cases, it is likely your audio data will be coming from blob storage, or otherwise already be in-memory as an `ArrayBuffer` or similar raw data structure. The following code:

- Creates a push stream using `createPushStream()`.
- Reads a `.wav` file using `fs.createReadStream` for demonstration purposes, but if you already have audio data in an `ArrayBuffer`, you can skip directly to writing the content to the input stream.

- Creates an audio config using the push stream.

```
const fs = require('fs');
const sdk = require("microsoft-cognitiveservices-speech-sdk");
const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");

function fromStream() {
    let pushStream = sdk.AudioInputStream.createPushStream();

    fs.createReadStream("YourAudioFile.wav").on('data', function(arrayBuffer) {
        pushStream.write(arrayBuffer.slice());
    }).on('end', function() {
        pushStream.close();
    });

    let audioConfig = sdk.AudioConfig.fromStreamInput(pushStream);
    let recognizer = new sdk.SpeechRecognizer(speechConfig, audioConfig);
    recognizer.recognizeOnceAsync(result => {
        console.log(`RECOGNIZED: Text=${result.text}`);
        recognizer.close();
    });
}
fromStream();
```

Using a push stream as input assumes that the audio data is a raw PCM, e.g. skipping 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 the `fs` starts at the *start of the audio data*.

Error handling

The previous examples simply get the recognized text from `result.text`, but to handle errors and other responses, you'll 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, inform the user: `ResultReason.NoMatch`
- If an error is encountered, print the error message: `ResultReason.Canceled`

```
switch (result.reason) {
    case sdk.ResultReason.RecognizedSpeech:
        console.log(`RECOGNIZED: Text=${result.text}`);
        break;
    case sdk.ResultReason.NoMatch:
        console.log("NOMATCH: Speech could not be recognized.");
        break;
    case sdk.ResultReason.Canceled:
        const cancellation = CancellationDetails.fromResult(result);
        console.log(`CANCELED: Reason=${cancellation.reason}`);

        if (cancellation.reason == sdk.CancellationReason.Error) {
            console.log(`CANCELED: ErrorCode=${cancellation.ErrorCode}`);
            console.log(`CANCELED: ErrorDetails=${cancellation.errorDetails}`);
            console.log("CANCELED: Did you update the key and location/region info?");
        }
        break;
    }
```

Continuous recognition

The previous examples use at-start 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, continuous recognition is used 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 a `SpeechRecognizer`:

```
const recognizer = new sdk.SpeechRecognizer(speechConfig, audioConfig);
```

Next, subscribe to the events sent from the `SpeechRecognizer`.

- `recognizing`: Signal for events containing intermediate recognition results.
- `recognized`: Signal for events containing final recognition results (indicating a successful recognition attempt).
- `sessionStopped`: Signal for events indicating the end of a recognition session (operation).
- `canceled`: Signal for events containing canceled recognition results (indicating a recognition attempt that was canceled as a result or a direct cancellation request or, alternatively, a transport or protocol failure).

```
recognizer.recognizing = (s, e) => {
    console.log(`RECOGNIZING: Text=${e.result.text}`);
};

recognizer.recognized = (s, e) => {
    if (e.result.reason == sdk.ResultReason.RecognizedSpeech) {
        console.log(`RECOGNIZED: Text=${e.result.text}`);
    }
    else if (e.result.reason == sdk.ResultReason.NoMatch) {
        console.log("NOMATCH: Speech could not be recognized.");
    }
};

recognizer.canceled = (s, e) => {
    console.log(`CANCELED: Reason=${e.reason}`);

    if (e.reason == sdk.CancellationReason.Error) {
        console.log(`"CANCELED: ErrorCode=${e.errorCode}`);
        console.log(`"CANCELED: ErrorDetails=${e.errorDetails}`);
        console.log("CANCELED: Did you update the key and location/region info?");
    }

    recognizer.stopContinuousRecognitionAsync();
};

recognizer.sessionStopped = (s, e) => {
    console.log("\n    Session stopped event.");
    recognizer.stopContinuousRecognitionAsync();
};
```

With everything set up, call `startContinuousRecognitionAsync` to start recognizing.

```
recognizer.startContinuousRecognitionAsync();

// make the following call at some point to stop recognition.
// recognizer.stopContinuousRecognitionAsync();
```

Dictation mode

When using continuous recognition, you can enable dictation processing by using the corresponding "enable dictation" function. This mode will cause the speech config instance to interpret word descriptions of sentence structures such as punctuation. For example, the utterance "Do you live in town question mark" would be interpreted as the text "Do you live in town?".

To enable dictation mode, use the `enableDictation` method on your `SpeechConfig`.

```
speechConfig.enableDictation();
```

Change source language

A common task for speech recognition is specifying the input (or source) language. Let's take a look at how you would change the input language to Italian. In your code, find your `SpeechConfig`, then add this line directly below it.

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

The `speechRecognitionLanguage` property expects a language-locale format string. You can provide any value in the **Locale** column in the list of supported locales/languages.

Improve recognition accuracy

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

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

Single words or complete phrases can be added to a Phrase List. During recognition, an entry in a phrase list is used to boost recognition of the words and phrases in the list even when the entries appear in the middle of the utterance.

> **IMPORTANT**
>
> The Phrase List feature is available in the following languages: en-US, de-DE, en-AU, en-CA, en-GB, en-IN, es-ES, fr-FR, it-IT, ja-JP, pt-BR, zh-CN
>
> The Phrase List feature should be used with no more than a few hundred phrases. If you have a larger list or for languages that are not currently supported, training a custom model will likely be the better choice to improve accuracy.
>
> The Phrase List feature is not supported with custom endpoints. Do not use it with custom endpoints. Instead, train a custom model that includes the phrases.

To use a phrase list, first create a `PhraseListGrammar` object, then add specific words and phrases with `addPhrase`.

Any changes to `PhraseListGrammar` take effect on the next recognition or after a reconnection to the Speech service.

```
const phraseList = sdk.PhraseListGrammar.fromRecognizer(recognizer);
phraseList.addPhrase("Supercalifragilisticexpialidocious");
```

If you need to clear your phrase list:

```
phraseList.clear();
```

Other options to improve recognition accuracy

Phrase lists are only one option to improve recognition accuracy. You can also:

- Improve accuracy with Custom Speech
- Improve accuracy with tenant models

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this sample, you learn how to use the Speech SDK in your apps and products to perform high-quality speech-to-text conversion.

React sample on GitHub

Go to the React sample on GitHub to learn how to use the Speech SDK in a browser-based JavaScript environment. This sample shows design pattern examples for authentication token exchange and management, and capturing audio from a microphone or file for speech-to-text conversions.

Additionally, the design patterns used in the Node.js quickstart can also be used in a browser environment.

You can transcribe speech into text using the Speech SDK for Swift and Objective-C.

Prerequisites

The following samples assume that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install Speech SDK and samples

The Cognitive Services Speech SDK contains samples written in in Swift and Objective-C for iOS and Mac. Click a link to see installation instructions for each sample:

- Recognize speech from a microphone in Objective-C on macOS
- Recognize speech in Swift on macOS
- Recognize speech in Objective-C on iOS
- Recognize speech in Swift on iOS
- Additional samples for Objective-C on iOS

We also provide an online Speech SDK for Objective-C Reference.

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech SDK in your apps and products to perform high-quality speech-to-text conversion.

Skip to samples on GitHub

If you want to skip straight to sample code, see the Python quickstart samples on GitHub.

Prerequisites

This article assumes:

- You have an Azure account and Speech service subscription. If you don't have and account and subscription -

- Try the Speech service for free.

Install and import the Speech SDK

Before you can do anything, you'll need to install the Speech SDK.

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

If you're on macOS and run into install issues, you may need to run this command first.

```
python3 -m pip install --upgrade pip
```

After the Speech SDK is installed, import it into your Python project.

```
import azure.cognitiveservices.speech as speechsdk
```

Create a speech configuration

To call the Speech service using the Speech SDK, you need to create a `SpeechConfig`. This class includes information about your subscription, like your key and associated location/region, endpoint, host, or authorization token. Create a `SpeechConfig` using your key and location/region. See the Find keys and location/region page to find your key-location/region pair.

```
speech_config = speechsdk.SpeechConfig(subscription="<paste-your-speech-key-here>", region="<paste-your-speech-location/region-here>")
```

There are a few other ways that you can initialize a `SpeechConfig`:

- 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.

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

Recognize from microphone

To recognize speech using your device microphone, simply create a `SpeechRecognizer` without passing an `AudioConfig`, and pass your `speech_config`.

```
import azure.cognitiveservices.speech as speechsdk

def from_mic():
    speech_config = speechsdk.SpeechConfig(subscription="<paste-your-speech-key-here>", region="<paste-your-speech-location/region-here>")
    speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config)

    print("Speak into your microphone.")
    result = speech_recognizer.recognize_once_async().get()
    print(result.text)

from_mic()
```

If you want to use a *specific* audio input device, you need to specify the device ID in an `AudioConfig`, and pass it to the `SpeechRecognizer` constructor's `audio_config` param. Learn how to get the device ID for your audio input device.

Recognize from file

If you want to recognize speech from an audio file instead of using a microphone, create an `AudioConfig` and use the `filename` parameter.

```
import azure.cognitiveservices.speech as speechsdk

def from_file():
    speech_config = speechsdk.SpeechConfig(subscription="<paste-your-speech-key-here>", region="<paste-your-speech-location/region-here>")
    audio_input = speechsdk.AudioConfig(filename="your_file_name.wav")
    speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config, audio_config=audio_input)

    result = speech_recognizer.recognize_once_async().get()
    print(result.text)

from_file()
```

Error handling

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

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

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

Continuous recognition

The previous examples use at-start 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, continuous recognition is used when you want to **control** when to stop recognizing. It requires you to connect to the `EventSignal` to get the recognition results, and to stop recognition, you must call stop_continuous_recognition() or stop_continuous_recognition(). Here's an example of how continuous recognition is performed on an audio input file.

Let's start by defining the input and initializing a `SpeechRecognizer`:

```
audio_config = speechsdk.audio.AudioConfig(filename=weatherfilename)
speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config, audio_config=audio_config)
```

Next, let's create a variable to manage the state of speech recognition. To start, we'll set this to `False`, since at the start of recognition we can safely assume that it's not finished.

```
done = False
```

Now, we're going to create a callback to stop continuous recognition when an `evt` is received. There's a few things to keep in mind.

- When an `evt` is received, the `evt` message is printed.
- After an `evt` is received, stop_continuous_recognition() is called to stop recognition.
- The recognition state is changed to `True`.

```
def stop_cb(evt):
    print('CLOSING on {}'.format(evt))
    speech_recognizer.stop_continuous_recognition()
    nonlocal done
    done = True
```

This code sample shows how to connect callbacks to events sent from the `SpeechRecognizer`.

- `recognizing`: Signal for events containing intermediate recognition results.
- `recognized`: Signal for events containing final recognition results (indicating a successful recognition attempt).
- `session_started`: Signal for events indicating the start of a recognition session (operation).
- `session_stopped`: Signal for events indicating the end of a recognition session (operation).
- `canceled`: Signal for events containing canceled recognition results (indicating a recognition attempt that was canceled as a result or a direct cancellation request or, alternatively, a transport or protocol failure).

```
speech_recognizer.recognizing.connect(lambda evt: print('RECOGNIZING: {}'.format(evt)))
speech_recognizer.recognized.connect(lambda evt: print('RECOGNIZED: {}'.format(evt)))
speech_recognizer.session_started.connect(lambda evt: print('SESSION STARTED: {}'.format(evt)))
speech_recognizer.session_stopped.connect(lambda evt: print('SESSION STOPPED {}'.format(evt)))
speech_recognizer.canceled.connect(lambda evt: print('CANCELED {}'.format(evt)))

speech_recognizer.session_stopped.connect(stop_cb)
speech_recognizer.canceled.connect(stop_cb)
```

With everything set up, we can call start_continuous_recognition().

```
speech_recognizer.start_continuous_recognition()
while not done:
    time.sleep(.5)
```

Dictation mode

When using continuous recognition, you can enable dictation processing by using the corresponding "enable dictation" function. This mode will cause the speech config instance to interpret word descriptions of sentence structures such as punctuation. For example, the utterance "Do you live in town question mark" would be interpreted as the text "Do you live in town?".

To enable dictation mode, use the `enable_dictation()` method on your `SpeechConfig`.

```
SpeechConfig.enable_dictation()
```

Change source language

A common task for speech recognition is specifying the input (or source) language. Let's take a look at how you would change the input language to German. In your code, find your SpeechConfig, then add this line directly below it.

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

`speech_recognition_language` is a parameter that takes a string as an argument. You can provide any value in the list of supported locales/languages.

Improve recognition accuracy

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

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

Single words or complete phrases can be added to a Phrase List. During recognition, an entry in a phrase list is used to boost recognition of the words and phrases in the list even when the entries appear in the middle of the utterance.

> **IMPORTANT**
>
> The Phrase List feature is available in the following languages: en-US, de-DE, en-AU, en-CA, en-GB, en-IN, es-ES, fr-FR, it-IT, ja-JP, pt-BR, zh-CN
>
> The Phrase List feature should be used with no more than a few hundred phrases. If you have a larger list or for languages that are not currently supported, training a custom model will likely be the better choice to improve accuracy.
>
> The Phrase List feature is not supported with custom endpoints. Do not use it with custom endpoints. Instead, train a custom model that includes the phrases.

To use a phrase list, first create a `PhraseListGrammar` object, then add specific words and phrases with `addPhrase`.

Any changes to `PhraseListGrammar` take effect on the next recognition or after a reconnection to the Speech

service.

```
phrase_list_grammar = speechsdk.PhraseListGrammar.from_recognizer(reco)
phrase_list_grammar.addPhrase("Supercalifragilisticexpialidocious")
```

If you need to clear your phrase list:

```
phrase_list_grammar.clear()
```

Other options to improve recognition accuracy

Phrase lists are only one option to improve recognition accuracy. You can also:

- Improve accuracy with Custom Speech
- Improve accuracy with tenant models

In this quickstart, you learn how to convert speech to text using the Speech service and cURL.

For a high-level look at Speech-to-Text concepts, see the overview article.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Convert speech to text

At a command prompt, run the following command. You will need to insert the following values into the command.

- Your Speech service subscription key.
- Your Speech service region.
- The input audio file path. You can generate audio files using text-to-speech.

```
curl --location --request POST
'https://INSERT_REGION_HERE.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?
language=en-US' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_AUDIO_FILE_PATH_HERE'
```

You should receive a response like the following one.

```
{
    "RecognitionStatus": "Success",
    "DisplayText": "My voice is my passport, verify me.",
    "Offset": 6600000,
    "Duration": 32100000
}
```

For more information see the speech-to-text REST API reference.

One of the core features of the Speech service is the ability to recognize and transcribe human speech (often referred to as speech-to-text). In this quickstart, you learn how to use the Speech CLI in your apps and products to perform high-quality speech-to-text conversion.

Download and install

- Windows Install
- Linux Install
- Docker Install (Windows, Linux, macOS)

Follow these steps to install the Speech CLI on Windows:

1. On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
2. Install .NET Core 3.1 SDK.
3. Install the Speech CLI using NuGet by entering this command:

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

Type `spx` to see help for the Speech CLI.

> **NOTE**
> As an alternative to NuGet, you can download and extract the Speech CLI for Windows as a zip file.

Font limitations

On Windows, the Speech CLI can only show fonts available to the command prompt on the local computer. Windows Terminal supports all fonts produced interactively by the Speech CLI.

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

Speech-to-text from microphone

Plug in and turn on your PC microphone, and turn off any apps that might also use the microphone. Some computers have a built-in microphone, while others require configuration of a Bluetooth device.

Now you're ready to run the Speech CLI to recognize speech from your microphone. From the command line, change to the directory that contains the Speech CLI binary file, and run the following command.

```
spx recognize --microphone
```

> **NOTE**
> The Speech CLI defaults to English. You can choose a different language from the Speech-to-text table. For example, add `--source de-DE` to recognize German speech.

Speak into the microphone, and you see transcription of your words into text in real-time. The Speech CLI will stop after a period of silence, or when you press ctrl-C.

Speech-to-text from audio file

The Speech CLI can recognize speech in many file formats and natural languages. In this example, you can use any WAV file (16kHz or 8kHz, 16-bit, and mono PCM) that contains English speech. Or if you want a quick sample, download the whatstheweatherlike.wav file and copy it to the same directory as the Speech CLI binary file.

Now you're ready to run the Speech CLI to recognize speech found in the audio file by running the following command.

```
spx recognize --file whatstheweatherlike.wav
```

> **NOTE**
>
> The Speech CLI defaults to English. You can choose a different language from the Speech-to-text table. For example, add `--source de-DE` to recognize German speech.

The Speech CLI will show a text transcription of the speech on the screen.

Next steps

- Use codec compressed audio formats
- See the quickstart samples on GitHub

Specify source language for speech to text

11/23/2021 • 4 minutes to read • Edit Online

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

How to specify source language in C#

In the following example, the source language is provided explicitly as a parameter using `SpeechRecognizer` construct.

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

In the following example, the source language is provided using `SourceLanguageConfig`. Then, the `sourceLanguageConfig` is passed as a parameter to `SpeechRecognizer` construct.

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

In the following example, the source language and custom endpoint are provided using `SourceLanguageConfig`. Then, the `sourceLanguageConfig` is passed as a parameter to `SpeechRecognizer` construct.

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

NOTE

`SpeechRecognitionLanguage` and `EndpointId` set methods are deprecated from the `SpeechConfig` class in C#. The use of these methods are discouraged, and shouldn't be used when constructing a `SpeechRecognizer`.

How to specify source language in C++

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

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

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

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

In the following example, the source language and custom endpoint are provided using `SourceLanguageConfig`.

The `sourceLanguageConfig` is passed as a parameter to `FromConfig` when creating the `recognizer`.

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

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

How to specify source language in Java

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

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

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

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

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

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

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

How to specify source language in Python

In the following example, the source language is provided explicitly as a parameter using `SpeechRecognizer` construct.

```
speech_recognizer = speechsdk.SpeechRecognizer(
        speech_config=speech_config, language="de-DE", audio_config=audio_config)
```

In the following example, the source language is provided using `SourceLanguageConfig`. Then, the `SourceLanguageConfig` is passed as a parameter to `SpeechRecognizer` construct.

```
source_language_config = speechsdk.languageconfig.SourceLanguageConfig("de-DE")
speech_recognizer = speechsdk.SpeechRecognizer(
        speech_config=speech_config, source_language_config=source_language_config,
audio_config=audio_config)
```

In the following example, the source language and custom endpoint are provided using `SourceLanguageConfig`. Then, the `SourceLanguageConfig` is passed as a parameter to `SpeechRecognizer` construct.

```
source_language_config = speechsdk.languageconfig.SourceLanguageConfig("de-DE", "The Endpoint ID for your
custom model.")
speech_recognizer = speechsdk.SpeechRecognizer(
        speech_config=speech_config, source_language_config=source_language_config,
audio_config=audio_config)
```

NOTE

`speech_recognition_language` and `endpoint_id` properties are deprecated from the `SpeechConfig` class in Python. The use of these properties is discouraged, and they shouldn't be used when constructing a `SpeechRecognizer`.

How to specify source language in Javascript

The first step is to create a `SpeechConfig`:

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

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

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

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

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

How to specify source language in Objective-C

In the following example, the source language is provided explicitly as a parameter using `SPXSpeechRecognizer` construct.

```
SPXSpeechRecognizer* speechRecognizer = \
    [[SPXSpeechRecognizer alloc] initWithSpeechConfiguration:speechConfig language:@"de-DE"
audioConfiguration:audioConfig];
```

In the following example, the source language is provided using `SPXSourceLanguageConfiguration`. Then, the `SPXSourceLanguageConfiguration` is passed as a parameter to `SPXSpeechRecognizer` construct.

```
SPXSourceLanguageConfiguration* sourceLanguageConfig = [[SPXSourceLanguageConfiguration alloc]init:@"de-
DE"];
SPXSpeechRecognizer* speechRecognizer = [[SPXSpeechRecognizer alloc]
initWithSpeechConfiguration:speechConfig

sourceLanguageConfiguration:sourceLanguageConfig

audioConfiguration:audioConfig];
```

In the following example, the source language and custom endpoint are provided using `SPXSourceLanguageConfiguration`. Then, the `SPXSourceLanguageConfiguration` is passed as a parameter to `SPXSpeechRecognizer` construct.

```
SPXSourceLanguageConfiguration* sourceLanguageConfig = \
        [[SPXSourceLanguageConfiguration alloc]initWithLanguage:@"de-DE"
                                                    endpointId:@"The Endpoint ID for your custom model."];
SPXSpeechRecognizer* speechRecognizer = [[SPXSpeechRecognizer alloc]
initWithSpeechConfiguration:speechConfig

sourceLanguageConfiguration:sourceLanguageConfig

audioConfiguration:audioConfig];
```

NOTE

`speechRecognitionLanguage` and `endpointId` properties are deprecated from the `SPXSpeechConfiguration` class in Objective-C. The use of these properties is discouraged, and they shouldn't be used when constructing a `SPXSpeechRecognizer`.

See also

- For a list of supported languages and locales for speech to text, see Language support.

Next steps

- Speech SDK reference documentation

Use codec compressed audio input

11/23/2021 • 9 minutes to read • Edit Online

The Speech SDK and Speech CLI can accept compressed audio formats using GStreamer. GStreamer decompresses the audio before it is sent over the wire to the Speech service as raw PCM.

PLATFORM	LANGUAGES	SUPPORTED GSTREAMER VERSION
Linux	C++, C#, Java, Python, Go	Supported Linux distributions and target architectures
Windows (excluding UWP)	C++, C#, Java, Python	1.18.3
Android	Java	1.18.3

Installing GStreamer on Linux

For more information, see Linux installation instructions.

```
sudo apt install libgstreamer1.0-0 \
gstreamer1.0-plugins-base \
gstreamer1.0-plugins-good \
gstreamer1.0-plugins-bad \
gstreamer1.0-plugins-ugly
```

Installing GStreamer on Windows

For more information, see Windows installation instructions.

- Create a folder c:\gstreamer
- Download installer
- Copy the installer to c:\gstreamer
- Open PowerShell as an administrator.
- Run the following command in the PowerShell:

```
cd c:\gstreamer
msiexec /passive INSTALLLEVEL=1000 INSTALLDIR=C:\gstreamer /i gstreamer-1.0-msvc-x86_64-1.18.3.msi
```

- Add the system variables GST_PLUGIN_PATH with value C:\gstreamer\1.0\msvc_x86_64\lib\gstreamer-1.0
- Add the system variables GSTREAMER_ROOT_X86_64 with value C:\gstreamer\1.0\msvc_x86_64
- Add another entry in the path variable as C:\gstreamer\1.0\msvc_x86_64\bin
- Reboot the machine

Using GStreamer in Android

Look at the Java tab above for the details about building libgstreamer_android.so

For more information see Android installation instructions.

Speech SDK version required for compressed audio input

- Speech SDK version 1.10.0 or later is required for RHEL 8 and CentOS 8
- Speech SDK version 1.11.0 or later is required for Windows.
- Speech SDK version 1.16.0 or later for the latest GStreamer on Windows and Android.

The default audio streaming format is WAV (16 kHz or 8 kHz, 16-bit, and mono PCM). Outside of WAV / PCM, the compressed input formats listed below are also supported using GStreamer.

- MP3
- OPUS/OGG
- FLAC
- ALAW in wav container
- MULAW in wav container
- ANY (For the scenario where the media format is not known)

GStreamer required to handle compressed audio

Handling compressed audio is implemented using GStreamer. For licensing reasons GStreamer binaries are not compiled and linked with the Speech SDK. Developers need to install several dependencies and plugins, see Installing on Windows or Installing on Linux. GStreamer binaries need to be in the system path, so that the Speech SDK can load the binaries during runtime. For example, on Windows, if the Speech SDK is able to find `libgstreamer-1.0-0.dll` or `gstreamer-1.0-0.dll` (for latest GStreamer) during runtime, it means the GStreamer binaries are in the system path.

Handling compressed audio is implemented using GStreamer. For licensing reasons GStreamer binaries are not compiled and linked with the Speech SDK. Developers need to install several dependencies and plugins.

> **NOTE**
>
> For mandatory general setup on Linux, see system requirements and setup instructions.

- Ubuntu/Debian
- RHEL/CentOS

```
sudo apt install libgstreamer1.0-0 \
gstreamer1.0-plugins-base \
gstreamer1.0-plugins-good \
gstreamer1.0-plugins-bad \
gstreamer1.0-plugins-ugly
```

Handling compressed audio is implemented using GStreamer. For licensing reasons GStreamer binaries are not compiled and linked with the Speech SDK. Instead, you'll need to use the prebuilt binaries for Android. To download the prebuilt libraries, see installing for Android development.

`libgstreamer_android.so` is required. Make sure that all the GStreamer plugins (from Android.mk file below) are linked in `libgstreamer_android.so`. When using the latest speech SDK (1.16 and above) with GStreamer version 1.18.3, `libc++_shared.so` is also required to be present from android ndk.

```
GSTREAMER_PLUGINS := coreelements app audioconvert mpg123 \
    audioresample audioparsers ogg opusparse \
    opus wavparse alaw mulaw flac
```

An example `Android.mk` and `Application.mk` file are provided below. Follow these steps to create the `gstreamer` shared object: `libgstreamer_android.so`.

```
# Android.mk
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := dummy
LOCAL_SHARED_LIBRARIES := gstreamer_android
include $(BUILD_SHARED_LIBRARY)

ifndef GSTREAMER_ROOT_ANDROID
$(error GSTREAMER_ROOT_ANDROID is not defined!)
endif

ifndef APP_BUILD_SCRIPT
$(error APP_BUILD_SCRIPT is not defined!)
endif

ifndef TARGET_ARCH_ABI
$(error TARGET_ARCH_ABI is not defined!)
endif

ifeq ($(TARGET_ARCH_ABI),armeabi)
GSTREAMER_ROOT        := $(GSTREAMER_ROOT_ANDROID)/arm
else ifeq ($(TARGET_ARCH_ABI),armeabi-v7a)
GSTREAMER_ROOT        := $(GSTREAMER_ROOT_ANDROID)/armv7
else ifeq ($(TARGET_ARCH_ABI),arm64-v8a)
GSTREAMER_ROOT        := $(GSTREAMER_ROOT_ANDROID)/arm64
else ifeq ($(TARGET_ARCH_ABI),x86)
GSTREAMER_ROOT        := $(GSTREAMER_ROOT_ANDROID)/x86
else ifeq ($(TARGET_ARCH_ABI),x86_64)
GSTREAMER_ROOT        := $(GSTREAMER_ROOT_ANDROID)/x86_64
else
$(error Target arch ABI not supported: $(TARGET_ARCH_ABI))
endif

GSTREAMER_NDK_BUILD_PATH  := $(GSTREAMER_ROOT)/share/gst-android/ndk-build/
include $(GSTREAMER_NDK_BUILD_PATH)/plugins.mk
GSTREAMER_PLUGINS         :=  $(GSTREAMER_PLUGINS_CORE) \
                              $(GSTREAMER_PLUGINS_CODECS) \
                              $(GSTREAMER_PLUGINS_PLAYBACK) \
                              $(GSTREAMER_PLUGINS_CODECS_GPL) \
                              $(GSTREAMER_PLUGINS_CODECS_RESTRICTED)
GSTREAMER_EXTRA_LIBS      := -liconv -lgstbase-1.0 -lGLESv2 -lEGL
include $(GSTREAMER_NDK_BUILD_PATH)/gstreamer-1.0.mk
```

```
# Application.mk
APP_STL = c++_shared
APP_PLATFORM = android-21
APP_BUILD_SCRIPT = Android.mk
```

You can build `libgstreamer_android.so` using the following command on Ubuntu 18.04 or 20.04. The following command lines have only been tested for GStreamer Android version 1.14.4 with Android NDK b16b.

```
# Assuming wget and unzip already installed on the system
mkdir buildLibGstreamer
cd buildLibGstreamer
wget https://dl.google.com/android/repository/android-ndk-r16b-linux-x86_64.zip
unzip -q -o android-ndk-r16b-linux-x86_64.zip
export PATH=$PATH:$(pwd)/android-ndk-r16b
export NDK_PROJECT_PATH=$(pwd)/android-ndk-r16b
wget https://gstreamer.freedesktop.org/data/pkg/android/1.14.4/gstreamer-1.0-android-universal-
1.14.4.tar.bz2
mkdir gstreamer_android
tar -xjf gstreamer-1.0-android-universal-1.14.4.tar.bz2 -C $(pwd)/gstreamer_android/
export GSTREAMER_ROOT_ANDROID=$(pwd)/gstreamer_android

mkdir gstreamer
# Copy the Application.mk and Android.mk from the documentation above and put it inside $(pwd)/gstreamer

# Enable only one of the following at one time to create the shared object for the targeted ABI
echo "building for armeabi-v7a. libgstreamer_android.so will be placed in $(pwd)/armeabi-v7a"
ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=armeabi-v7a NDK_LIBS_OUT=$(pwd)

#echo "building for arm64-v8a. libgstreamer_android.so will be placed in $(pwd)/arm64-v8a"
#ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=arm64-v8a NDK_LIBS_OUT=$(pwd)

#echo "building for x86_64. libgstreamer_android.so will be placed in $(pwd)/x86_64"
#ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=x86_64 NDK_LIBS_OUT=$(pwd)

#echo "building for x86. libgstreamer_android.so will be placed in $(pwd)/x86"
#ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=x86 NDK_LIBS_OUT=$(pwd)
```

Once the shared object (`libgstreamer_android.so`) is built application developer needs to place the shared object in the Android app, so that it can be loaded by speech SDK.

Handling compressed audio is implemented using GStreamer. For licensing reasons GStreamer binaries are not compiled and linked with the Speech SDK. Developers need to install several dependencies and plugins, see Installing on Windows or Installing on Linux. GStreamer binaries need to be in the system path, so that the Speech SDK can load the binaries during runtime. For example, on Windows, if the Speech SDK is able to find `libgstreamer-1.0-0.dll` during runtime, it means the GStreamer binaries are in the system path.

Speech SDK can use GStreamer to handle compressed audio. However, for licensing reasons GStreamer binaries are not compiled and linked with the Speech SDK. Developers need to install several dependencies and plugins, see Installing on Linux. Go language is only supported in Speech SDK on the Linux platform. See Speech SDK for Go to get started with Microsoft Speech SDK in Go.

Example code using codec compressed audio input

To configure Speech SDK to accept compressed audio input, create `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying 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 may look like this:

```
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 audioFormat =
    AudioStreamFormat.GetCompressedFormat(
        AudioStreamContainerFormat.OGG_OPUS);
var audioConfig =
    AudioConfig.FromStreamInput(
        pullStream,
        audioFormat);

using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
var result = await recognizer.RecognizeOnceAsync();

var text = result.Text;
```

To configure Speech SDK to accept compressed audio input, create `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying the compression format of the stream. Find related sample code in Speech SDK samples.

Let's assume that you have an input stream class called `pushStream` and are using OPUS/OGG. Your code may look like this:

```
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Audio;

// ... omitted for brevity

 auto config =
    SpeechConfig::FromSubscription(
        "YourSubscriptionKey",
        "YourServiceRegion"
    );

auto audioFormat =
    AudioStreamFormat::GetCompressedFormat(
        AudioStreamContainerFormat::OGG_OPUS
    );
auto audioConfig =
    AudioConfig::FromStreamInput(
        pushStream,
        audioFormat
    );

auto recognizer = SpeechRecognizer::FromConfig(config, audioConfig);
auto result = recognizer->RecognizeOnceAsync().get();

auto text = result->Text;
```

To configure Speech SDK to accept compressed audio input, create a `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying the compression format of the stream. Find related sample code in Speech SDK samples.

Let's assume that you have an input stream class called `pullStream` and are using OPUS/OGG. Your code may

look like this:

```
import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import com.microsoft.cognitiveservices.speech.audio.AudioInputStream;
import com.microsoft.cognitiveservices.speech.audio.AudioStreamFormat;
import com.microsoft.cognitiveservices.speech.audio.PullAudioInputStream;
import com.microsoft.cognitiveservices.speech.audio.AudioStreamContainerFormat;

// ... omitted for brevity

SpeechConfig speechConfig =
    SpeechConfig.fromSubscription(
        "YourSubscriptionKey",
        "YourServiceRegion");

// Create an audio config specifying the compressed
// audio format and the instance of your input stream class.
AudioStreamFormat audioFormat =
    AudioStreamFormat.getCompressedFormat(
        AudioStreamContainerFormat.OGG_OPUS);
AudioConfig audioConfig =
    AudioConfig.fromStreamInput(
        pullStream,
        audioFormat);

SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, audioConfig);
SpeechRecognitionResult result = recognizer.recognizeOnceAsync().get();

String text = result.getText();
```

To configure Speech SDK to accept compressed audio input, create `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying the compression format of the stream.

Let's assume that your use case is to use `PullStream` for an `MP3` file. Your code may look like this:

```
import azure.cognitiveservices.speech as speechsdk

class BinaryFileReaderCallback(speechsdk.audio.PullAudioInputStreamCallback):
    def __init__(self, filename: str):
        super().__init__()
        self._file_h = open(filename, "rb")

    def read(self, buffer: memoryview) -> int:
        print('trying to read {} frames'.format(buffer.nbytes))
        try:
            size = buffer.nbytes
            frames = self._file_h.read(size)

            buffer[:len(frames)] = frames
            print('read {} frames'.format(len(frames)))

            return len(frames)
        except Exception as ex:
            print('Exception in `read`: {}'.format(ex))
            raise

    def close(self) -> None:
        print('closing file')
        try:
            self._file_h.close()
        except Exception as ex:
            print('Exception in `close`: {}'.format(ex))
            raise
```

```
def compressed_stream_helper(compressed_format,
        mp3_file_path,
        default_speech_auth):
    callback = BinaryFileReaderCallback(mp3_file_path)
    stream = speechsdk.audio.PullAudioInputStream(stream_format=compressed_format,
pull_stream_callback=callback)

    speech_config = speechsdk.SpeechConfig(**default_speech_auth)
    audio_config = speechsdk.audio.AudioConfig(stream=stream)

    speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config, audio_config=audio_config)

    done = False

    def stop_cb(evt):
        """callback that signals to stop continuous recognition upon receiving an event `evt`"""
        print('CLOSING on {}'.format(evt))
        nonlocal done
        done = True

    # Connect callbacks to the events fired by the speech recognizer
    speech_recognizer.recognizing.connect(lambda evt: print('RECOGNIZING: {}'.format(evt)))
    speech_recognizer.recognized.connect(lambda evt: print('RECOGNIZED: {}'.format(evt)))
    speech_recognizer.session_started.connect(lambda evt: print('SESSION STARTED: {}'.format(evt)))
    speech_recognizer.session_stopped.connect(lambda evt: print('SESSION STOPPED {}'.format(evt)))
    speech_recognizer.canceled.connect(lambda evt: print('CANCELED {}'.format(evt)))
    # stop continuous recognition on either session stopped or canceled events
    speech_recognizer.session_stopped.connect(stop_cb)
    speech_recognizer.canceled.connect(stop_cb)

    # Start continuous speech recognition
    speech_recognizer.start_continuous_recognition()
    while not done:
        time.sleep(.5)

    speech_recognizer.stop_continuous_recognition()
    # </SpeechContinuousRecognitionWithFile>

def pull_audio_input_stream_compressed_mp3(mp3_file_path: str,
        default_speech_auth):
    # Create a compressed format
    compressed_format =
speechsdk.audio.AudioStreamFormat(compressed_stream_format=speechsdk.AudioStreamContainerFormat.MP3)
    compressed_stream_helper(compressed_format, mp3_file_path, default_speech_auth)
```

To configure the Speech SDK to accept compressed audio input, create a `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying the compression format of the stream.

In the following example let's assume that your use case is to use `PushStream` for a compressed file.

```
package recognizer

import (
  "fmt"
  "time"
    "strings"

  "github.com/Microsoft/cognitive-services-speech-sdk-go/audio"
  "github.com/Microsoft/cognitive-services-speech-sdk-go/speech"
  "github.com/Microsoft/cognitive-services-speech-sdk-go/samples/helpers"
)
```

```
func RecognizeOnceFromCompressedFile(subscription string, region string, file string) {
  var containerFormat audio.AudioStreamContainerFormat
  if strings.Contains(file, ".mulaw") {
    containerFormat = audio.MULAW
  } else if strings.Contains(file, ".alaw") {
    containerFormat = audio.ALAW
  } else if strings.Contains(file, ".mp3") {
    containerFormat = audio.MP3
  } else if strings.Contains(file, ".flac") {
    containerFormat = audio.FLAC
  } else if strings.Contains(file, ".opus") {
    containerFormat = audio.OGGOPUS
  } else {
    containerFormat = audio.ANY
  }
  format, err := audio.GetCompressedFormat(containerFormat)
  if err != nil {
    fmt.Println("Got an error: ", err)
    return
  }
  defer format.Close()
  stream, err := audio.CreatePushAudioInputStreamFromFormat(format)
  if err != nil {
    fmt.Println("Got an error: ", err)
    return
  }
  defer stream.Close()
  audioConfig, err := audio.NewAudioConfigFromStreamInput(stream)
  if err != nil {
    fmt.Println("Got an error: ", err)
    return
  }
  defer audioConfig.Close()
  config, err := speech.NewSpeechConfigFromSubscription(subscription, region)
  if err != nil {
    fmt.Println("Got an error: ", err)
    return
  }
  defer config.Close()
  speechRecognizer, err := speech.NewSpeechRecognizerFromConfig(config, audioConfig)
  if err != nil {
    fmt.Println("Got an error: ", err)
    return
  }
  defer speechRecognizer.Close()
  speechRecognizer.SessionStarted(func(event speech.SessionEventArgs) {
    defer event.Close()
    fmt.Println("Session Started (ID=", event.SessionID, ")")
  })
  speechRecognizer.SessionStopped(func(event speech.SessionEventArgs) {
    defer event.Close()
    fmt.Println("Session Stopped (ID=", event.SessionID, ")")
  })
  helpers.PumpFileIntoStream(file, stream)
  task := speechRecognizer.RecognizeOnceAsync()
  var outcome speech.SpeechRecognitionOutcome
  select {
  case outcome = <-task:
  case <-time.After(40 * time.Second):
    fmt.Println("Timed out")
    return
  }
  defer outcome.Close()
  if outcome.Error != nil {
    fmt.Println("Got an error: ", outcome.Error)
  }
  fmt.Println("Got a recognition!")
  fmt.Println(outcome.Result.Text)
}
```

Next steps

Learn how to recognize speech

How to: Select an audio input device with the Speech SDK

11/23/2021 • 5 minutes to read • Edit Online

Version 1.3.0 of the Speech SDK introduces an API to select the audio input. This article describes how to obtain the IDs of the audio devices connected to a system. These can then be used in the Speech SDK by configuring the audio device through the `AudioConfig` object:

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

```
audio_config = AudioConfig(device_name="<device id>");
```

```
audioConfig = AudioConfiguration.FromMicrophoneInput("<device id>");
```

```
audioConfig = AudioConfiguration.fromMicrophoneInput("<device id>");
```

```
audioConfig = AudioConfiguration.fromMicrophoneInput("<device id>");
```

NOTE

Microphone usage is not available for JavaScript running in Node.js

Audio device IDs on Windows for Desktop applications

Audio device endpoint ID strings can be retrieved from the `IMMDevice` object in Windows for Desktop applications.

The following code sample illustrates how to use it to enumerate audio devices in C++:

```
#include <cstdio>
#include <mmdeviceapi.h>

#include <Functiondiscoverykeys_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 pointer to endpoint number i.
        hr = pCollection->Item(i, &pEndpoint);
        EXIT_ON_ERROR(hr);

        // Get the endpoint ID string.
        hr = pEndpoint->GetId(&pwszID);
        EXIT_ON_ERROR(hr);

        hr = pEndpoint->OpenPropertyStore(
            STGM_READ, &pProps);
        EXIT_ON_ERROR(hr);

        // Initialize container for property value.
        PropVariantInit(&varName);

        // Get the endpoint's friendly-name property.
        hr = pProps->GetValue(PKEY_Device_FriendlyName, &varName);
        EXIT_ON_ERROR(hr);

        // Print endpoint friendly name and endpoint ID.
        printf("Endpoint %d: \"%S\" (%S)\n", i, varName.pwszVal, pwszID);

        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#, the NAudio library can be used to access the CoreAudio API and enumerate devices as follows:

```
using System;

using NAudio.CoreAudioApi;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            var enumerator = new MMDeviceEnumerator();
            foreach (var endpoint in
                     enumerator.EnumerateAudioEndPoints(DataFlow.Capture, DeviceState.Active))
            {
                Console.WriteLine("{0} ({1})", endpoint.FriendlyName, endpoint.ID);
            }
        }
    }
}
```

A sample device ID is `{0.0.1.00000000}.{5f23ab69-6181-4f4a-81a4-45414013aac8}`.

Audio device IDs on UWP

On the Universal Windows Platform (UWP), audio input devices can be obtained using the `Id()` property of the corresponding `DeviceInformation` object.

The following code samples show how to do this in C++ and C#:

```
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Devices.Enumeration.h>

using namespace winrt::Windows::Devices::Enumeration;

void enumerateDeviceIds()
{
    auto promise = DeviceInformation::FindAllAsync(DeviceClass::AudioCapture);

    promise.Completed(
        [](winrt::Windows::Foundation::IAsyncOperation<DeviceInformationCollection> const& sender,
           winrt::Windows::Foundation::AsyncStatus /* asyncStatus */ ) {
        auto info = sender.GetResults();
        auto num_devices = info.Size();

        for (const auto &device : info)
        {
            std::wstringstream ss{};
            ss << "looking at device (of " << num_devices << "): " << device.Id().c_str() << "\n";
            OutputDebugString(ss.str().c_str());
        }
    });
}
```

```
using Windows.Devices.Enumeration;
using System.Linq;

namespace helloworld {
    private async void EnumerateDevices()
    {
        var devices = await DeviceInformation.FindAllAsync(DeviceClass.AudioCapture);

        foreach (var device in devices)
        {
            Console.WriteLine($"{device.Name}, {device.Id}\n");
        }
    }
}
```

A sample device ID is

```
\\\\?\\SWD#MMDEVAPI#{0.0.1.00000000}.{5f23ab69-6181-4f4a-81a4-45414013aac8}#{2eef81be-33fa-4800-9670-1cd474972c3f}
```

.

Audio device IDs on Linux

The device IDs are selected using standard ALSA device IDs.

The IDs of the inputs attached to the system are contained in the output of the command `arecord -L`. Alternatively, they can be obtained using the ALSA C library.

Sample IDs are `hw:1,0` and `hw:CARD=CC,DEV=0`.

Audio device IDs on macOS

The following function implemented in Objective-C creates a list of the names and IDs of the audio devices attached to a Mac.

The `deviceUID` string is used to identify a device in the Speech SDK for macOS.

```
#import <Foundation/Foundation.h>
```

```
#import <CoreAudio/CoreAudio.h>

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(kCFAllocatorDefault, deviceCount,
&kCFTypeArrayCallBacks);
    if (NULL == inputDeviceArray) {
        fputs("CFArrayCreateMutable failed", stderr);
        free(audioDevices);
        audioDevices = NULL;
        return NULL;
    }

    // Iterate through all the devices and determine which are input-capable
    propertyAddress.mScope = kAudioDevicePropertyScopeInput;
    for (UInt32 i = 0; i < deviceCount; ++i) {
        // Query device UID
        CFStringRef deviceUID = NULL;
        dataSize = sizeof(deviceUID);
        propertyAddress.mSelector = kAudioDevicePropertyDeviceUID;
        status = AudioObjectGetPropertyData(audioDevices[i], &propertyAddress, 0, NULL, &dataSize,
&deviceUID);
        if (kAudioHardwareNoError != status) {
            fprintf(stderr, "AudioObjectGetPropertyData (kAudioDevicePropertyDeviceUID) failed: %i\n",
status);
            continue;
        }

        // Query device name
        CFStringRef deviceName = NULL;
        dataSize = sizeof(deviceName);
        propertyAddress.mSelector = kAudioDevicePropertyDeviceNameCFString;
        status = AudioObjectGetPropertyData(audioDevices[i], &propertyAddress, 0, NULL, &dataSize,
&deviceName);
        if (kAudioHardwareNoError != status) {
            fprintf(stderr, "AudioObjectGetPropertyData (kAudioDevicePropertyDeviceNameCFString) failed:
%i\n", status);
```

```
            continue;
        }

        // Determine if the device is an input device (it is an input device if it has input channels)
        dataSize = 0;
        propertyAddress.mSelector = kAudioDevicePropertyStreamConfiguration;
        status = AudioObjectGetPropertyDataSize(audioDevices[i], &propertyAddress, 0, NULL, &dataSize);
        if (kAudioHardwareNoError != status) {
            fprintf(stderr, "AudioObjectGetPropertyDataSize (kAudioDevicePropertyStreamConfiguration)
failed: %i\n", status);
            continue;
        }

        AudioBufferList *bufferList = (AudioBufferList *)(malloc(dataSize));
        if (NULL == bufferList) {
            fputs("Unable to allocate memory", stderr);
            break;
        }

        status = AudioObjectGetPropertyData(audioDevices[i], &propertyAddress, 0, NULL, &dataSize,
bufferList);
        if (kAudioHardwareNoError != status || 0 == bufferList->mNumberBuffers) {
            if (kAudioHardwareNoError != status)
                fprintf(stderr, "AudioObjectGetPropertyData (kAudioDevicePropertyStreamConfiguration)
failed: %i\n", status);
            free(bufferList);
            bufferList = NULL;
            continue;
        }

        free(bufferList);
        bufferList = NULL;

        // Add a dictionary for this device to the array of input devices
        CFStringRef keys    []  = { CFSTR("deviceUID"),     CFSTR("deviceName")};
        CFStringRef values  []  = { deviceUID,              deviceName};

        CFDictionaryRef deviceDictionary = CFDictionaryCreate(kCFAllocatorDefault,
                                                              (const void **)(keys),
                                                              (const void **)(values),
                                                              2,
                                                              &kCFTypeDictionaryKeyCallBacks,
                                                              &kCFTypeDictionaryValueCallBacks);

        CFArrayAppendValue(inputDeviceArray, deviceDictionary);

        CFRelease(deviceDictionary);
        deviceDictionary = NULL;
    }

    free(audioDevices);
    audioDevices = NULL;

    // Return a non-mutable copy of the array
    CFArrayRef immutableInputDeviceArray = CFArrayCreateCopy(kCFAllocatorDefault, inputDeviceArray);
    CFRelease(inputDeviceArray);
    inputDeviceArray = NULL;

    return immutableInputDeviceArray;
}
```

For example, the UID for the built-in microphone is `BuiltInMicrophoneDevice`.

Audio device IDs on iOS

Audio device selection with the Speech SDK is not supported on iOS. However, apps using the SDK can influence

audio routing through the `AVAudioSession` Framework.

For example, the instruction

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryRecord
    withOptions:AVAudioSessionCategoryOptionAllowBluetooth error:NULL];
```

enables the use of a Bluetooth headset for a speech-enabled app.

Audio device IDs in JavaScript

In JavaScript the MediaDevices.enumerateDevices() method can be used to enumerate the media devices and find a device ID to pass to `fromMicrophone(...)`.

Next steps

Explore our samples on GitHub

See also

- Customize acoustic models
- Customize language models

Language identification (Preview)

11/23/2021 • 2 minutes to read • Edit Online

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

Language identification can also be used while doing speech translation, or by doing language identification during speech recognition.

To see which languages are available, see Language support.

Prerequisites

This article assumes you have an Azure subscription and speech resource, and also assumes knowledge of speech recognition basics.

Standalone language identification

In uses cases where you only want to detect the source language being spoken, you can use standalone language identification as shown in the following code sample. `SourceLanguageRecognizer` can also be used in continuous recognition scenarios.

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

var speechConfig = SpeechConfig.FromSubscription("<paste-your-subscription-key>","<paste-your-region>");
// can switch "Latency" to "Accuracy" depending on priority
speechConfig.SetProperty(PropertyId.SpeechServiceConnection_SingleLanguageIdPriority, "Latency");

var autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.FromLanguages(
        new string[] { "en-US", "de-DE" });

using (var recognizer = new SourceLanguageRecognizer(speechConfig, autoDetectSourceLanguageConfig))
{
    var result = await recognizer.RecognizeOnceAsync();
    if (result.Reason == ResultReason.RecognizedSpeech)
    {
        var lang = AutoDetectSourceLanguageResult.FromResult(result).Language;
        Console.WriteLine($"DETECTED: Language={lang}");
    }
}
```

See the sample on GitHub for more examples of standalone language identification, including an example of continuous identification.

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

auto config = SpeechConfig::FromSubscription("<paste-your-subscription-key>","<paste-your-region>");
config->SetProperty(PropertyId::SpeechServiceConnection_SingleLanguageIdPriority, "Latency");

auto autoDetectSourceLanguageConfig = AutoDetectSourceLanguageConfig::FromLanguages({ "en-US", "de-DE" });

auto recognizer = SourceLanguageRecognizer::FromConfig(config, autoDetectSourceLanguageConfig);
cout << "Say something...\n";

auto result = recognizer->RecognizeOnceAsync().get();
if (result->Reason == ResultReason::RecognizedSpeech)
{
    auto lidResult = AutoDetectSourceLanguageResult::FromResult(result);
    cout << "DETECTED: Language="<< lidResult->Language << std::endl;
}
```

See the sample on GitHub for more examples of standalone language identification, including an example of continuous identification.

```
import azure.cognitiveservices.speech as speechsdk

speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)

speech_config.set_property(property_id=speechsdk.PropertyId.SpeechServiceConnection_SingleLanguageIdPriority
, value='Accuracy')

speech_language_detection = speechsdk.SourceLanguageRecognizer(speech_config=speech_config,
auto_detect_source_language_config=auto_detect_source_language_config)

result = speech_language_detection.recognize_once()

# Check the result
if result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("RECOGNIZED: {}".format(result))
    detectedSrcLang =
result.properties[speechsdk.PropertyId.SpeechServiceConnection_AutoDetectSourceLanguageResult]
    print("Detected Language: {}".format(detectedSrcLang))
elif result.reason == speechsdk.ResultReason.NoMatch:
    print("No speech could be recognized")
elif result.reason == speechsdk.ResultReason.Canceled:
    cancellation_details = result.cancellation_details
    print("Speech Recognition canceled: {}".format(cancellation_details.reason))
    if cancellation_details.reason == speechsdk.CancellationReason.Error:
        print("Error details: {}".format(cancellation_details.error_details))
```

See the sample on GitHub for more examples of standalone language identification, including an example of continuous identification.

How to use language identification

11/23/2021 • 7 minutes to read • Edit Online

Language identification is used to determine the language being spoken in audio passed to the Speech SDK when compared against a list of provided languages. The value returned by language identification is then used to select the language model for speech to text, providing you with a more accurate transcription.

Language identification can also be used while doing speech translation, or by doing standalone identification.

To see which languages are available, see Language support.

Prerequisites

This article assumes you have an Azure subscription and speech resource, and also assumes knowledge of speech recognition basics. Complete the quickstart if you haven't already.

Language identification with speech-to-text

Language identification currently has a limit of **four languages** for at-start recognition, and **10 languages** for continuous recognition. Keep this limitation in mind when constructing your `AutoDetectSourceLanguageConfig` object. In the samples below, you use `AutoDetectSourceLanguageConfig` to define a list of possible languages that you want to identify, and then reference those languages when running speech recognition.

> **IMPORTANT**
>
> Continuous language identification is only supported in C#, C++, and Python.

The following example runs at-start recognition, prioritizing `Latency`. This property can also be set to `Accuracy` depending on the priority for your use-case. `Latency` is the best option to use if you need a low-latency result (e.g. for live streaming scenarios), but don't know the language in the audio sample.

`Accuracy` should be used in scenarios where the audio quality may be poor, and more latency is acceptable. For example, a voicemail could have background noise, or some silence at the beginning, and allowing the engine more time will improve recognition results.

In either case, at-start recognition as shown below should **not be used** for scenarios where the language may be changing within the same audio sample. See below for continuous recognition for these types of scenarios.

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

var speechConfig = SpeechConfig.FromSubscription("<paste-your-subscription-key>","<paste-your-region>");
// can switch "Latency" to "Accuracy" depending on priority
speechConfig.SetProperty(PropertyId.SpeechServiceConnection_SingleLanguageIdPriority, "Latency");

var autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.FromLanguages(
        new string[] { "en-US", "de-DE", "ja-JP", "de-DE" });

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;
}
```

The following example shows continuous speech recognition set up for a multilingual scenario. This example only uses `en-US` and `ja-JP` in the language config, but you can use up to **ten languages** for this design pattern. Each time speech is detected, the source language is identified and the audio is also converted to text output. This example uses `Latency` mode, which prioritizes response time.

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

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 = SpeechConfig.FromEndpoint(endpointUrl, "<paste-your-subscription-key>");
// can switch "Latency" to "Accuracy" depending on priority
config.SetProperty(PropertyId.SpeechServiceConnection_ContinuousLanguageIdPriority, "Latency");

var autoDetectSourceLanguageConfig = AutoDetectSourceLanguageConfig.FromLanguages(new string[] { "en-US",
"ja-JP" });

var stopRecognition = new TaskCompletionSource<int>();
using (var audioInput = AudioConfig.FromWavFileInput(@"path-to-your-audio-file.wav"))
{
    using (var recognizer = new SpeechRecognizer(config, autoDetectSourceLanguageConfig, audioInput))
    {
        // Subscribes to events.
        recognizer.Recognizing += (s, e) =>
        {
            if (e.Result.Reason == ResultReason.RecognizingSpeech)
            {
                Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
                var autoDetectSourceLanguageResult = AutoDetectSourceLanguageResult.FromResult(e.Result);
                Console.WriteLine($"DETECTED: Language={autoDetectSourceLanguageResult.Language}");
            }
        };

        recognizer.Recognized += (s, e) =>
        {
            if (e.Result.Reason == ResultReason.RecognizedSpeech)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                var autoDetectSourceLanguageResult = AutoDetectSourceLanguageResult.FromResult(e.Result);
```

```
                Console.WriteLine($"DETECTED: Language={autoDetectSourceLanguageResult.Language}");
            }
            else if (e.Result.Reason == ResultReason.NoMatch)
            {
                Console.WriteLine($"NOMATCH: Speech could not be recognized.");
            }
        };

        recognizer.Canceled += (s, e) =>
        {
            Console.WriteLine($"CANCELED: Reason={e.Reason}");

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

            stopRecognition.TrySetResult(0);
        };

        recognizer.SessionStarted += (s, e) =>
        {
            Console.WriteLine("\n    Session started event.");
        };

        recognizer.SessionStopped += (s, e) =>
        {
            Console.WriteLine("\n    Session stopped event.");
            Console.WriteLine("\nStop recognition.");
            stopRecognition.TrySetResult(0);
        };

        // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
        await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

        // Waits for completion.
        // Use Task.WaitAny to keep the task rooted.
        Task.WaitAny(new[] { stopRecognition.Task });

        // Stops recognition.
        await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
    }
}
```

> **NOTE**
>
> `Latency` and `Accuracy` modes, and multilingual continuous recognition, are currently only supported in C#, C++, and Python.

The following example runs at-start recognition, prioritizing `Latency`. This property can also be set to `Accuracy` depending on the priority for your use-case. `Latency` is the best option to use if you need a low-latency result (e.g. for a live streaming case), but don't know the language in the audio sample.

`Accuracy` should be used in scenarios where the audio quality may be poor, and more latency is acceptable. For example, a voicemail could have background noise, or some silence at the beginning, and allowing the engine more time will improve recognition results.

In either case, at-start recognition as shown below should **not be used** for scenarios where the language may be changing within the same audio sample. See below for continuous recognition for these types of scenarios.

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

auto speechConfig = SpeechConfig::FromSubscription("<paste-your-subscription-key>","<paste-your-region>");
speechConfig->SetProperty(PropertyId::SpeechServiceConnection_SingleLanguageIdPriority, "Latency");

auto autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig::FromLanguages({ "en-US", "de-DE", "ja-JP", "de-DE" });

auto recognizer = SpeechRecognizer::FromConfig(
    speechConfig,
    autoDetectSourceLanguageConfig
    );

speechRecognitionResult = recognizer->RecognizeOnceAsync().get();
auto autoDetectSourceLanguageResult =
    AutoDetectSourceLanguageResult::FromResult(speechRecognitionResult);
auto detectedLanguage = autoDetectSourceLanguageResult->Language;
```

The following example shows continuous speech recognition set up for a multilingual scenario. This example only uses `en-US` and `ja-JP` in the language config, but you can use up to **ten languages** for this design pattern. Each time speech is detected, the source language is identified and the audio is also converted to text output. This example uses `Latency` mode, which prioritizes response time.

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

auto region = "<paste-your-region>";
// currently the v2 endpoint is required for this design pattern
auto endpointString = std::format("wss://{}.stt.speech.microsoft.com/speech/universal/v2", region);
auto config = SpeechConfig::FromEndpoint(endpointString, "<paste-your-subscription-key>");

config->SetProperty(PropertyId::SpeechServiceConnection_ContinuousLanguageIdPriority, "Latency");
auto autoDetectSourceLanguageConfig = AutoDetectSourceLanguageConfig::FromLanguages({ "en-US", "ja-JP" });

auto audioInput = AudioConfig::FromWavFileInput("path-to-your-audio-file.wav");
auto recognizer = SpeechRecognizer::FromConfig(config, autoDetectSourceLanguageConfig, audioInput);

// promise for synchronization of recognition end.
promise<void> recognitionEnd;

// Subscribes to events.
recognizer->Recognizing.Connect([](const SpeechRecognitionEventArgs& e)
    {
        auto lidResult = AutoDetectSourceLanguageResult::FromResult(e.Result);
        cout << "Recognizing in " << lidResult->Language << ": Text =" << e.Result->Text << std::endl;
    });

recognizer->Recognized.Connect([](const SpeechRecognitionEventArgs& e)
    {
        if (e.Result->Reason == ResultReason::RecognizedSpeech)
        {
            auto lidResult = AutoDetectSourceLanguageResult::FromResult(e.Result);
            cout << "RECOGNIZED in " << lidResult->Language << ": Text=" << e.Result->Text << "\n"
                << "  Offset=" << e.Result->Offset() << "\n"
                << "  Duration=" << e.Result->Duration() << std::endl;
        }
        else if (e.Result->Reason == ResultReason::NoMatch)
        {
            cout << "NOMATCH: Speech could not be recognized." << std::endl;
        }
    });
```

```
recognizer->Canceled.Connect([&recognitionEnd](const SpeechRecognitionCanceledEventArgs& e)
    {
        cout << "CANCELED: Reason=" << (int)e.Reason << std::endl;

        if (e.Reason == CancellationReason::Error)
        {
            cout << "CANCELED: ErrorCode=" << (int)e.ErrorCode << "\n"
                << "CANCELED: ErrorDetails=" << e.ErrorDetails << "\n"
                << "CANCELED: Did you update the subscription info?" << std::endl;

            recognitionEnd.set_value(); // Notify to stop recognition.
        }
    });

recognizer->SessionStopped.Connect([&recognitionEnd](const SessionEventArgs& e)
    {
        cout << "Session stopped.";
        recognitionEnd.set_value(); // Notify to stop recognition.
    });

recognizer->StartContinuousRecognitionAsync().get();

// Waits for recognition end.
recognitionEnd.get_future().get();

// Stops recognition.
recognizer->StopContinuousRecognitionAsync().get();
```

NOTE

`Latency` and `Accuracy` modes, and multilingual continuous recognition, are currently only supported in C#, C++, and Python.

```
AutoDetectSourceLanguageConfig autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.fromLanguages(Arrays.asList("en-US", "de-DE"));

SpeechRecognizer recognizer = new SpeechRecognizer(
    speechConfig,
    autoDetectSourceLanguageConfig,
    audioConfig);

Future<SpeechRecognitionResult> future = recognizer.recognizeOnceAsync();
SpeechRecognitionResult result = future.get(30, TimeUnit.SECONDS);
AutoDetectSourceLanguageResult autoDetectSourceLanguageResult =
    AutoDetectSourceLanguageResult.fromResult(result);
String detectedLanguage = autoDetectSourceLanguageResult.getLanguage();

recognizer.close();
speechConfig.close();
autoDetectSourceLanguageConfig.close();
audioConfig.close();
result.close();
```

```
auto_detect_source_language_config = \
        speechsdk.languageconfig.AutoDetectSourceLanguageConfig(languages=["en-US", "de-DE"])
speech_recognizer = speechsdk.SpeechRecognizer(
        speech_config=speech_config,
        auto_detect_source_language_config=auto_detect_source_language_config,
        audio_config=audio_config)
result = speech_recognizer.recognize_once()
auto_detect_source_language_result = speechsdk.AutoDetectSourceLanguageResult(result)
detected_language = auto_detect_source_language_result.language
```

```
NSArray *languages = @[@"zh-CN", @"de-DE"];
SPXAutoDetectSourceLanguageConfiguration* autoDetectSourceLanguageConfig = \
        [[SPXAutoDetectSourceLanguageConfiguration alloc]init:languages];
SPXSpeechRecognizer* speechRecognizer = \
        [[SPXSpeechRecognizer alloc] initWithSpeechConfiguration:speechConfig
                            autoDetectSourceLanguageConfiguration:autoDetectSourceLanguageConfig
                                                    audioConfiguration:audioConfig];
SPXSpeechRecognitionResult *result = [speechRecognizer recognizeOnce];
SPXAutoDetectSourceLanguageResult *languageDetectionResult = [[SPXAutoDetectSourceLanguageResult alloc]
init:result];
NSString *detectedLanguage = [languageDetectionResult language];
```

```
var autoDetectConfig = SpeechSDK.AutoDetectSourceLanguageConfig.fromLanguages(["en-US", "de-DE"]);
var speechRecognizer = SpeechSDK.SpeechRecognizer.FromConfig(speechConfig, autoDetectConfig, audioConfig);
speechRecognizer.recognizeOnceAsync((result: SpeechSDK.SpeechRecognitionResult) => {
        var languageDetectionResult = SpeechSDK.AutoDetectSourceLanguageResult.fromResult(result);
        var detectedLanguage = languageDetectionResult.language;
},
{});
```

Use language detection with a custom Speech-to-Text model

In addition to language identification using Speech service base models, you can also specify a custom model for enhanced recognition. If a custom model isn't provided, the service will use the default language model.

The snippets below illustrate how to specify a custom model in your call to the Speech service. If the detected language is `en-US`, then the default model is used. If the detected language is `fr-FR`, then the endpoint for the custom model is used:

```
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);
```

```
std::vector<std::shared_ptr<SourceLanguageConfig>> sourceLanguageConfigs;
sourceLanguageConfigs.push_back(
    SourceLanguageConfig::FromLanguage("en-US"));
sourceLanguageConfigs.push_back(
    SourceLanguageConfig::FromLanguage("fr-FR", "The Endpoint Id for custom model of fr-FR"));

auto autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig::FromSourceLanguageConfigs(
        sourceLanguageConfigs);
```

```
List sourceLanguageConfigs = new ArrayList<SourceLanguageConfig>();
sourceLanguageConfigs.add(
    SourceLanguageConfig.fromLanguage("en-US"));
sourceLanguageConfigs.add(
    SourceLanguageConfig.fromLanguage("fr-FR", "The Endpoint Id for custom model of fr-FR"));

AutoDetectSourceLanguageConfig autoDetectSourceLanguageConfig =
    AutoDetectSourceLanguageConfig.fromSourceLanguageConfigs(
        sourceLanguageConfigs);
```

```
 en_language_config = speechsdk.languageconfig.SourceLanguageConfig("en-US")
 fr_language_config = speechsdk.languageconfig.SourceLanguageConfig("fr-FR", "The Endpoint Id for custom
model of fr-FR")
 auto_detect_source_language_config = speechsdk.languageconfig.AutoDetectSourceLanguageConfig(
        sourceLanguageConfigs=[en_language_config, fr_language_config])
```

```
SPXSourceLanguageConfiguration* enLanguageConfig = [[SPXSourceLanguageConfiguration alloc]init:@"en-US"];
SPXSourceLanguageConfiguration* frLanguageConfig = \
        [[SPXSourceLanguageConfiguration alloc]initWithLanguage:@"fr-FR"
                                                     endpointId:@"The Endpoint Id for custom model of fr-
FR"];
NSArray *languageConfigs = @[enLanguageConfig, frLanguageConfig];
SPXAutoDetectSourceLanguageConfiguration* autoDetectSourceLanguageConfig = \
        [[SPXAutoDetectSourceLanguageConfiguration
alloc]initWithSourceLanguageConfigurations:languageConfigs];
```

```
var enLanguageConfig = SpeechSDK.SourceLanguageConfig.fromLanguage("en-US");
var frLanguageConfig = SpeechSDK.SourceLanguageConfig.fromLanguage("fr-FR", "The Endpoint Id for custom
model of fr-FR");
var autoDetectConfig = SpeechSDK.AutoDetectSourceLanguageConfig.fromSourceLanguageConfigs([enLanguageConfig,
frLanguageConfig]);
```

Next steps

- See the sample code on GitHub for language identification
- See the sample code on GitHub for language identification
- See the sample code on GitHub for language identification
- See the sample code on GitHub for language identification
- See the sample code on GitHub for language identification

How to use batch transcription

11/23/2021 • 10 minutes to read • Edit Online

Batch transcription is a set of REST API operations that enables you to transcribe a large amount of audio in storage. You can point to audio files using a typical URI or a shared access signature (SAS) URI and asynchronously receive transcription results. With the v3.0 API, you can transcribe one or more audio files, or process a whole storage container.

You can use batch transcription REST APIs to call the following methods:

BATCH TRANSCRIPTION OPERATION	METHOD	REST API CALL
Creates a new transcription.	POST	speechtotext/v3.0/transcriptions
Retrieves a list of transcriptions for the authenticated subscription.	GET	speechtotext/v3.0/transcriptions
Gets a list of supported locales for offline transcriptions.	GET	speechtotext/v3.0/transcriptions/locales
Updates the mutable details of the transcription identified by its ID.	PATCH	speechtotext/v3.0/transcriptions/{id}
Deletes the specified transcription task.	DELETE	speechtotext/v3.0/transcriptions/{id}
Gets the transcription identified by the given ID.	GET	speechtotext/v3.0/transcriptions/{id}
Gets the result files of the transcription identified by the given ID.	GET	speechtotext/v3.0/transcriptions/{id}/files

You can review and test the detailed API, which is available as a Swagger document.

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

Prerequisites

As with all features of the Speech service, you create a subscription key from the Azure portal by following our Get started guide.

> **NOTE**
> A standard subscription (S0) for Speech service is required to use batch transcription. Free subscription keys (F0) will not work. For more information, see pricing and limits.

If you plan to customize models, follow the steps in Acoustic customization and Language customization. To use the created models in batch transcription, you need their model location. You can retrieve the model location when you inspect the details of the model (`self` property). A deployed custom endpoint is *not needed* for the

batch transcription service.

> **NOTE**
> As a part of the REST API, Batch Transcription has a set of quotas and limits, which we encourage to review. To take the full advantage of Batch Transcription ability to efficiently transcribe a large number of audio files we recommend always sending multiple files per request or pointing to a Blob Storage container with the audio files to transcribe. The service will transcribe the files concurrently reducing the turnaround time. Using multiple files in a single request is very simple and straightforward - see Configuration section.

Batch transcription API

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 being created for each channel. To create an ordered final transcript, use the timestamps generated per utterance.

Configuration

Configuration parameters are provided as JSON.

Transcribing one or more individual files. If you have more than one file to transcribe, we recommend sending multiple files in one request. The example below is using three files:

```
{
  "contentUrls": [
    "<URL to an audio file 1 to transcribe>",
    "<URL to an audio file 2 to transcribe>",
    "<URL to an audio file 3 to transcribe>"
  ],
  "properties": {
    "wordLevelTimestampsEnabled": true
  },
  "locale": "en-US",
  "displayName": "Transcription of file using default model for en-US"
}
```

Processing a whole storage container. Container SAS should contain `r` (read) and `l` (list) permissions:

```
{
  "contentContainerUrl": "<SAS URL to the Azure blob container to transcribe>",
  "properties": {
    "wordLevelTimestampsEnabled": true
  },
  "locale": "en-US",
  "displayName": "Transcription of container using default model for en-US"
}
```

Use a custom trained model in a batch transcription. The example is using three files:

```
{
  "contentUrls": [
    "<URL to an audio file 1 to transcribe>",
    "<URL to an audio file 2 to transcribe>",
    "<URL to an audio file 3 to transcribe>"
  ],
  "properties": {
    "wordLevelTimestampsEnabled": true
  },
  "locale": "en-US",
  "model": {
    "self": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/models/{id}"
  },
  "displayName": "Transcription of file using default model for en-US"
}
```

Configuration properties

Use these optional properties to configure transcription:

Parameter	Description
`profanityFilterMode`	Optional, defaults to `Masked`. Specifies how to handle profanity in recognition results. Accepted values are `None` to disable profanity filtering, `Masked` to replace profanity with asterisks, `Removed` to remove all profanity from the result, or `Tags` to add "profanity" tags.
`punctuationMode`	Optional, defaults to `DictatedAndAutomatic`. Specifies how to handle punctuation in recognition results. Accepted values are `None` to disable punctuation, `Dictated` to imply explicit (spoken) punctuation, `Automatic` to let the decoder deal with punctuation, or `DictatedAndAutomatic` to use dictated and automatic punctuation.
`wordLevelTimestampsEnabled`	Optional, `false` by default. Specifies if word level timestamps should be added to the output.
`diarizationEnabled`	Optional, `false` by default. Specifies that diarization analysis should be carried out on the input, which is expected to be mono channel containing two voices. Note: Requires `wordLevelTimestampsEnabled` to be set to `true`.
`channels`	Optional, `0` and `1` transcribed by default. An array of channel numbers to process. Here a subset of the available channels in the audio file can be specified to be processed (for example `0` only).
`timeToLive`	Optional, no deletion by default. A duration to automatically delete transcriptions after completing the transcription. The `timeToLive` is useful in mass processing transcriptions to ensure they will be eventually deleted (for example, `PT12H` for 12 hours).
`destinationContainerUrl`	Optional URL with ad hoc SAS to a writeable container in Azure. The result is stored in this container. SAS with

stored access policy are **not** supported. When not specified, Microsoft stores the results in a storage container managed by Microsoft. When the transcription is deleted by calling Delete transcription, the result data will also be deleted.

Storage

Batch transcription can read audio from a public-visible internet URI, and can read audio or write transcriptions using a SAS URI with Azure Blob storage.

Batch transcription result

For each audio input, one transcription result file is created. The Get transcriptions files operation returns a list of result files for this transcription. To find the transcription file for a specific input file, filter all returned files with `kind` `==` `Transcription` and `name` `==` `{originalInputName.suffix}.json`.

Each transcription result file has this format:

```
{
  "source": "...",                      // sas url of a given contentUrl or the path relative to the root of
a given container
  "timestamp": "2020-06-16T09:30:21Z",  // creation time of the transcription, ISO 8601 encoded timestamp,
combined date and time
  "durationInTicks": 41200000,          // total audio duration in ticks (1 tick is 100 nanoseconds)
  "duration": "PT4.12S",                // total audio duration, ISO 8601 encoded duration
  "combinedRecognizedPhrases": [        // concatenated results for simple access in single string for each
channel
    {
      "channel": 0,                     // channel number of the concatenated results
      "lexical": "hello world",
      "itn": "hello world",
      "maskedITN": "hello world",
      "display": "Hello world."
    }
  ],
  "recognizedPhrases": [                // results for each phrase and each channel individually
    {
      "recognitionStatus": "Success",   // recognition state, e.g. "Success", "Failure"
      "speaker": 1,                     // if `diarizationEnabled` is `true`, this is the identified speaker
(1 or 2), otherwise this property is not present
      "channel": 0,                     // channel number of the result
      "offset": "PT0.07S",              // offset in audio of this phrase, ISO 8601 encoded duration
      "duration": "PT1.59S",            // audio duration of this phrase, ISO 8601 encoded duration
      "offsetInTicks": 700000.0,        // offset in audio of this phrase in ticks (1 tick is 100
nanoseconds)
      "durationInTicks": 15900000.0,    // audio duration of this phrase in ticks (1 tick is 100
nanoseconds)

      // possible transcriptions of the current phrase with confidences
      "nBest": [
        {
          "confidence": 0.898652852,    // confidence value for the recognition of the whole phrase
          "lexical": "hello world",
          "itn": "hello world",
          "maskedITN": "hello world",
          "display": "Hello world.",

          // if wordLevelTimestampsEnabled is `true`, there will be a result for each word of the phrase,
otherwise this property is not present
          "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
            }
          ]
        }
      ]
    }
  ]
}
```

The result contains the following fields:

Field

Content

`lexical`

The actual words recognized.

`itn`

Inverse-text-normalized form of the recognized text. Abbreviations ("doctor smith" to "dr smith"), phone numbers, and other transformations are applied.

`maskedITN`

The ITN form with profanity masking applied.

`display`

The display form of the recognized text. Added punctuation and capitalization are included.

Speaker separation (diarization)

Diarization is the process of separating speakers in a piece of audio. The batch pipeline supports diarization and is capable of recognizing two speakers on mono channel recordings. The feature is not available on stereo recordings.

The output of transcription with diarization enabled contains a `Speaker` entry for each transcribed phrase. If diarization is not used, the `Speaker` property is not present in the JSON output. For diarization we support two voices, so the speakers are identified as `1` or `2`.

To request diarization, add set the `diarizationEnabled` property to `true` like the HTTP request shows below.

```
{
  "contentUrls": [
    "<URL to an audio file to transcribe>",
  ],
  "properties": {
    "diarizationEnabled": true,
    "wordLevelTimestampsEnabled": true,
    "punctuationMode": "DictatedAndAutomatic",
    "profanityFilterMode": "Masked"
  },
  "locale": "en-US",
  "displayName": "Transcription of file using default model for en-US"
}
```

Word-level timestamps must be enabled as the parameters in the above request indicate.

Best practices

The batch transcription service can handle large number of submitted transcriptions. You can query the status of your transcriptions with Get transcriptions. Call Delete transcription regularly from the service once you retrieved the results. Alternatively set `timeToLive` property to ensure eventual deletion of the results.

> **TIP**
>
> You can use the Ingestion Client tool and resulting solution to process high volume of audio.

Sample code

Complete samples are available in the GitHub sample repository inside the `samples/batch` subdirectory.

Update the sample code with your subscription information, service region, URI pointing to the audio file to transcribe, and model location if you're using a custom model.

```
var newTranscription = new Transcription
{
    DisplayName = DisplayName,
    Locale = Locale,
    ContentUrls = new[] { RecordingsBlobUri },
    //ContentContainerUrl = ContentAzureBlobContainer,
    Model = CustomModel,
    Properties = new TranscriptionProperties
    {
        IsWordLevelTimestampsEnabled = true,
        TimeToLive = TimeSpan.FromDays(1)
    }
};

newTranscription = await client.CreateTranscriptionAsync(newTranscription).ConfigureAwait(false);
Console.WriteLine($"Created transcription {newTranscription.Self}");
```

The sample code sets up the client and submits the transcription request. It then polls for the status information and print details about the transcription progress.

```
// get the status of our transcriptions periodically and log results
int completed = 0, running = 0, notStarted = 0;
while (completed < 1)
{
    completed = 0; running = 0; notStarted = 0;

    // get all transcriptions for the user
    paginatedTranscriptions = null;
    do
    {
        // <transcriptionstatus>
        if (paginatedTranscriptions == null)
        {
            paginatedTranscriptions = await client.GetTranscriptionsAsync().ConfigureAwait(false);
        }
        else
        {
            paginatedTranscriptions = await
client.GetTranscriptionsAsync(paginatedTranscriptions.NextLink).ConfigureAwait(false);
        }

        // delete all pre-existing completed transcriptions. If transcriptions are still running or not
started, they will not be deleted
        foreach (var transcription in paginatedTranscriptions.Values)
        {
            switch (transcription.Status)
            {
                case "Failed":
                case "Succeeded":
                    // we check to see if it was one of the transcriptions we created from this client.
                    if (!createdTranscriptions.Contains(transcription.Self))
                    {
                        // not created form here, continue
                        continue;
                    }

                    completed++;
```

```
                    // if the transcription was successful, check the results
                    if (transcription.Status == "Succeeded")
                    {
                        var paginatedfiles = await
client.GetTranscriptionFilesAsync(transcription.Links.Files).ConfigureAwait(false);

                        var resultFile = paginatedfiles.Values.FirstOrDefault(f => f.Kind ==
ArtifactKind.Transcription);
                        var result = await client.GetTranscriptionResultAsync(new
Uri(resultFile.Links.ContentUrl)).ConfigureAwait(false);
                        Console.WriteLine("Transcription succeeded. Results: ");
                        Console.WriteLine(JsonConvert.SerializeObject(result,
SpeechJsonContractResolver.WriterSettings));
                    }
                    else
                    {
                        Console.WriteLine("Transcription failed. Status: {0}",
transcription.Properties.Error.Message);
                    }

                    break;

                case "Running":
                    running++;
                    break;

                case "NotStarted":
                    notStarted++;
                    break;
            }
        }

        // for each transcription in the list we check the status
        Console.WriteLine(string.Format("Transcriptions status: {0} completed, {1} running, {2} not started
yet", completed, running, notStarted));
    }
    while (paginatedTranscriptions.NextLink != null);

    // </transcriptionstatus>
    // check again after 1 minute
    await Task.Delay(TimeSpan.FromMinutes(1)).ConfigureAwait(false);
}
```

For full details about the preceding calls, see our Swagger document. For the full sample shown here, go to GitHub in the `samples/batch` subdirectory.

This sample uses an asynchronous setup to post audio and receive transcription status. The `PostTranscriptions` method sends the audio file details and the `GetTranscriptions` method receives the states. `PostTranscriptions` returns a handle, and `GetTranscriptions` uses it to create a handle to get the transcription status.

This sample code doesn't specify a custom model. The service uses the baseline model for transcribing the file or files. To specify the model, you can pass on the same method the model reference for the custom model.

> **NOTE**
>
> For baseline transcriptions, you don't need to declare the ID for the baseline model.

Next steps

- Speech to text v3 API reference

Migrate code from v2.0 to v3.0 of the REST API

11/23/2021 • 13 minutes to read • Edit Online

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.

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 are**not**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 propertyidtoselfin 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 Swagger document 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:

```
{
    "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:

```
{
    "self": "https://westus.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://westus.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:

```
{
    "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/v3.0/{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:

```
{
  "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:

```
{
  "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.

```
{
  "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:

```
{
  "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:

```
{
    "self": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-
b14b-6d44efb158f3",
    "links": {
      "files": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-
4880-b14b-6d44efb158f3/files"
    }
}
```

A GET on `$.links.files` would result in:

```
{
  "values": [
    {
      "self": "https://westus.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://westus.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://westus.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:

```
{
    "name": "Transcription using locale en-US"
}
```

v3 transcription:

```
{
    "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:

```
{
  "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:

```
{
  "self": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/transcriptions/9891c965-bb32-4880-
b14b-6d44efb158f3",
  "model": {
    "self": "https://westus.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:

```
{
  "self": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/afa0669c-a01e-4693-ae3a-93baf40f26d6",
  "links": {
    "logs": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/afa0669c-a01e-4693-ae3a-93baf40f26d6/files/logs"
  }
}
```

Response of GET `$.links.logs`:

```
{
  "values": [
    {
      "self": "https://westus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/6d72ad7e-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-bd27-8804f9eed13f.wav?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://westus.api.cognitive.microsoft.com/speechtotext/v3.0/endpoints/afa0669c-a01e-4693-ae3a-93baf40f26d6/files/logs?top=2&SkipToken=2!188!MDAwMDk1ITZhMjhiMDllLTg0MDYtNDViMi1hMGRkLWFlNzRlOGRhZWJkNi8yMDIwLTA0LTAxLzEyNDY0M182MzI5NGRkMi1mZGYzLTRhZmEtOTA0NC1mODU5ZTcxOWJiYzYud2F2ITAwMDAyOCE5OTk5LTEyLTMxVDIzOjU5OjU5Ljk5OTk5OTlaIQ--"
}
```

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:

```
{
  "properties": {
    "customerDefinedKey": "value",
    "diarizationEnabled": "False",
    "wordLevelTimestampsEnabled": "False"
  }
}
```

v3 transcription entity:

```
{
  "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://{region}.api.cognitive.microsoft.com/speechtotext/v3.0/evaluations`.

> **IMPORTANT**
> Rename the path segment `accuracytests` to `evaluations` in your client code.

Next steps

Examine all features of these commonly used REST APIs provided by Speech services:

- Speech-to-text REST API
- Swagger document for v3 of the REST API

- For sample code to perform batch transcriptions, view the the GitHub sample repository in the `samples/batch` subdirectory.

Pronunciation assessment

11/23/2021 • 4 minutes to read • Edit Online

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. Educators can use the capability to evaluate pronunciation of multiple speakers in real-time.

In this article, you'll learn how to set up `PronunciationAssessmentConfig` and retrieve the `PronunciationAssessmentResult` using the speech SDK.

> **NOTE**
>
> The pronunciation assessment feature currently supports `en-US` language, which is available on all speech-to-text regions. The support for `en-GB` and `zh-CN` languages is under preview.

Pronunciation assessment with the Speech SDK

In the samples below, you'll create a `PronunciationAssessmentConfig`, then apply it to a `SpeechRecognizer`.

The following snippets illustrate how to use language identification in your apps:

```
var pronunciationAssessmentConfig = new PronunciationAssessmentConfig(
    "reference text", GradingSystem.HundredMark, Granularity.Phoneme);

using (var recognizer = new SpeechRecognizer(
    speechConfig,
    audioConfig))
{
    // apply the pronunciation assessment configuration to the speech recognizer
    pronunciationAssessmentConfig.ApplyTo(recognizer);
    var speechRecognitionResult = await recognizer.RecognizeOnceAsync();
    var pronunciationAssessmentResult =
        PronunciationAssessmentResult.FromResult(speechRecognitionResult);
    var pronunciationScore = pronunciationAssessmentResult.PronunciationScore;
}
```

```
auto pronunciationAssessmentConfig =
    PronunciationAssessmentConfig::Create("reference text",
        PronunciationAssessmentGradingSystem::HundredMark,
        PronunciationAssessmentGranularity::Phoneme);

auto recognizer = SpeechRecognizer::FromConfig(
    speechConfig,
    audioConfig);

// apply the pronunciation assessment configuration to the speech recognizer
pronunciationAssessmentConfig->ApplyTo(recognizer);
speechRecognitionResult = recognizer->RecognizeOnceAsync().get();
auto pronunciationAssessmentResult =
    PronunciationAssessmentResult::FromResult(speechRecognitionResult);
auto pronunciationScore = pronunciationAssessmentResult->PronunciationScore;
```

```
PronunciationAssessmentConfig pronunciationAssessmentConfig =
    new PronunciationAssessmentConfig("reference text",
        PronunciationAssessmentGradingSystem.HundredMark,
        PronunciationAssessmentGranularity.Phoneme);

SpeechRecognizer recognizer = new SpeechRecognizer(
    speechConfig,
    audioConfig);

// apply the pronunciation assessment configuration to the speech recognizer
pronunciationAssessmentConfig.applyTo(recognizer);
Future<SpeechRecognitionResult> future = recognizer.recognizeOnceAsync();
SpeechRecognitionResult result = future.get(30, TimeUnit.SECONDS);
PronunciationAssessmentResult pronunciationAssessmentResult =
    PronunciationAssessmentResult.fromResult(result);
Double pronunciationScore = pronunciationAssessmentResult.getPronunciationScore();

recognizer.close();
speechConfig.close();
audioConfig.close();
pronunciationAssessmentConfig.close();
result.close();
```

```
pronunciation_assessment_config = \
        speechsdk.PronunciationAssessmentConfig(reference_text='reference text',
                grading_system=msspeech.PronunciationAssessmentGradingSystem.HundredMark,
                granularity=msspeech.PronunciationAssessmentGranularity.Phoneme)
speech_recognizer = speechsdk.SpeechRecognizer(
        speech_config=speech_config, \
        audio_config=audio_config)

# apply the pronunciation assessment configuration to the speech recognizer
pronunciation_assessment_config.apply_to(speech_recognizer)
result = speech_recognizer.recognize_once()
pronunciation_assessment_result = speechsdk.PronunciationAssessmentResult(result)
pronunciation_score = pronunciation_assessment_result.pronunciation_score
```

```
var pronunciationAssessmentConfig = new SpeechSDK.PronunciationAssessmentConfig("reference text",
    PronunciationAssessmentGradingSystem.HundredMark,
    PronunciationAssessmentGranularity.Word, true);
var speechRecognizer = SpeechSDK.SpeechRecognizer.FromConfig(speechConfig, audioConfig);
// apply the pronunciation assessment configuration to the speech recognizer
pronunciationAssessmentConfig.applyTo(speechRecognizer);

speechRecognizer.recognizeOnceAsync((result: SpeechSDK.SpeechRecognitionResult) => {
        var pronunciationAssessmentResult = SpeechSDK.PronunciationAssessmentResult.fromResult(result);
        var pronunciationScore = pronunciationAssessmentResult.pronunciationScore;
        var wordLevelResult = pronunciationAssessmentResult.detailResult.Words;
},
{});
```

```
SPXPronunciationAssessmentConfiguration* pronunciationAssessmentConfig =
    [[SPXPronunciationAssessmentConfiguration alloc]init:@"reference text"
                                          gradingSystem:SPXPronunciationAssessmentGradingSystem_HundredMark
                                            granularity:SPXPronunciationAssessmentGranularity_Phoneme];

SPXSpeechRecognizer* speechRecognizer = \
        [[SPXSpeechRecognizer alloc] initWithSpeechConfiguration:speechConfig
                                             audioConfiguration:audioConfig];

// apply the pronunciation assessment configuration to the speech recognizer
[pronunciationAssessmentConfig applyToRecognizer:speechRecognizer];

SPXSpeechRecognitionResult *result = [speechRecognizer recognizeOnce];
SPXPronunciationAssessmentResult* pronunciationAssessmentResult = [[SPXPronunciationAssessmentResult alloc]
init:result];
double pronunciationScore = pronunciationAssessmentResult.pronunciationScore;
```

Pronunciation assessment configuration parameters

This table lists the configuration parameters for pronunciation assessment.

PARAMETER	DESCRIPTION	REQUIRED?
ReferenceText	The text that the pronunciation will be evaluated against.	Required
GradingSystem	The point system for score calibration. The `FivePoint` system gives a 0-5 floating point score, and `HundredMark` gives a 0-100 floating point score. Default: `FivePoint`.	Optional
Granularity	The evaluation granularity. Accepted values are `Phoneme`, which shows the score on the full text, word and phoneme level, `Word`, which shows the score on the full text and word level, `FullText`, which shows the score on the full text level only. Default: `Phoneme`.	Optional
EnableMiscue	Enables miscue calculation. With this enabled, the pronounced words will be compared to the reference text, and will be marked with omission/insertion based on the comparison. Accepted values are `False` and `True`. Default: `False`.	Optional
ScenarioId	A GUID indicating a customized point system.	Optional

Pronunciation assessment result parameters

This table lists the result parameters of pronunciation assessment.

PARAMETER	DESCRIPTION

PARAMETER	DESCRIPTION
`AccuracyScore`	Pronunciation accuracy of the speech. Accuracy indicates how closely the phonemes match a native speaker's pronunciation. Word and full text level accuracy score is aggregated from phoneme level accuracy score.
`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, determined by calculating the ratio of pronounced words to reference text input.
`PronunciationScore`	Overall score indicating the pronunciation quality of the given speech. This is aggregated from `AccuracyScore`, `FluencyScore` and `CompletenessScore` with weight.
`ErrorType`	This value indicates whether a word is omitted, inserted or badly pronounced, compared to `ReferenceText`. Possible values are `None` (meaning no error on this word), `Omission`, `Insertion` and `Mispronunciation`.

Sample responses

A typical pronunciation assessment result in JSON:

```
{
  "RecognitionStatus": "Success",
  "Offset": "400000",
  "Duration": "11000000",
  "NBest": [
      {
        "Confidence" : "0.87",
        "Lexical" : "good morning",
        "ITN" : "good morning",
        "MaskedITN" : "good morning",
        "Display" : "Good morning.",
        "PronunciationAssessment":
        {
            "PronScore" : 84.4,
            "AccuracyScore" : 100.0,
            "FluencyScore" : 74.0,
            "CompletenessScore" : 100.0,
        },
        "Words": [
            {
              "Word" : "Good",
              "Offset" : 500000,
              "Duration" : 2700000,
              "PronunciationAssessment":
              {
                "AccuracyScore" : 100.0,
                "ErrorType" : "None"
              }
            },
            {
              "Word" : "morning",
              "Offset" : 5300000,
              "Duration" : 900000,
              "PronunciationAssessment":
              {
                "AccuracyScore" : 100.0,
                "ErrorType" : "None"
              }
            }
        ]
      }
  ]
}
```

Next steps

- Watch the video introduction and video tutorial of pronunciation assessment
- Try out the pronunciation assessment demo
- See the sample code on GitHub for pronunciation assessment.
- See the sample code on GitHub for pronunciation assessment.
- See the sample code on GitHub for pronunciation assessment.
- See the sample code on GitHub for pronunciation assessment.
- See the sample code on GitHub for pronunciation assessment.
- Speech SDK reference documentation
- Create a free Azure account

How to: Monitor and control service connections with the Speech SDK

11/23/2021 • 2 minutes to read • Edit Online

`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`.

```
var connection = Connection.FromRecognizer(recognizer);
```

```
auto connection = Connection::FromRecognizer(recognizer);
```

```
Connection 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.

```
connection.Connected += (sender, connectedEventArgs) =>
{
    Console.WriteLine($"Successfully connected, sessionId: {connectedEventArgs.SessionId}");
};
connection.Disconnected += (sender, disconnectedEventArgs) =>
{
    Console.WriteLine($"Disconnected, sessionId: {disconnectedEventArgs.SessionId}");
};
```

```
connection->Connected += [&](ConnectionEventArgs& args)
{
    std::cout << "Successfully connected, sessionId: " << args.SessionId << std::endl;
};
connection->Disconnected += [&](ConnectionEventArgs& args)
{
    std::cout << "Disconnected, sessionId: " << args.SessionId << std::endl;
};
```

```
connection.connected.addEventListener((s, connectionEventArgs) -> {
    System.out.println("Successfully connected, sessionId: " + connectionEventArgs.getSessionId());
});
connection.disconnected.addEventListener((s, connectionEventArgs) -> {
    System.out.println("Disconnected, sessionId: " + connectionEventArgs.getSessionId());
});
```

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.

```
try
{
    connection.Open(forContinuousRecognition: false);
}
catch (ApplicationException ex)
{
    Console.WriteLine($"Couldn't pre-connect. Details: {ex.Message}");
}
// ... Use the SpeechRecognizer
connection.Close();
```

```
try
{
    connection->Open(false);
}
catch (std::runtime_error&)
{
    std::cout << "Unable to pre-connect." << std::endl;
}
// ... Use the SpeechRecognizer
connection->Close();
```

```
try {
    connection.openConnection(false);
} catch {
   System.out.println("Unable to pre-connect.");
}
// ... Use the SpeechRecognizer
connection.closeConnection();
```

Next steps

Explore our samples on GitHub

What is Custom Speech?

11/23/2021 • 3 minutes to read • Edit Online

Custom Speech allows you to evaluate and improve the Microsoft speech-to-text accuracy for your applications and products. Follow the links in this article to start creating a custom speech-to-text experience.

What's in Custom Speech?

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

This diagram highlights the pieces that make up the Custom Speech area of the Speech Studio. Use the links below to learn more about each step.

1. Subscribe and create a project. Create an Azure account and subscribe to the Speech service. This unified subscription gives you access to speech-to-text, text-to-speech, speech translation, and the Speech Studio. Then use your Speech service subscription to create your first Custom Speech project.

2. Upload test data. Upload test data (audio files) to evaluate the Microsoft speech-to-text offering for your applications, tools, and products.

3. Inspect recognition quality. Use the Speech Studio to play back uploaded audio and inspect the speech recognition quality of your test data. For quantitative measurements, see Inspect data.

4. Evaluate and improve accuracy. Evaluate and improve the accuracy of the speech-to-text model. The Speech Studio will provide a *Word Error Rate*, which you can use to determine if additional training is required. If you're satisfied with the accuracy, you can use the Speech service APIs directly. If you want to improve accuracy by a relative average of 5% to 20%, use the **Training** tab in the portal to upload additional training data, like human-labeled transcripts and related text.

5. Train and deploy a model. Improve the accuracy of your speech-to-text model by providing written transcripts (10 to 1,000 hours) and related text (<200 MB) along with your audio test data. This data helps to train the speech-to-text model. After training, retest. If you're satisfied with the result, you can deploy your model to a custom endpoint.

Set up your Azure account

You need to have an Azure account and Speech service subscription before you can use the Speech Studio to create a custom model. If you don't have an account and subscription, try the Speech service for free.

If you plan to train a custom model with **audio data**, pick one of the following regions that have dedicated hardware available for training. This will reduce the time it takes to train a model and allow you to use more audio for training. In these regions, the Speech service will use up to 20 hours of audio for training; in other regions it will only use up to 8 hours.

- Australia East
- Canada Central
- Central India
- East US
- East US 2
- North Central US
- North Europe
- South Central US
- Southeast Asia
- UK South
- US Gov Arizona
- US Gov Virginia
- West Europe
- West US 2

After you create an Azure account and a Speech service subscription, you'll need to sign in to the Speech Studio and connect your subscription.

1. Sign in to the Speech Studio.
2. Select the subscription you need to work in and create a speech project.
3. If you want to modify your subscription, select the cog button in the top menu.

How to create a project

Content like data, models, tests, and endpoints are organized into *projects* in the Speech Studio. Each project is specific to a domain and country/language. For example, you might create a project for call centers that use English in the United States.

To create your first project, select **Speech-to-text/Custom speech**, and then select **New Project**. Follow the instructions provided by the wizard to create your project. After you create 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.

> **IMPORTANT**
>
> The Speech Studio (formerly known as "Custom Speech portal") was recently updated! If you created previous data, models, tests, and published endpoints in the CRIS.ai portal or with APIs, you need to create a new project in the new portal to connect to these old entities.

Model and Endpoint lifecycle

Older models typically become less useful over time because the newest model usually has higher accuracy. Therefore, base models as well as custom models and endpoints created through the portal are subject to

expiration after 1 year for adaptation and 2 years for decoding. See a detailed description in the Model and endpoint lifecycle article.

Next steps

- Prepare and test your data
- Inspect your data
- Evaluate and improve model accuracy
- Train and deploy a model

Prepare data for Custom Speech

11/23/2021 • 16 minutes to read • Edit Online

When testing the accuracy of Microsoft speech recognition or training your custom models, you'll need audio and text data. On this page, we cover the types of data a custom speech model needs.

Data diversity

Text and audio used to test and train a custom model need to include samples from a diverse set of speakers and scenarios you need your model to recognize. Consider these factors when gathering data for custom model testing and training:

- Your text and speech audio data need to cover the kinds of verbal statements your users will make when interacting with your model. For example, a model that raises and lowers the temperature needs training on statements people might make to request such changes.
- Your data need to include all speech variances your model will need to recognize. Many factors can vary speech, including accents, dialects, language-mixing, age, gender, voice pitch, stress level, and time of day.
- You must include samples from different environments (indoor, outdoor, road noise) where your model will be used.
- Audio must be gathered using hardware devices the production system will use. If your model needs to identify speech recorded on recording devices of varying quality, the audio data you provide to train your model must also represent these diverse scenarios.
- You can add more data to your model later, but take care to keep the dataset diverse and representative of your project needs.
- Including data that is *not* within your custom model recognition needs can harm recognition quality overall, so do not include data that your model does not need to transcribe.

A model trained on a subset of scenarios can only perform well in those scenarios. Carefully choose data that represents the full scope of scenarios you need your custom model to recognize.

> **TIP**
>
> Start with small sets of sample data that match the language and acoustics your model will encounter. For example, record a small but representative sample of audio on the same hardware and in the same acoustic environment your model will find in production scenarios. Small datasets of representative data can expose problems before you have invested in gathering a much larger datasets for training.
>
> To quickly get started, consider using sample data. See this GitHub repository for sample Custom Speech data

Data types

This table lists accepted data types, when each data type should be used, and the recommended quantity. Not every data type is required to create a model. Data requirements will vary depending on whether you're creating a test or training a model.

DATA TYPE	USED FOR TESTING	RECOMMENDED QUANTITY	USED FOR TRAINING	RECOMMENDED QUANTITY

DATA TYPE	USED FOR TESTING	RECOMMENDED QUANTITY	USED FOR TRAINING	RECOMMENDED QUANTITY
Audio only	Yes Used for visual inspection	5+ audio files	No	N/A
Audio + Human-labeled transcripts	Yes Used to evaluate accuracy	0.5-5 hours of audio	Yes	1-20 hours of audio
Plain text	No	N/a	Yes	1-200 MB of related text
Structured text (Public Preview)	No	N/a	Yes	Up to 10 classes with up to 4000 items and up to 50,000 training sentences
Pronunciation	No	N/a	Yes	1 KB - 1 MB of pronunciation text

Files should be grouped by type into a dataset and uploaded as a .zip file. Each dataset can only contain a single data type.

TIP

When you train a new model, 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 vs. days).

NOTE

Not all base models support training with audio. If a base model does not support it, the Speech service will only use the text from the transcripts and ignore the audio. See Language support for a list of base models that support training with audio data. Even if a base model supports training with audio data, the service might use only part of the audio. Still it will use all the transcripts.

In cases when you change the base model used for training, and you have audio in the training dataset, *always* check whether the new selected base model supports training with audio data. If the previously used base model did not support training with audio data, and the training dataset contains audio, training time with the new base model will **drastically** increase, and may easily go from several hours to several days and more. This is especially true if your Speech service subscription is **not** in a region with the dedicated hardware for training.

If you face the issue described in the paragraph above, you can quickly decrease the training time by reducing the amount of audio in the dataset or removing it completely and leaving only the text. The latter option is highly recommended if your Speech service subscription is **not** in a region with the dedicated hardware for training.

In regions with dedicated hardware for training, the Speech service will use up to 20 hours of audio for training. In other regions, it will only use up to 8 hours of audio.

> **NOTE**
>
> Training with structured text is only supported for these locales: en-US, en-UK, en-IN, de-DE, fr-FR, fr-CA, es-ES, es-MX and 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.

Upload data

To upload your data, navigate to Speech Studio. After creating a project, navigate to **Speech datasets** tab, and click **Upload data** to launch the wizard and create your first dataset. Select a speech data type for your dataset, and upload your data.

> **NOTE**
>
> If your dataset file size exceeds 128 MB, you can only upload it using *Azure Blob or shared location* option. You can also use Speech-to-text REST API v3.0 to upload a dataset of any allowed size. See the next section for details.

First, you need to specify whether the dataset is to be used for **Training** or **Testing**. There are many types of data that can be uploaded and used for **Training** or **Testing**. Each dataset you upload must be correctly formatted before uploading, and must meet the requirements for the data type that you choose. Requirements are listed in the following sections.

After your dataset is uploaded, you have a few options:

- You can navigate to the **Train custom models** tab to train a custom model.
- You can navigate to the **Test models** tab to visually inspect quality with audio only data or evaluate accuracy with audio + human-labeled transcription data.

Upload data using Speech-to-text REST API v3.0

You can use Speech-to-text REST API v3.0 to automate any operations related to your custom models. In particular, you can use it to upload a dataset. This is particularly useful when your dataset file exceeds 128 MB, because files that large cannot be uploaded using *Local file* option in Speech Studio. (You can also use *Azure Blob or shared location* option in Speech Studio for the same purpose as described in the previous section.)

To create and upload a dataset use Create Dataset request.

REST API created datasets and Speech Studio projects

A dataset created with the Speech-to-text REST API v3.0 will *not* be connected to any of the Speech Studio projects, unless a special parameter is specified in the request body (see below). Connection with a Speech Studio project is *not* required for any model customization operations, if they are performed via the REST API.

When you log on to the Speech Studio, its user interface will notify you when any unconnected object is found (like datasets uploaded through the REST API without any project reference) and offer to connect such objects to an existing project.

To connect the new dataset to an existing project in the Speech Studio during its upload, use Create Dataset and fill out the request body according to the following format:

```
{
  "kind": "Acoustic",
  "contentUrl": "https://contoso.com/mydatasetlocation",
  "locale": "en-US",
  "displayName": "My speech dataset name",
  "description": "My speech dataset description",
  "project": {
    "self": "https://westeurope.api.cognitive.microsoft.com/speechtotext/v3.0/projects/c1c643ae-7da5-4e38-
9853-e56e840efcb2"
  }
}
```

The Project URL required for the `project` element can be obtained with the Get Projects request.

Audio + human-labeled transcript data for training/testing

Audio + human-labeled transcript data can be used for both training and testing purposes. To improve the acoustic aspects like slight accents, speaking styles, background noises, or to measure the accuracy of Microsoft's speech-to-text accuracy when processing your audio files, you must provide human-labeled transcriptions (word-by-word) for comparison. While human-labeled transcription is often time consuming, it's necessary to evaluate accuracy and to train the model for your use cases. Keep in mind, the improvements in recognition will only be as good as the data provided. For that reason, it's important that only high-quality transcripts are uploaded.

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. While audio with low recording volume or disruptive background noise is not helpful, it should not hurt your custom model. Always consider upgrading your microphones and signal processing hardware before gathering audio samples.

PROPERTY	VALUE
File format	RIFF (WAV)
Sample rate	8,000 Hz or 16,000 Hz
Channels	1 (mono)
Maximum length per audio	2 hours (testing) / 60 s (training)
Sample format	PCM, 16-bit
Archive format	.zip
Maximum zip size	2 GB

The default audio streaming format is WAV (16 kHz or 8 kHz, 16-bit, and mono PCM). Outside of WAV / PCM, the compressed input formats listed below are also supported using GStreamer.

- MP3
- OPUS/OGG
- FLAC
- ALAW in wav container
- MULAW in wav container
- ANY (For the scenario where the media format is not known)

> **NOTE**
> When uploading training and testing data, the .zip file size cannot exceed 2 GB. You can only test from a *single* dataset, be sure to keep it within the appropriate file size. Additionally, each training file cannot exceed 60 seconds otherwise it will error out.

To address issues like word deletion or substitution, a significant amount of data is required to improve recognition. Generally, it's recommended to provide word-by-word transcriptions for 1 to 20 hours of audio. However, even as little as 30 minutes can help to improve recognition results. The transcriptions for all WAV files should be contained in a single plain-text file. Each line of the transcription file should contain the name of one of the audio files, followed by the corresponding transcription. The file name and transcription should be separated by a tab (\t).

For example:

```
speech01.wav speech recognition is awesome
speech02.wav the quick brown fox jumped all over the place
speech03.wav the lazy dog was not amused
```

> **IMPORTANT**
> Transcription should be encoded as UTF-8 byte order mark (BOM).

The transcriptions are text-normalized so they can be processed by the system. However, there are some important normalizations that must be done before uploading the data to the Speech Studio. For the appropriate language to use when you prepare your transcriptions, see How to create a human-labeled transcription

After you've gathered your audio files and corresponding transcriptions, package them as a single .zip file before uploading to the Speech Studio . Below is an example dataset with three audio files and a human-labeled transcription file:

See Set up your Azure account for a list of recommended regions for your Speech service subscriptions. Setting up the Speech subscriptions in one of these regions will reduce the time it takes to train the model. In these regions, training can process about 10 hours of audio per day compared to just 1 hour per day in other regions. If model training cannot be completed within a week, the model will be marked as failed.

Not all base models support training with audio data. If the base model does not support it, the service will ignore the audio and just train with the text of the transcriptions. In this case, training will be the same as training with related text. See Language support for a list of base models that support training with audio data.

Plain text data for training

You can use domain related sentences to improve accuracy when recognizing product names, or industry-specific jargon. Provide sentences in a single text file. To improve accuracy, use text data that is closer to the expected spoken utterances.

Training with plain text usually completes within a few minutes.

To create a custom model using sentences, you'll need to provide a list of sample utterances. Utterances *do not* need to be complete or grammatically correct, but they must accurately reflect the spoken input you expect in production. If you want certain terms to have increased weight, add several sentences that include these specific terms.

As general guidance, model adaptation is most effective when the training text is as close as possible to the real text expected in production. Domain-specific jargon and phrases that you're targeting to enhance, should be included in training text. When possible, try to have one sentence or keyword controlled on a separate line. For keywords and phrases that are important to you (for example, product names), you can copy them a few times. But keep in mind, don't copy too much - it could affect the overall recognition rate.

Use this table to ensure that your related data file for utterances is formatted correctly:

PROPERTY	VALUE
Text encoding	UTF-8 BOM
# of utterances per line	1
Maximum file size	200 MB

Additionally, you'll want to account for the following restrictions:

- Avoid repeating characters, words, or groups of words more than three times. For example: "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`.
- URIs will be rejected.
- For some languages (for example Japanese or Korean), importing large amounts of text data can take very long or time out. Please consider to divide the uploaded data into text files of up to 20.000 lines each.

Structured text data for training (Public Preview)

Often the expected utterances follow a certain pattern. One common pattern is that utterances only differ by words or phrases from a list. Examples of this could be “I have a question about `product`,” where `product` is a list of possible products. Or, “Make that `object` `color`,” where `object` is a list of geometric shapes and `color` is a list of colors. 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 then reference these inside your training utterances. Additionally, the markdown format also supports specifying the phonetic pronunciation of words. The markdown file should have a `.md` extension. 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 is an example of the markdown format:

```
// This is a comment

// 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

@ list sports =
- soccer
- tennis
- cricket
- basketball
- baseball
- football

// This is a list of phonetic pronunciations.
// This adjusts the pronunciation of every instance of these word in both a list or example training
sentences
@ speech:phoneticlexicon
- cat/k ae t
- cat/f i l ai n

// Here are example training sentences. They are grouped into two sections to help organize the example
training sentences.
// You can refer to one of the lists we declared above by using {@listname} and you can refer to multiple
lists in the same training sentence
// A training sentence does not have to refer to a list.
# SomeTrainingSentence
- you can include sentences without a class reference
- what {@pet} do you have
- I like eating {@food} and playing {@sports}
- my {@pet} likes {@food}

# SomeMoreSentence
- you can include more sentences without a class reference
- or more sentences that have a class reference like {@pet}
```

Like plain text, training with structured text typically takes a few minutes. Also, your example sentences and lists should reflect the type of spoken input you expect in production. For pronunciation entries see the description of the Universal Phone Set.

The table below specifies the limits and other properties for the markdown format:

PROPERTY	VALUE
Text encoding	UTF-8 BOM
Maximum file size	200 MB
Maximum number of example sentences	50,000
Maximum number of list classes	10
Maximum number of items in a list class	4,000

PROPERTY	VALUE
Maximum number of speech:phoneticlexicon entries	15000
Maximum number of pronunciations per word	2

Pronunciation data for training

If there are uncommon terms without standard pronunciations that your users will encounter or use, you can provide a custom pronunciation file to improve recognition. For a list of languages that support custom pronunciation, see **Pronunciation** in the **Customizations** column in the Speech-to-text table.

> **IMPORTANT**
> It is not recommended to use custom pronunciation files to alter the pronunciation of common words.

> **NOTE**
> You cannot combine this type of pronunciation file with structured text training data. For structured text data use the phonetic pronunciation capability that is included in the structured text markdown format.

Provide pronunciations in a single text file. This includes examples of a spoken utterance, and a custom pronunciation for each:

RECOGNIZED/DISPLAYED FORM	SPOKEN FORM
3CPO	three c p o
CNTK	c n t k
IEEE	i triple e

The spoken form is the phonetic sequence spelled out. It can be composed of letter, words, syllables, or a combination of all three.

Use the following table to ensure that your related data file for pronunciations is correctly formatted. Pronunciation files are small, and should only be a few kilobytes in size.

PROPERTY	VALUE
Text encoding	UTF-8 BOM (ANSI is also supported for English)
# of pronunciations per line	1
Maximum file size	1 MB (1 KB for free tier)

Audio data for testing

Audio data is optimal for testing the accuracy of Microsoft's baseline speech-to-text model or a custom model. Keep in mind, audio data is used to inspect the accuracy of speech with regard to a specific model's performance. If you want to quantify the accuracy of a model, use audio + human-labeled transcripts.

Custom Speech requires 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
Sample format	PCM, 16-bit
Archive format	.zip
Maximum archive size	2 GB

The default audio streaming format is WAV (16 kHz or 8 kHz, 16-bit, and mono PCM). Outside of WAV / PCM, the compressed input formats listed below are also supported using GStreamer.

- MP3
- OPUS/OGG
- FLAC
- ALAW in wav container
- MULAW in wav container
- ANY (For the scenario where the media format is not known)

> **NOTE**
> When uploading training and testing data, the .zip file size cannot exceed 2 GB. If you require more data for training, divide it into several .zip files and upload them separately. Later, you can choose to train from *multiple* datasets. However, you can only test from a *single* dataset.

Use SoX to verify audio properties or convert existing audio to the appropriate formats. Below are some example SoX commands:

ACTIVITY	SOX COMMAND
Check the audio file format.	`sox --i <filename>`
Convert the audio file to single channel, 16-bit, 16 KHz.	`sox <input> -b 16 -e signed-integer -c 1 -r 16k -t wav <output>.wav`

Next steps

- Inspect your data
- Evaluate your data
- Train custom model
- Deploy model

Inspect Custom Speech data

11/23/2021 • 3 minutes to read • Edit Online

Custom Speech lets you visually inspect the recognition quality of a model in the Speech Studio. You can play back uploaded audio and determine if the provided recognition result is correct. This tool helps you inspect quality of Microsoft's baseline speech-to-text model, inspect a trained custom model, or compare transcription by two models.

This article describes how to visually inspect the quality of Microsoft's baseline speech-to-text model or custom models that you've trained. You'll also see how to use the online transcription editor to create and refine labeled audio datasets.

Prerequisites

You've read Prepare test data for Custom Speech and have uploaded a dataset for inspection.

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, then select **Next**.

6. Choose one or two models to evaluate and compare accuracy.
7. Enter the test name and description, then select **Next**.
8. Review your settings, then select **Save and close**.

After a test has been successfully created, you can see how a model transcribes the audio dataset you specified, or compare results from two models side by side.

> **NOTE**
> When testing, the system will perform a transcription. This is important to keep in mind, as pricing varies per service offering and subscription level. Always refer to the official Azure Cognitive Services pricing - Speech service for the latest details.

Side-by-side model comparisons

When the test status is *Succeeded*, select the test item name to see details of the test. This detail page lists all the utterances in your dataset, and shows the recognition results of the two models you are comparing.

To help inspect the side-by-side comparison, you can toggle various error types including insertion, deletion, and substitution. By listening to the audio and comparing recognition results in each column (showing human-labeled transcription and the results of two speech-to-text models), you can decide which model meets your needs and where improvements are needed.

Side-by-side model testing is useful to validate which speech recognition model is best for an application. For an objective measure of accuracy, requiring transcribed audio, follow the instructions found in Evaluate Accuracy.

Online transcription editor

The online transcription editor allows you to easily work with audio transcriptions in 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 (either audio-only, or audio + transcription).

Import datasets to Editor

To import data into the Editor, first navigate to **Custom Speech > [Your project] > Speech datasets > Editor**

Next, use the following steps to import data.

1. Select **Import data**
2. Create a new dataset(s) and give it a description
3. Select datasets. You can select audio data only, audio + human-labeled data, or both.
4. For audio-only data, you can use the default models to automatically generate machine transcription after

importing to the editor.

5. Select **Import**

After data has been successfully imported, you can select datasets and start editing.

> **TIP**
>
> You can also import datasets into the Editor directly by selecting datasets and selecting **Export to Editor**

Edit transcription by listening to audio

After the data upload has succeeded, select each item name to see details of the data. You can also use **Previous** and **Next** to move between each file.

The detail page lists all the segments in each audio file, and you can select the desired utterance. For each utterance, you can play back the audio and examine the transcripts, and edit the transcriptions if you find any insertion, deletion, or substitution errors. See the data evaluation how-to for more detail on error types.

After you've made edits, select **Save**.

Export datasets from the Editor

To export datasets back to the **Data** tab, navigate to the data detail page and select **Export** to export all the files as a new dataset. You can also filter the files by last edited time, audio durations, etc. to partially select the desired files.

The files exported to Data will be used as a brand-new dataset and will not affect any of the existing data/training/testing entities.

Next steps

- Evaluate your data
- Train your model
- Improve your model
- Deploy your model

Additional resources

- Prepare test data for Custom Speech

Evaluate and improve Custom Speech accuracy

11/23/2021 • 9 minutes to read • Edit Online

In this article, you learn how to quantitatively measure and improve the accuracy of Microsoft's speech-to-text models or your own custom models. Audio + human-labeled transcription data is required to test accuracy, and 30 minutes to 5 hours of representative audio should be provided.

Evaluate Custom Speech accuracy

The industry standard to measure model accuracy is Word Error Rate (WER). WER counts the number of incorrect words identified during recognition, then divides by the total number of words provided in the human-labeled transcript (shown below as N). Finally, that number is multiplied by 100% to calculate the WER.

$$WER = \frac{I + D + S}{N} * 100\%$$

Incorrectly identified words fall into three categories:

- Insertion (I): Words that are incorrectly added in the hypothesis transcript
- Deletion (D): Words that are undetected in the hypothesis transcript
- Substitution (S): Words that were substituted between reference and hypothesis

Here's an example:

If you want to replicate WER measurements locally, you can use sclite from SCTK.

Resolve errors and improve WER

You can use the WER from the machine recognition results to evaluate the quality of the model you are using with your app, tool, or product. A WER of 5%-10% is considered to be good quality and is ready to use. A WER of 20% is acceptable, however you may want to consider additional training. A WER of 30% or more signals poor quality and requires customization and training.

How the errors are distributed is important. When many deletion errors are encountered, it's usually because of weak audio signal strength. To resolve this issue, you'll need to collect audio data closer to the source. Insertion errors mean that the audio was recorded in a noisy environment and crosstalk may be present, causing recognition issues. Substitution errors are often encountered when an insufficient sample of domain-specific terms has been provided as either human-labeled transcriptions or related text.

By analyzing individual files, you can determine what type of errors exist, and which errors are unique to a specific file. Understanding issues at the file level will help you target improvements.

Create a test

If you'd like to test the quality of Microsoft's speech-to-text baseline model or a custom model that you've trained, you can compare two models side by side to evaluate accuracy. The comparison includes WER and recognition results. Typically, a custom model is compared with Microsoft's baseline model.

To evaluate models side by side:

1. Sign in to the Custom Speech portal.
2. Navigate to **Speech-to-text > Custom Speech > [name of project] > Testing**.
3. Click **Add Test**.
4. Select **Evaluate accuracy**. Give the test a name, description, and select your audio + human-labeled transcription dataset.
5. Select up to two models that you'd like to test.
6. Click **Create**.

After your test has been successfully created, you can compare the results side by side.

Side-by-side comparison

Once the test is complete, indicated by the status change to *Succeeded*, you'll find a WER number for both models included in your test. Click on the test name to view the testing detail page. This detail page lists all the utterances in your dataset, indicating the recognition results of the two models alongside the transcription from the submitted dataset. To help inspect the side-by-side comparison, you can toggle various error types including insertion, deletion, and substitution. By listening to the audio and comparing recognition results in each column, which shows the human-labeled transcription and the results for two speech-to-text models, you can decide which model meets your needs and where additional training and improvements are required.

Improve Custom Speech accuracy

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 2 humans on 1 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 word error rate (WER). The table shows which error types are most common in each scenario.

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
Video closed captioning	Depends on video type (can be < 50% WER)	Low	Can be high due to music, noises, microphone quality	Jargon may cause these errors

Determining the components of the WER (number of insertion, deletion, and substitution errors) helps determine what kind of data to add to improve the model. Use the Custom Speech portal to view the quality of a baseline model. The portal reports insertion, substitution, and deletion error rates that are combined in the WER quality rate.

Improve model recognition

You can reduce recognition errors by adding training data in the Custom Speech portal.

Plan to maintain your custom model by adding source materials periodically. Your custom model needs additional training to stay aware of changes to your entities. For example, you may need updates to product names, song names, or new service locations.

The following sections describe how each kind of additional training data can reduce errors.

Add plain text data

When you train a new custom model, start by adding plain text sentences of related text to improve the recognition of domain-specific words and phrases. Related text sentences can primarily 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.

> **NOTE**
> Avoid related text sentences that include noise such as unrecognizable characters or words.

Add structured text data

You can use structured text data in markdown format similarly to plain text sentences, but you would use structured text data when your data follows a particular pattern in particular utterances that only differ by words or phrases from a list. See Structured text data for training for more information.

> **NOTE**
>
> Training with structured text is only supported for these locales: `en-US`, `de-DE`, `en-UK`, `en-IN`, `fr-FR`, `fr-CA`, `es-ES`, `es-MX` and you must use the latest base model for these locales. See Language support for a list of base models that support training with structured text data.
>
> 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.

Add audio with human-labeled transcripts

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 most calls about swimwear and sunglasses during summer months. Assure that your sample includes the full scope of speech 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 just using related text.
- If you use one of the most heavily used languages such as US-English, there's a good chance that there's no need to train with audio data. For such languages, the base models offer already very good recognition results in most scenarios; it's probably enough to train with related text.
- Custom Speech can only capture word context 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 not related to your problem domain. Unrelated sentences can harm your model.
- When the quality of transcripts vary, you can duplicate exceptionally good sentences (like excellent transcriptions that include key phrases) to increase their weight.
- The Speech service will automatically use the transcripts to improve the recognition of domain-specific words and phrases, as if they were added as related text.
- It can take several days for a training operation to complete. To improve the speed of training, make sure to create your Speech service subscription in a region with the dedicated hardware for training.

> **NOTE**
>
> Not all base models support training with audio. If a base model does not support it, the Speech service will only use the text from the transcripts and ignore the audio. See Language support for a list of base models that support training with audio data. Even if a base model supports training with audio data, the service might use only part of the audio. Still it will use all the transcripts.

> **NOTE**
>
> In cases when you change the base model used for training, and you have audio in the training dataset, *always* check whether the new selected base model supports training with audio data. If the previously used base model did not support training with audio data, and the training dataset contains audio, training time with the new base model will **drastically** increase, and may easily go from several hours to several days and more. This is especially true if your Speech service subscription is **not** in a region with the dedicated hardware for training.
>
> If you face the issue described in the paragraph above, you can quickly decrease the training time by reducing the amount of audio in the dataset or removing it completely and leaving only the text. The latter option is highly recommended if your Speech service subscription is **not** in a region with the dedicated hardware for training.

Add new words with pronunciation

Words that are made-up or highly specialized may have unique pronunciations. These words can be recognized if the word can be broken down into smaller words to pronounce it. For example, to recognize **Xbox**, pronounce as **X box**. This approach will not increase overall accuracy, but can increase recognition of these keywords.

> **NOTE**
>
> This technique is only available for some languages at this time. See customization for pronunciation in the Speech-to-text table for details.

Sources by scenario

The following table shows voice recognition scenarios and lists source materials to consider within the three training content categories listed above.

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 Xbox above)
Voice assistant	list sentences using all combinations of commands and entities	record voices speaking commands into device, and transcribe into text	names (movies, songs, products) that have unique pronunciations
Dictation	written input, like instant messages or emails	similar to above	similar to above
Video closed captioning	TV show scripts, movies, marketing content, video summaries	exact transcripts of videos	similar to above

Next steps

- Train and deploy a model

Additional resources

- Prepare and test your data
- Inspect your data

Train and deploy a Custom Speech model

11/23/2021 • 5 minutes to read • Edit Online

In this article, you'll learn how to train and deploy Custom Speech models. Training a speech-to-text model can improve recognition accuracy for the Microsoft baseline model. You use human-labeled transcriptions and related text to train a model. These datasets, along with previously uploaded audio data, are used to refine and train the speech-to-text model.

Use training to resolve accuracy problems

If you're encountering recognition problems with a base model, you can use human-labeled transcripts and related data to train a custom model and help improve accuracy. Use this table to determine which dataset to use to address your problems:

USE CASE	DATA TYPE
Improve recognition accuracy on industry-specific vocabulary and grammar, like 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, like 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

Train and evaluate a model

The first step to train a model is to upload training data. See Prepare and test your data for step-by-step instructions to prepare human-labeled transcriptions and related text (utterances and pronunciations). After you upload training data, follow these instructions to start training your model:

1. Sign in to the Custom Speech portal. If you plan to train a model with audio + human-labeled transcription datasets, pick a Speech subscription in a region with dedicated hardware for training.
2. Go to **Speech-to-text** > **Custom Speech** > **[name of project]** > **Training**.
3. Select **Train model**.
4. Give your training a **Name** and **Description**.
5. In the **Scenario and Baseline model** list, select the scenario that best fits your domain. If you're not sure which scenario to choose, select **General**. The baseline model is the starting point for training. The latest model is usually the best choice.
6. On the **Select training data** page, choose one or more related text datasets or audio + human-labeled transcription datasets that you want to use for training.

> **NOTE**
>
> When you train a new model, start with related text; training with audio + human-labeled transcription might take much longer (**up to several days**).

> **NOTE**
>
> Not all base models support training with audio. If a base model does not support it, the Speech service will only use the text from the transcripts and ignore the audio. See Language support for a list of base models that support training with audio data.

> **NOTE**
>
> In cases when you change the base model used for training, and you have audio in the training dataset, *always* check whether the new selected base model supports training with audio data. If the previously used base model did not support training with audio data, and the training dataset contains audio, training time with the new base model will **drastically** increase, and may easily go from several hours to several days and more. This is especially true if your Speech service subscription is **not** in a region with the dedicated hardware for training.
>
> If you face the issue described in the paragraph above, you can quickly decrease the training time by reducing the amount of audio in the dataset or removing it completely and leaving only the text. The latter option is highly recommended if your Speech service subscription is **not** in a region with the dedicated hardware for training.

7. After training is complete, you can do accuracy testing on the newly trained model. This step is optional.
8. Select **Create** to build your custom model.

The **Training** table displays a new entry that corresponds to the new model. The table also displays the status: **Processing**, **Succeeded**, or **Failed**.

See the how-to on evaluating and improving Custom Speech model accuracy. If you choose to test accuracy, it's important to select an acoustic dataset that's different from the one you used with your model to get a realistic sense of the model's performance.

> **NOTE**
>
> Both base models and custom models can be used only up to a certain date (see Model and endpoint lifecycle). Speech Studio shows this date in the **Expiration** column for each model and endpoint. After that date request to an endpoint or to batch transcription might fail or fall back to base model.
>
> Retrain your model using the then most recent base model to benefit from accuracy improvements and to avoid that your model expires.

Deploy a custom model

After you upload and inspect data, evaluate accuracy, and train a custom model, you can deploy a custom endpoint to use with your apps, tools, and products.

To create a custom endpoint, sign in to the Custom Speech portal. Select **Deployment** in the **Custom Speech** menu at the top of the page. If this is your first run, you'll notice that there are no endpoints listed in the table. After you create an endpoint, you use this page to track each deployed endpoint.

Next, select **Add endpoint** and enter a **Name** and **Description** for your custom endpoint. Then select the custom model that you want to associate with the endpoint. You can also enable logging from this page. Logging allows you to monitor endpoint traffic. If logging is disabled, traffic won't be stored.

> **NOTE**
>
> Don't forget to accept the terms of use and pricing details.

Next, select **Create**. This action returns you to the **Deployment** page. The table now includes an entry that corresponds to your custom endpoint. The endpoint's status shows its current state. It can take up to 30 minutes to instantiate a new endpoint using your custom models. When the status of the deployment changes to **Complete**, the endpoint is ready to use.

After your endpoint is deployed, the endpoint name appears as a link. Select the link to see information specific to your endpoint, like the endpoint key, endpoint URL, and sample code. Take a note of the expiration date and update the endpoint's model before that date to ensure uninterrupted service.

View logging data

Logging data is available for export if you go to the endpoint's page under **Deployments**.

> **NOTE**
>
> Logging data is available for 30 days on Microsoft-owned storage. It will be removed afterwards. If a customer-owned storage account is linked to the Cognitive Services subscription, the logging data won't be automatically deleted.

Next steps

- Learn how to use your custom model

Additional resources

- Prepare and test your data
- Inspect your data
- Evaluate your data

CI/CD for Custom Speech

11/23/2021 • 4 minutes to read • Edit Online

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

Learn more about DevOps with Speech:

- Use the Speech DevOps template repo to implement DevOps for Custom Speech with GitHub Actions.

Customize pronunciation

11/23/2021 • 2 minutes to read • Edit Online

Custom speech allows you to provide different pronunciations for specific words using the Universal Phone Set. The Universal Phone Set (UPS) is a machine-readable phone set that is based on the International Phonetic Set Alphabet (IPA). The IPA is used by linguists world-wide and is accepted as a standard.

UPS pronunciations consist of a string of UPS phones, each separated by whitespace. The phone set is case-sensitive. UPS phone labels are all defined using ASCII character strings.

For steps on implementing UPS, see Structured text data for training phone sets

This structured text data is not the same as pronunciation files, and they cannot be used together.

Languages Supported

Use the table below to navigate to the UPS for the respective language.

LANGUAGE	LOCALE
English (United States)	`en-US`

Next steps

- Provide UPS pronunciation to Custom Speech

How to create human-labeled transcriptions

11/23/2021 • 7 minutes to read • Edit Online

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.

A large sample of transcription data is required to improve recognition. We suggest providing between 1 and 20 hours of transcription data. The Speech service will use up to 20 hours of audio for training. On this page, we'll review guidelines designed to help you create high-quality transcriptions. This guide is broken up by locale, with sections for US English, Mandarin Chinese, and German.

> **NOTE**
>
> Not all base models support customization with audio files. If a base model does not support it, training will just use the text of the transcriptions in the same way as related text is used. See Language support for a list of base models that support training with audio data.

> **NOTE**
>
> In cases when you change the base model used for training, and you have audio in the training dataset, *always* check whether the new selected base model supports training with audio data. If the previously used base model did not support training with audio data, and the training dataset contains audio, training time with the new base model will **drastically** increase, and may easily go from several hours to several days and more. This is especially true if your Speech service subscription is **not** in a region with the dedicated hardware for training.
>
> If you face the issue described in the paragraph above, you can quickly decrease the training time by reducing the amount of audio in the dataset or removing it completely and leaving only the text. The latter option is highly recommended if your Speech service subscription is **not** in a region with the dedicated hardware for training.

US English (en-US)

Human-labeled transcriptions for English audio must be provided as plain text, only using ASCII characters. Avoid the use of Latin-1 or Unicode punctuation characters. These characters are often inadvertently added when copying text from a word-processing application or scraping data from web pages. If these characters are present, make sure to update them with the appropriate ASCII substitution.

Here are a few examples:

CHARACTERS TO AVOID	SUBSTITUTION	NOTES
“Hello world”	"Hello world"	The opening and closing quotations marks have been substituted with appropriate ASCII characters.
John’s day	John's day	The apostrophe has been substituted with the appropriate ASCII character.
It was good—no, it was great!	it was good--no, it was great!	The em dash was substituted with two hyphens.

Text normalization for US English

Text normalization is the transformation of words into a consistent format used when training a model. Some normalization rules are applied to text automatically, however, we recommend using these guidelines as you prepare your human-labeled transcription data:

- Write out abbreviations in words.
- Write out non-standard numeric strings in words (such as accounting terms).
- Non-alphabetic characters or mixed alphanumeric characters should be transcribed as pronounced.
- Abbreviations that are pronounced as words shouldn't be edited (such as "radar", "laser", "RAM", or "NATO").
- Write out abbreviations that are pronounced as separate letters with each letter separated by a space.
- 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
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

ORIGINAL TEXT	TEXT AFTER NORMALIZATION (AUTOMATIC)
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

Mandarin Chinese (zh-CN)

Human-labeled transcriptions for Mandarin Chinese audio must be UTF-8 encoded with a byte-order marker. Avoid the use of half-width punctuation characters. These characters can be included inadvertently when you prepare the data in a word-processing program or scrape data from web pages. If these characters are present, make sure to update them with the appropriate full-width substitution.

Here are a few examples:

CHARACTERS TO AVOID	SUBSTITUTION	NOTES
"你好"	"你好"	The opening and closing quotations marks have been substituted with appropriate characters.
需要什么帮助?	需要什么帮助？	The question mark has been substituted with appropriate character.

Text normalization for Mandarin Chinese

Text normalization is the transformation of words into a consistent format used when training a model. Some normalization rules are applied to text automatically, however, we recommend using these guidelines as you prepare your human-labeled transcription data:

- Write out abbreviations in words.
- Write out numeric strings in spoken form.

Here are a few examples of normalization that you should perform on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
我今年 21	我今年二十一
3 号楼 504	三号 楼 五 零 四

The following normalization rules are automatically applied to transcriptions:

- Remove all punctuation
- Expand numbers to spoken form
- Convert full-width letters to half-width letters
- Using uppercase letters for all English words

Here are 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

German (de-DE) and other languages

Human-labeled transcriptions for German audio (and other non-English or Mandarin Chinese languages) must be UTF-8 encoded with a byte-order marker. One human-labeled transcript should be provided for each audio file.

Text normalization for German

Text normalization is the transformation of words into a consistent format used when training a model. Some normalization rules are applied to text automatically, however, we recommend using these guidelines as you prepare your human-labeled transcription data:

- Write decimal points as "," and not ".".
- Write time separators as ":" and not "." (for example: 12:00 Uhr).
- Abbreviations such as "ca." aren't replaced. We recommend that you use the full spoken form.
- The four main mathematical operators (+, -, *, and /) are removed. We recommend replacing them with the written form: "plus," "minus," "mal," and "geteilt."
- Comparison operators are removed (=, <, and >). We recommend replacing them with "gleich," "kleiner als," and "grösser als."
- Write fractions, such as 3/4, in written form (for example: "drei viertel" instead of 3/4).
- Replace the "€" symbol with its written form "Euro."

Here are a few examples of normalization that you should perform on the transcription:

ORIGINAL TEXT	TEXT AFTER USER NORMALIZATION	TEXT AFTER SYSTEM NORMALIZATION
Es ist 12.23 Uhr	Es ist 12:23 Uhr	es ist zwölf uhr drei und zwanzig uhr
{12.45}	{12,45}	zwölf komma vier fünf
2 + 3 - 4	2 plus 3 minus 4	zwei plus drei minus vier

The following normalization rules are automatically applied to transcriptions:

- Use lowercase letters for all text.
- Remove all punctuation, including various types of quotation marks ("test", 'test', "test„, and «test» are OK).
- Discard rows with any special characters from this set: ¢ ¤ ¥ ¦ § © ª ¬ ® ° ± ² µ × ÿ Ø ¬ ¬.

- Expand numbers to spoken form, including dollar or Euro amounts.
- Accept umlauts only for a, o, and u. Others will be replaced by "th" or be discarded.

Here are a few examples of normalization automatically performed on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
Frankfurter Ring	frankfurter ring
¡Eine Frage!	eine frage
Wir, haben	wir haben

Text normalization for Japanese

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.

Next Steps

- Prepare and test your data
- Inspect your data
- Evaluate your data
- Train your model
- Deploy your model

Model and endpoint lifecycle

11/23/2021 • 4 minutes to read • Edit Online

Our standard (not customized) speech is built upon AI models that we call base models. In most cases, we train a different base model for each spoken language we support. We update the speech service with new base models every few months to improve accuracy and quality.
With Custom Speech, custom models are created by adapting a chosen base model with data from your particular customer scenario. Once you create a custom model, that model will not be updated or changed, even if the corresponding base model from which it was adapted gets updated in the standard speech service.
This policy allows you to keep using a particular custom model for a long time after you have a custom model that meets your needs. But we recommend that you periodically recreate your custom model so you can adapt from the latest base model to take advantage of the improved accuracy and quality.

Other key terms related to the model lifecycle include:

- **Adaptation**: Taking a base model and customizing it to your domain/scenario by using text data and/or audio data.
- **Decoding**: Using a model and performing speech recognition (decoding audio into text).
- **Endpoint**: A user-specific deployment of either a base model or a custom model that's accessible *only* to a given user.

Expiration timeline

As new models and new functionality become available and older, less accurate models are retired, see the following timelines for model and endpoint expiration:

Base models

- Adaptation: Available for one year. After the model is imported, it's available for one year to create custom models. After one year, new custom models must be created from a newer base model version.
- Decoding: Available for two years after import. So you can create an endpoint and use batch transcription for two years with this model.
- Endpoints: Available on the same timeline as decoding.

Custom models

- Decoding: Available for two years after the model is created. So you can use the custom model for two years (batch/realtime/testing) after it's created. After two years, *you should retrain your model* because the base model will usually have been deprecated for adaptation.
- Endpoints: Available on the same timeline as decoding.

When either a base model or custom model expires, it will always fall back to the *newest base model version*. So your implementation will never break, but it might become less accurate for *your specific data* if custom models reach expiration. You can see the expiration for a model in the following places in the Custom Speech area of the Speech Studio:

- Model training summary
- Model training detail
- Deployment summary
- Deployment detail

Here is an example form the model training summary:

And also from the model training detail page:

You can also check the expiration dates via the `GetModel` and `GetBaseModel` custom speech APIs under the `deprecationDates` property in the JSON response.

Here is an example of the expiration data from the GetModel API call. The **DEPRECATIONDATES** show when the model expires:

```
{
    "SELF": "HTTPS://WESTUS2.API.COGNITIVE.MICROSOFT.COM/SPEECHTOTEXT/V3.0/MODELS/{id}",
    "BASEMODEL": {
    "SELF": HTTPS://WESTUS2.API.COGNITIVE.MICROSOFT.COM/SPEECHTOTEXT/V3.0/MODELS/BASE/{id}
    },
    "DATASETS": [
    {
        "SELF": https://westus2.api.cognitive.microsoft.com/speechtotext/v3.0/datasets/{id}
    }
    ],
    "LINKS": {
    "MANIFEST": "HTTPS://WESTUS2.API.COGNITIVE.MICROSOFT.COM/SPEECHTOTEXT/V3.0/MODELS/{id}/MANIFEST",
    "COPYTO": https://westus2.api.cognitive.microsoft.com/speechtotext/v3.0/models/{id}/copyto
    },
    "PROJECT": {
        "SELF": https://westus2.api.cognitive.microsoft.com/speechtotext/v3.0/projects/{id}
    },
    "PROPERTIES": {
    "DEPRECATIONDATES": {
        "ADAPTATIONDATETIME": "2022-01-15T00:00:00Z",     // last date the base model can be used for
adaptation
        "TRANSCRIPTIONDATETIME": "2023-03-01T21:27:29Z"   // last date this model can be used for decoding
    }
    },
    "LASTACTIONDATETIME": "2021-03-01T21:27:40Z",
    "STATUS": "SUCCEEDED",
    "CREATEDDATETIME": "2021-03-01T21:27:29Z",
    "LOCALE": "EN-US",
    "DISPLAYNAME": "EXAMPLE MODEL",
    "DESCRIPTION": "",
    "CUSTOMPROPERTIES": {
    "PORTALAPIVERSION": "3"
    }
}
```

Note that you can upgrade the model on a custom speech endpoint without downtime by changing the model used by the endpoint in the deployment section of the Speech Studio, or via the custom speech API.

What happens when models expire and how to update them

What happens when a model expires and how to update the model depends on how it is being used.

Batch transcription

If a model expires that is used with batch transcription transcription requests will fail with a 4xx error. To prevent this update the `model` parameter in the JSON sent in the **Create Transcription** request body to either point to a more recent base model or more recent custom model. You can also remove the `model` entry from the JSON to always use the latest base model.

Custom speech endpoint

If a model expires that is used by a custom speech endpoint, then the service will automatically fall back to using the latest base model for the language you are using. To update a model you are using, you can select **Deployment** in the **Custom Speech** menu at the top of the page and then click on the endpoint name to see its details. At the top of the details page, you will see an **Update Model** button that lets you seamlessly update the model used by this endpoint without downtime. You can also make this change programmatically by using the **Update Model** Rest API.

Next steps

- Train and deploy a model

Additional resources

- Prepare and test your data
- Inspect your data

Tutorial: Create a tenant model (preview)

11/23/2021 • 6 minutes to read • Edit Online

Tenant Model (Custom Speech with Microsoft 365 data) is an opt-in service for Microsoft 365 enterprise customers that automatically generates a custom speech recognition model from your organization's Microsoft 365 data. The model is optimized for technical terms, jargon, and people's names, all in a secure and compliant way.

> **IMPORTANT**
>
> If your organization enrolls by using the Tenant Model service, Speech Service may access your organization's language model. The model is generated from Microsoft 365 public group emails and documents, which can be seen by anyone in your organization. Your organization's admin can turn on or turn off the use of the organization-wide language model from the admin portal.

In this tutorial, you learn how to:

- Enroll in the Tenant Model by using the Microsoft 365 admin center
- Get a Speech subscription key
- Create a tenant model
- Deploy a tenant model
- Use your tenant model with the Speech SDK

Enroll in the Tenant Model service

Before you can deploy your tenant model, you need to be enrolled in the Tenant Model service. Enrollment is completed in the Microsoft 365 admin center and can be done only by your admin.

1. Sign in to the Microsoft 365 admin center.
2. In the left pane, select **Settings**, then select **Settings** from the nested menu, then select **Azure Speech Services** from the main window.

3. Select the **Allow the organization-wide language model** check box, and then select **Save changes**.

To turn off the tenant model instance:

1. Repeat the preceding steps 1 and 2.
2. Clear the **Allow the organization-wide language model** check box, and then select **Save changes**.

Get a Speech subscription key

To use your tenant model with the Speech SDK, you need a Speech resource and its associated subscription key.

1. Sign in to the Azure portal.
2. Select **Create a resource**.
3. In the **Search** box, type **Speech**.
4. In the results list, select **Speech**, and then select **Create**.
5. Follow the onscreen instructions to create your resource. Make sure that:
   - **Location** is set to either **eastus** or **westus**.
   - **Pricing tier** is set to **S0**.
6. Select **Create**.

   After a few minutes, your resource is created. The subscription key is available in the **Overview** section for your resource.

Create a language model

After your admin has enabled Tenant Model for your organization, you can create a language model that's based on your Microsoft 365 data.

1. Sign in to Speech Studio.
2. At the top right, select **Settings** (gear icon), and then select **Tenant Model settings**.

Speech Studio displays a message that lets you know whether you're qualified to create a tenant model.

> **NOTE**
>
> Enterprise customers in North America are eligible to create a tenant model (English). If you're a Customer Lockbox, Customer Key, or Office 365 Government customer, this feature isn't available. To determine whether you're a Customer Lockbox or Customer Key customer, see:
>
> - Customer Lockbox
> - Customer Key
> - Office 365 Government

3. Select **Opt in**.

   When your tenant model is ready, you'll receive a confirmation email message with further instructions.

Deploy your tenant model

When your tenant model instance is ready, deploy it by doing the following:

1. In your confirmation email message, select the **View model** button. Or sign in to Speech Studio.
2. At the top right, select **Settings** (gear icon), and then select **Tenant Model settings**.

3. Select **Deploy**.

   When your model has been deployed, the status changes to *Deployed*.

Use your tenant model with the Speech SDK

Now that you've deployed your model, you can use it with the Speech SDK. In this section, you use sample code to call Speech Service by using Azure Active Directory (Azure AD) authentication.

Let's look at the code you'll use to call the Speech SDK in C#. In this example, you perform speech recognition by using your tenant model. This guide presumes that your platform is already set up. If you need setup help, see Quickstart: Recognize speech, C# (.NET Core).

Copy this code into your project:

```
namespace PrincetonSROnly.FrontEnd.Samples
{
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Net.Http;
    using System.Text;
    using System.Text.RegularExpressions;
    using System.Threading.Tasks;
    using Microsoft.CognitiveServices.Speech;
    using Microsoft.CognitiveServices.Speech.Audio;
    using Microsoft.IdentityModel.Clients.ActiveDirectory;
    using Newtonsoft.Json.Linq;

    // ServiceApplicationId is a fixed value. No need to change it.

    public class TenantLMSample
    {
        private const string EndpointUriArgName = "EndpointUri";
```

```
        private const string SubscriptionKeyArgName = "SubscriptionKey";
        private const string UsernameArgName = "Username";
        private const string PasswordArgName = "Password";
        private const string ClientApplicationId = "f87bc118-1576-4097-93c9-dbf8f45ef0dd";
        private const string ServiceApplicationId = "18301695-f99d-4cae-9618-6901d4bdc7be";

        public static async Task ContinuousRecognitionWithTenantLMAsync(Uri endpointUri, string
subscriptionKey, string audioDirPath, string username, string password)
        {
            var config = SpeechConfig.FromEndpoint(endpointUri, subscriptionKey);

            // Passing client specific information for obtaining LM
            if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
            {
                config.AuthorizationToken = await
AcquireAuthTokenWithInteractiveLoginAsync().ConfigureAwait(false);
            }
            else
            {
                config.AuthorizationToken = await AcquireAuthTokenWithUsernamePasswordAsync(username,
password).ConfigureAwait(false);
            }

            var stopRecognition = new TaskCompletionSource<int>();

            // Creates a speech recognizer using file as audio input.
            // Replace with your own audio file name.
            using (var audioInput = AudioConfig.FromWavFileInput(audioDirPath))
            {
                using (var recognizer = new SpeechRecognizer(config, audioInput))
                {
                    // Subscribes to events
                    recognizer.Recognizing += (s, e) =>
                    {
                        Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
                    };

                    recognizer.Recognized += (s, e) =>
                    {
                        if (e.Result.Reason == ResultReason.RecognizedSpeech)
                        {
                            Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                        }
                        else if (e.Result.Reason == ResultReason.NoMatch)
                        {
                            Console.WriteLine($"NOMATCH: Speech could not be recognized.");
                        }
                    };

                    recognizer.Canceled += (s, e) =>
                    {
                        Console.WriteLine($"CANCELED: Reason={e.Reason}");
                        if (e.Reason == CancellationReason.Error)
                        {
                            Exception exp = new Exception(string.Format("Error Code: {0}\nError
Details{1}\nIs your subscription information updated?", e.ErrorCode, e.ErrorDetails));
                            throw exp;
                        }

                        stopRecognition.TrySetResult(0);
                    };

                    recognizer.SessionStarted += (s, e) =>
                    {
                        Console.WriteLine("\n    Session started event.");
                    };

                    recognizer.SessionStopped += (s, e) =>
                    {
```

```
                        Console.WriteLine("\n    Session stopped event.");
                        Console.WriteLine("\nStop recognition.");
                        stopRecognition.TrySetResult(0);
                    };

                    // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop
recognition.
                    await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

                    // Waits for completion.
                    // Use Task.WaitAny to keep the task rooted.
                    Task.WaitAny(new[] { stopRecognition.Task });

                    // Stops recognition.
                    await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
                }
            }
        }

        public static void Main(string[] args)
        {
            var arguments = new Dictionary<string, string>();
            string inputArgNamePattern = "--";
            Regex regex = new Regex(inputArgNamePattern);
            if (args.Length > 0)
            {
                foreach (var arg in args)
                {
                    var userArgs = arg.Split("=");
                    arguments[regex.Replace(userArgs[0], string.Empty)] = userArgs[1];
                }
            }

            var endpointString = arguments.GetValueOrDefault(EndpointUriArgName,
$"wss://westus.online.princeton.customspeech.ai/msgraphcustomspeech/conversation/v1");
            var endpointUri = new Uri(endpointString);

            if (!arguments.ContainsKey(SubscriptionKeyArgName))
            {
                Exception exp = new Exception("Subscription Key missing! Please pass in a Cognitive services
subscription Key using --SubscriptionKey=\"your_subscription_key\"" +
                    "Find more information on creating a Cognitive services resource and accessing your
Subscription key here: https://docs.microsoft.com/azure/cognitive-services/cognitive-services-apis-create-
account?tabs=multiservice%2Cwindows");
                throw exp;
            }

            var subscriptionKey = arguments[SubscriptionKeyArgName];
            var username = arguments.GetValueOrDefault(UsernameArgName, null);
            var password = arguments.GetValueOrDefault(PasswordArgName, null);

            var audioDirPath =
Path.Combine(Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location),
"../../../AudioSamples/DictationBatman.wav");
            if (!File.Exists(audioDirPath))
            {
                Exception exp = new Exception(string.Format("Audio File does not exist at path: {0}",
audioDirPath));
                throw exp;
            }

            ContinuousRecognitionWithTenantLMAsync(endpointUri, subscriptionKey, audioDirPath, username,
password).GetAwaiter().GetResult();
        }

        private static async Task<string> AcquireAuthTokenWithUsernamePasswordAsync(string username, string
password)
        {
            var tokenEndpoint = "https://login.microsoftonline.com/common/oauth2/token";
```

```
            var postBody = $"resource={ServiceApplicationId}&client_id=
{ClientApplicationId}&grant_type=password&username={username}&password={password}";
            var stringContent = new StringContent(postBody, Encoding.UTF8, "application/x-www-form-
urlencoded");
            using (HttpClient httpClient = new HttpClient())
            {
                var response = await httpClient.PostAsync(tokenEndpoint,
stringContent).ConfigureAwait(false);

                if (response.IsSuccessStatusCode)
                {
                    var result = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

                    JObject jobject = JObject.Parse(result);
                    return jobject["access_token"].Value<string>();
                }
                else
                {
                    throw new Exception($"Requesting token from {tokenEndpoint} failed with status code
{response.StatusCode}: {await response.Content.ReadAsStringAsync().ConfigureAwait(false)}");
                }
            }
        }

        private static async Task<string> AcquireAuthTokenWithInteractiveLoginAsync()
        {
            var authContext = new
AuthenticationContext("https://login.windows.net/microsoft.onmicrosoft.com");
            var deviceCodeResult = await authContext.AcquireDeviceCodeAsync(ServiceApplicationId,
ClientApplicationId).ConfigureAwait(false);

            Console.WriteLine(deviceCodeResult.Message);

            var authResult = await
authContext.AcquireTokenByDeviceCodeAsync(deviceCodeResult).ConfigureAwait(false);

            return authResult.AccessToken;
        }
    }
}
```

Next, you need to rebuild and run the project from the command line. Before you run the command, update a few parameters by doing the following:

1. Replace `<Username>` and `<Password>` with the values for a valid tenant user.
2. Replace `<Subscription-Key>` with the subscription key for your Speech resource. This value is available in the **Overview** section for your Speech resource in the Azure portal.
3. Replace `<Endpoint-Uri>` with the following endpoint. Make sure that you replace `{your region}` with the region where your Speech resource was created. These regions are supported: `westus`, `westus2`, and `eastus`. Your region information is available in the **Overview** section of your Speech resource in the Azure portal.

   ```
   "wss://{your region}.online.princeton.customspeech.ai/msgraphcustomspeech/conversation/v1".
   ```

4. Run the following command:

   ```
   dotnet TenantLMSample.dll --Username=<Username> --Password=<Password> --SubscriptionKey=
   <Subscription-Key> --EndpointUri=<Endpoint-Uri>
   ```

In this tutorial, you've learned how to use Microsoft 365 data to create a custom speech recognition model,

deploy it, and use it with the Speech SDK.

Next steps

- Speech Studio
- Speech SDK

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

Speech-to-text REST API

11/23/2021 • 15 minutes to read • Edit Online

Speech-to-text has two different REST APIs. Each API serves its special purpose and uses different sets of endpoints.

The Speech-to-text REST APIs are:

- Speech-to-text REST API v3.0 is used for Batch transcription and Custom Speech. v3.0 is a successor of v2.0.
- Speech-to-text REST API for short audio is used for online transcription as an alternative to the Speech SDK. Requests using this API can transmit only up to 60 seconds of audio per request.

Speech-to-text REST API v3.0

Speech-to-text REST API v3.0 is used for Batch transcription and Custom Speech. If you need to communicate with the online transcription via REST, use Speech-to-text REST API for short audio.

Use REST API v3.0 to:

- Copy models to other subscriptions in case you want colleagues to have access to a model you built, or in cases where you want to deploy a model to more than one region
- Transcribe data from a container (bulk transcription) as well as provide multiple audio file URLs
- Upload data from Azure Storage accounts through the use of a SAS Uri
- Get logs per endpoint if logs have been requested for that endpoint
- Request the manifest of the models you create, for the purpose of setting up on-premises containers

REST API v3.0 includes such features as:

- **Notifications-Webhooks**—All running processes of the service now support webhook notifications. REST API v3.0 provides the calls to enable you to register your webhooks where notifications are sent
- **Updating models behind endpoints**
- **Model adaptation with multiple data sets**—Adapt a model using multiple data set combinations of acoustic, language, and pronunciation data
- **Bring your own storage**—Use your own storage accounts for logs, transcription files, and other data

See examples on using REST API v3.0 with the Batch transcription is this article.

If you are using Speech-to-text REST API v2.0, see how you can migrate to v3.0 in this guide.

See the full Speech-to-text REST API v3.0 Reference here.

Speech-to-text REST API for short audio

As an alternative to the Speech SDK, the Speech service allows you to convert Speech-to-text using a REST API. The REST API for short audio is very limited, and it should only be used in cases were the Speech SDK cannot.

Before using the Speech-to-text REST API for short audio, consider the following:

- Requests that use the REST API for short audio and transmit audio directly can only contain up to 60 seconds of audio.
- The Speech-to-text REST API for short audio only returns final results. Partial results are not provided.

If sending longer audio is a requirement for your application, consider using the Speech SDK or Speech-to-text

REST API v3.0.

> **TIP**
> See this article for Azure Government and Azure China endpoints.

Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each service:

SUPPORTED AUTHORIZATION HEADERS	SPEECH-TO-TEXT	TEXT-TO-SPEECH
Ocp-Apim-Subscription-Key	Yes	Yes
Authorization: Bearer	Yes	Yes

When using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your subscription key. For example:

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your subscription key for an access token that's valid for 10 minutes. In the next few sections you'll learn how to get a token, and use a token.

How to get an access token

To get an access token, you'll need to make a request to the `issueToken` endpoint using the `Ocp-Apim-Subscription-Key` and your subscription key.

The `issueToken` endpoint has this format:

```
https://<REGION_IDENTIFIER>.api.cognitive.microsoft.com/sts/v1.0/issueToken
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

Use these samples to create your access token request.

HTTP sample

This example is a simple HTTP request to get a token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.api.cognitive.microsoft.com
Content-type: application/x-www-form-urlencoded
Content-Length: 0
```

The body of the response contains the access token in JSON Web Token (JWT) format.

PowerShell sample

This example is a simple PowerShell script to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
$FetchTokenHeader = @{
  'Content-type'='application/x-www-form-urlencoded';
  'Content-Length'= '0';
  'Ocp-Apim-Subscription-Key' = 'YOUR_SUBSCRIPTION_KEY'
}

$OAuthToken = Invoke-RestMethod -Method POST -Uri
https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
 -Headers $FetchTokenHeader

# show the token received
$OAuthToken
```

cURL sample

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). This cURL command illustrates how to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
curl -v -X POST \
 "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
 -H "Content-type: application/x-www-form-urlencoded" \
 -H "Content-Length: 0" \
 -H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

C# sample

This C# class illustrates how to get an access token. Pass your Speech Service subscription key when you instantiate the class. If your subscription isn't in the West US region, change the value of `FetchTokenUri` to match the region for your subscription.

```
public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
    private string subscriptionKey;
    private string token;

    public Authentication(string subscriptionKey)
    {
        this.subscriptionKey = subscriptionKey;
        this.token = FetchTokenAsync(FetchTokenUri, subscriptionKey).Result;
    }

    public string GetAccessToken()
    {
        return this.token;
    }

    private async Task<string> FetchTokenAsync(string fetchUri, string subscriptionKey)
    {
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
            UriBuilder uriBuilder = new UriBuilder(fetchUri);

            var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
            Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
            return await result.Content.ReadAsStringAsync();
        }
    }
}
```

Python sample

```
# Request module must be installed.
# Run pip install requests if necessary.
import requests

subscription_key = 'REPLACE_WITH_YOUR_KEY'


def get_token(subscription_key):
    fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key
    }
    response = requests.post(fetch_token_url, headers=headers)
    access_token = str(response.text)
    print(access_token)
```

How to use an access token

The access token should be sent to the service as the `Authorization: Bearer <TOKEN>` header. Each access token is valid for 10 minutes. You can get a new token at any time, however, to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the Speech-to-text REST API for short audio:

```
POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.stt.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

// Message body here...
```

Regions and endpoints

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 matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

> **NOTE**
>
> The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is:
>
> ```
> https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
> ```
>
> .

Query parameters

These parameters may be included in the query string of the REST request.

PARAMETER	DESCRIPTION	REQUIRED / OPTIONAL
`language`	Identifies the spoken language that is being recognized. See Supported languages.	Required
`format`	Specifies the result format. Accepted values are `simple` and `detailed`. Simple results include `RecognitionStatus`, `DisplayText`, `Offset`, and `Duration`. Detailed responses include four different representations of display text. The default setting is `simple`.	Optional
`profanity`	Specifies how to handle profanity in recognition results. Accepted values are `masked`, which replaces profanity with asterisks, `removed`, which removes all profanity from the result, or `raw`, which includes the profanity in the result. The default setting is `masked`.	Optional
`cid`	When using the Custom Speech portal to create custom models, you can use custom models via their **Endpoint ID** found on the **Deployment** page. Use the **Endpoint ID** as the argument to the `cid` query string parameter.	Optional

Request headers

This table lists required and optional headers for Speech-to-text requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
`Ocp-Apim-Subscription-Key`	Your Speech service subscription key.	Either this header or `Authorization` is required.
`Authorization`	An authorization token preceded by the word `Bearer`. For more information, see Authentication.	Either this header or `Ocp-Apim-Subscription-Key` is required.
`Pronunciation-Assessment`	Specifies the parameters for showing pronunciation scores in recognition results, which assess the pronunciation quality of speech input, with indicators of accuracy, fluency, completeness, etc. This parameter is a base64 encoded json containing multiple detailed parameters. See Pronunciation assessment parameters for how to build this header.	Optional
`Content-type`	Describes the format and codec of the provided audio data. Accepted values are `audio/wav; codecs=audio/pcm; samplerate=16000` and `audio/ogg; codecs=opus`.	Required
`Transfer-Encoding`	Specifies that chunked audio data is being sent, rather than a single file. Only use this header if chunking audio data.	Optional
`Expect`	If using chunked transfer, send `Expect: 100-continue`. The Speech service acknowledges the initial request and awaits additional data.	Required if sending chunked audio data.
`Accept`	If provided, it must be `application/json`. The Speech service provides results in JSON. Some request frameworks provide an incompatible default value. It is good practice to always include `Accept`.	Optional, but recommended.

Audio formats

Audio is sent in the body of the HTTP `POST` request. It must be in one of the formats in this table:

FORMAT	CODEC	BIT RATE	SAMPLE RATE
WAV	PCM	256 kbps	16 kHz, mono
OGG	OPUS	256 kpbs	16 kHz, mono

> **NOTE**
> The above formats are supported through REST API for short audio and WebSocket in the Speech service. The Speech SDK currently supports the WAV format with PCM codec as well as other formats.

Pronunciation assessment parameters

This table lists required and optional parameters for pronunciation assessment.

PARAMETER	DESCRIPTION	REQUIRED?
ReferenceText	The text that the pronunciation will be evaluated against.	Required
GradingSystem	The point system for score calibration. The `FivePoint` system gives a 0-5 floating point score, and `HundredMark` gives a 0-100 floating point score. Default: `FivePoint`.	Optional
Granularity	The evaluation granularity. Accepted values are `Phoneme`, which shows the score on the full text, word and phoneme level, `Word`, which shows the score on the full text and word level, `FullText`, which shows the score on the full text level only. The default setting is `Phoneme`.	Optional
Dimension	Defines the output criteria. Accepted values are `Basic`, which shows the accuracy score only, `Comprehensive` shows scores on more dimensions (e.g. fluency score and completeness score on the full text level, error type on word level). Check Response parameters to see definitions of different score dimensions and word error types. The default setting is `Basic`.	Optional
EnableMiscue	Enables miscue calculation. With this enabled, the pronounced words will be compared to the reference text, and will be marked with omission/insertion based on the comparison. Accepted values are `False` and `True`. The default setting is `False`.	Optional
ScenarioId	A GUID indicating a customized point system.	Optional

Below is an example JSON containing the pronunciation assessment parameters:

```
{
  "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:

```
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) uploading while posting the audio data, which can significantly reduce the latency. See sample code in different programming languages for how to enable streaming.

> **NOTE**
>
> The pronunciation assessment feature currently supports `en-US` language, which is available on all speech-to-text regions. The support for `en-GB` and `zh-CN` languages is under preview.

Sample request

The sample below includes the hostname and required headers. It's important to note that the service also expects audio data, which is not included in this sample. As mentioned earlier, chunking is recommended, however, not required.

```
POST speech/recognition/conversation/cognitiveservices/v1?language=en-US&format=detailed HTTP/1.1
Accept: application/json;text/xml
Content-Type: audio/wav; codecs=audio/pcm; samplerate=16000
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.stt.speech.microsoft.com
Transfer-Encoding: chunked
Expect: 100-continue
```

To enable pronunciation assessment, you can add below header. See Pronunciation assessment parameters for how to build this header.

```
Pronunciation-Assessment: eyJSZWZlcm...
```

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
`100`	Continue	The initial request has been accepted. Proceed with sending the rest of the data. (Used with chunked transfer)
`200`	OK	The request was successful; the response body is a JSON object.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
`400`	Bad request	Language code not provided, not a supported language, invalid audio file, etc.
`401`	Unauthorized	Subscription key or authorization token is invalid in the specified region, or invalid endpoint.
`403`	Forbidden	Missing subscription key or authorization token.

Chunked transfer

Chunked transfer (`Transfer-Encoding: chunked`) can help reduce recognition latency. It allows the Speech service to begin processing the audio file while it is transmitted. The REST API for short audio does not provide partial or interim results.

This code sample shows how to send audio in chunks. Only the first chunk should contain the audio file's header. `request` is an `HttpWebRequest` object connected to the appropriate REST endpoint. `audioFile` is the path to an audio file on disk.

```
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_SUBSCRIPTION_KEY";
request.AllowWriteStreamBuffering = false;

using (var fs = new FileStream(audioFile, FileMode.Open, FileAccess.Read))
{
    // Open a request stream and write 1024 byte chunks in the stream one at a time.
    byte[] buffer = null;
    int bytesRead = 0;
    using (var requestStream = request.GetRequestStream())
    {
        // Read 1024 raw bytes from the input audio file.
        buffer = new Byte[checked((uint)Math.Min(1024, (int)fs.Length))];
        while ((bytesRead = fs.Read(buffer, 0, buffer.Length)) != 0)
        {
            requestStream.Write(buffer, 0, bytesRead);
        }

        requestStream.Flush();
    }
}
```

Response parameters

Results are provided as JSON. The `simple` format includes these top-level fields.

PARAMETER	DESCRIPTION
`RecognitionStatus`	Status, such as `Success` for successful recognition. See next table.

PARAMETER	DESCRIPTION
`DisplayText`	The recognized text after capitalization, punctuation, inverse text normalization (conversion of spoken text to shorter forms, such as 200 for "two hundred" or "Dr. Smith" for "doctor smith"), and profanity masking. Present only on success.
`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 may contain these values:

STATUS	DESCRIPTION
`Success`	The recognition was successful and the `DisplayText` field is present.
`NoMatch`	Speech was detected in the audio stream, but no words from the target language were matched. Usually means the recognition language is a different language from the one the user is speaking.
`InitialSilenceTimeout`	The start of the audio stream contained only silence, and the service timed out waiting for speech.
`BabbleTimeout`	The start of the audio stream contained only noise, and the service timed out 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 using the `detailed` format, `DisplayText` is provided as `Display` for each result in the `NBest` list.

The object in the `NBest` list can include:

PARAMETER	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.

PARAMETER	DESCRIPTION
`ITN`	The inverse-text-normalized ("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 `DisplayText` provided when format is set to `simple`.
`AccuracyScore`	Pronunciation accuracy of the speech. Accuracy indicates how closely the phonemes match a native speaker's pronunciation. Word and full text level accuracy score is aggregated from phoneme level accuracy score.
`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, determined by calculating the ratio of pronounced words to reference text input.
`PronScore`	Overall score indicating the pronunciation quality of the given speech. This is aggregated from `AccuracyScore`, `FluencyScore` and `CompletenessScore` with weight.
`ErrorType`	This value indicates whether a word is omitted, inserted or badly pronounced, compared to `ReferenceText`. Possible values are `None` (meaning no error on this word), `Omission`, `Insertion` and `Mispronunciation`.

Sample responses

A typical response for `simple` recognition:

```
{
  "RecognitionStatus": "Success",
  "DisplayText": "Remind me to buy 5 pencils.",
  "Offset": "1236645672289",
  "Duration": "1236645672289"
}
```

A typical response for `detailed` recognition:

```
{
  "RecognitionStatus": "Success",
  "Offset": "1236645672289",
  "Duration": "1236645672289",
  "NBest": [
    {
      "Confidence": 0.9052885,
      "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"
    }
  ]
}
```

A typical response for recognition with pronunciation assessment:

```
{
  "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
            }
        ]
      }
  ]
}
```

Next steps

- Create a free Azure account
- Customize acoustic models

- Customize language models
- Get familiar with Batch transcription

Language and voice support for the Speech service

11/23/2021 • 22 minutes to read • Edit Online

Language support varies by Speech service functionality. The following tables summarize language support for Speech-to-text, Text-to-speech, Speech translation and Speaker Recognition service offerings.

Speech-to-text

Both the Microsoft Speech SDK and the REST API support the following languages (locales).

To improve accuracy, customization is offered for a subset of the languages through uploading **Audio + Human-labeled Transcripts** or **Related Text: Sentences**. Support for customization of the acoustic model with **Audio + Human-labeled Transcripts** is limited to the specific base models listed below. Other base models and languages will only use the text of the transcripts to train custom models just like with **Related Text: Sentences**. To learn more about customization, see Get started with Custom Speech.

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Algeria)	`ar-DZ`	Text		
Arabic (Bahrain), modern standard	`ar-BH`	Text		
Arabic (Egypt)	`ar-EG`	Text	Yes	
Arabic (Iraq)	`ar-IQ`	Text		
Arabic (Israel)	`ar-IL`	Text		
Arabic (Jordan)	`ar-JO`	Text		
Arabic (Kuwait)	`ar-KW`	Text		
Arabic (Lebanon)	`ar-LB`	Text		
Arabic (Libya)	`ar-LY`	Text		
Arabic (Morocco)	`ar-MA`	Text		
Arabic (Oman)	`ar-OM`	Text		
Arabic (Qatar)	`ar-QA`	Text		
Arabic (Saudi Arabia)	`ar-SA`	Text		
Arabic (Palestinian Authority)	`ar-PS`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Syria)	`ar-SY`	Text		
Arabic (Tunisia)	`ar-TN`	Text		
Arabic (United Arab Emirates)	`ar-AE`	Text		
Arabic (Yemen)	`ar-YE`	Text		
Bulgarian (Bulgaria)	`bg-BG`	Text		
Catalan (Spain)	`ca-ES`	Text Pronunciation	Yes	
Chinese (Cantonese, Traditional)	`zh-HK`	Audio (20201015) Text	Yes	
Chinese (Mandarin, Simplified)	`zh-CN`	Audio (20200910) Text	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Audio (20190701, 20201015) Text	Yes	
Croatian (Croatia)	`hr-HR`	Text Pronunciation		
Czech (Czech)	`cs-CZ`	Text Pronunciation		
Danish (Denmark)	`da-DK`	Text Pronunciation	Yes	
Dutch (Netherlands)	`nl-NL`	Audio (20201015) Text Pronunciation	Yes	
English (Australia)	`en-AU`	Audio (20201019) Text Pronunciation	Yes	
English (Canada)	`en-CA`	Audio (20201019) Text Pronunciation	Yes	
English (Ghana)	`en-GH`	Text Pronunciation		
English (Hong Kong)	`en-HK`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
English (India)	`en-IN`	Audio (20200923) Text Pronunciation		
English (Ireland)	`en-IE`	Text Pronunciation		
English (Kenya)	`en-KE`	Text Pronunciation		
English (New Zealand)	`en-NZ`	Audio (20201019) Text Pronunciation		
English (Nigeria)	`en-NG`	Text Pronunciation		
English (Philippines)	`en-PH`	Text Pronunciation		
English (Singapore)	`en-SG`	Text Pronunciation		
English (South Africa)	`en-ZA`	Text Pronunciation		
English (Tanzania)	`en-TZ`	Text Pronunciation		
English (United Kingdom)	`en-GB`	Audio (20201019) Text Pronunciation	Yes	Yes
English (United States)	`en-US`	Audio (20201019, 20210223) Text Pronunciation	Yes	Yes
Estonian(Estonia)	`et-EE`	Text Pronunciation		
Filipino (Philippines)	`fil-PH`	Text Pronunciation		
Finnish (Finland)	`fi-FI`	Text Pronunciation	Yes	
French (Canada)	`fr-CA`	Audio (20201015) Text Pronunciation	Yes	
French (France)	`fr-FR`	Audio (20201015) Text Pronunciation	Yes	

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
French (Switzerland)	`fr-CH`	Text Pronunciation		
German (Austria)	`de-AT`	Text Pronunciation		
German (Switzerland)	`de-CH`	Text Pronunciation		
German (Germany)	`de-DE`	Audio (20190701, 20200619, 20201127) Text Pronunciation	Yes	
Greek (Greece)	`el-GR`	Text	Yes	
Gujarati (Indian)	`gu-IN`	Text		
Hebrew (Israel)	`he-IL`	Text		
Hindi (India)	`hi-IN`	Audio (20200701) Text	Yes	
Hungarian (Hungary)	`hu-HU`	Text Pronunciation		
Indonesian (Indonesia)	`id-ID`	Text Pronunciation		
Irish (Ireland)	`ga-IE`	Text Pronunciation		
Italian (Italy)	`it-IT`	Audio (20201016) Text Pronunciation	Yes	
Japanese (Japan)	`ja-JP`	Text	Yes	
Kannada (India)	`kn-IN`	Text		
Korean (Korea)	`ko-KR`	Audio (20201015) Text	Yes	
Latvian (Latvia)	`lv-LV`	Text Pronunciation		
Lithuanian (Lithuania)	`lt-LT`	Text Pronunciation		
Malay (Malaysia)	`ms-MY`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Maltese (Malta)	`mt-MT`	Text		
Marathi (India)	`mr-IN`	Text		
Norwegian (Bokmål, Norway)	`nb-NO`	Text	Yes	
Persian (Iran)	`fa-IR`	Text		
Polish (Poland)	`pl-PL`	Text Pronunciation	Yes	
Portuguese (Brazil)	`pt-BR`	Audio (20190620, 20201015) Text Pronunciation	Yes	
Portuguese (Portugal)	`pt-PT`	Text Pronunciation	Yes	
Romanian (Romania)	`ro-RO`	Text Pronunciation	Yes	
Russian (Russia)	`ru-RU`	Audio (20200907) Text	Yes	
Slovak (Slovakia)	`sk-SK`	Text Pronunciation		
Slovenian (Slovenia)	`sl-SI`	Text Pronunciation		
Spanish (Argentina)	`es-AR`	Text Pronunciation		
Spanish (Bolivia)	`es-BO`	Text Pronunciation		
Spanish (Chile)	`es-CL`	Text Pronunciation		
Spanish (Colombia)	`es-CO`	Text Pronunciation		
Spanish (Costa Rica)	`es-CR`	Text Pronunciation		
Spanish (Cuba)	`es-CU`	Text Pronunciation		
Spanish (Dominican Republic)	`es-DO`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Spanish (Ecuador)	`es-EC`	Text Pronunciation		
Spanish (El Salvador)	`es-SV`	Text Pronunciation		
Spanish (Equatorial Guinea)	`es-GQ`	Text		
Spanish (Guatemala)	`es-GT`	Text Pronunciation		
Spanish (Honduras)	`es-HN`	Text Pronunciation		
Spanish (Mexico)	`es-MX`	Audio (20200907) Text Pronunciation	Yes	
Spanish (Nicaragua)	`es-NI`	Text Pronunciation		
Spanish (Panama)	`es-PA`	Text Pronunciation		
Spanish (Paraguay)	`es-PY`	Text Pronunciation		
Spanish (Peru)	`es-PE`	Text Pronunciation		
Spanish (Puerto Rico)	`es-PR`	Text Pronunciation		
Spanish (Spain)	`es-ES`	Audio (20201015) Text Pronunciation	Yes	
Spanish (Uruguay)	`es-UY`	Text Pronunciation		
Spanish (USA)	`es-US`	Text Pronunciation		
Spanish (Venezuela)	`es-VE`	Text Pronunciation		
Swahili (Kenya)	`sw-KE`	Text		
Swedish (Sweden)	`sv-SE`	Text Pronunciation	Yes	
Tamil (India)	`ta-IN`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Telugu (India)	`te-IN`	Text		
Thai (Thailand)	`th-TH`	Text	Yes	
Turkish (Turkey)	`tr-TR`	Text		
Vietnamese (Vietnam)	`vi-VN`	Text		

Text-to-speech

Both the Microsoft Speech SDK and REST APIs support these voices, each of which supports a specific language and dialect, identified by locale. You can also get a full list of languages and voices supported for each specific region/endpoint through the voices list API.

> **IMPORTANT**
> Pricing varies for standard, custom and neural voices. Please visit the Pricing page for additional information.

Neural voices

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

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

> **NOTE**
> Neural voices are created from samples that use a 24 khz sample rate. All voices can upsample or downsample to other sample rates when synthesizing.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Afrikaans (South Africa)	`af-ZA`	Female	`af-ZA-AdriNeural` New	General
Afrikaans (South Africa)	`af-ZA`	Male	`af-ZA-WillemNeural` New	General
Amharic (Ethiopia)	`am-ET`	Female	`am-ET-MekdesNeural` New	General
Amharic (Ethiopia)	`am-ET`	Male	`am-ET-AmehaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Algeria)	`ar-DZ`	Female	`ar-DZ-AminaNeural` New	General
Arabic (Algeria)	`ar-DZ`	Male	`ar-DZ-IsmaelNeural` New	General
Arabic (Bahrain)	`ar-BH`	Female	`ar-BH-LailaNeural` New	General
Arabic (Bahrain)	`ar-BH`	Male	`ar-BH-AliNeural` New	General
Arabic (Egypt)	`ar-EG`	Female	`ar-EG-SalmaNeural`	General
Arabic (Egypt)	`ar-EG`	Male	`ar-EG-ShakirNeural`	General
Arabic (Iraq)	`ar-IQ`	Female	`ar-IQ-RanaNeural` New	General
Arabic (Iraq)	`ar-IQ`	Male	`ar-IQ-BasselNeural` New	General
Arabic (Jordan)	`ar-JO`	Female	`ar-JO-Sana Neural` New	General
Arabic (Jordan)	`ar-JO`	Male	`ar-JO-Taim Neural` New	General
Arabic (Kuwait)	`ar-KW`	Female	`ar-KW-NouraNeural` New	General
Arabic (Kuwait)	`ar-KW`	Male	`ar-KW-FahedNeural` New	General
Arabic (Libya)	`ar-LY`	Female	`ar-LY-ImanNeural` New	General
Arabic (Libya)	`ar-LY`	Male	`ar-LY-OmarNeural` New	General
Arabic (Morocco)	`ar-MA`	Female	`ar-MA-MounaNeural` New	General
Arabic (Morocco)	`ar-MA`	Male	`ar-MA-JamalNeural` New	General
Arabic (Qatar)	`ar-QA`	Female	`ar-QA-AmalNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Qatar)	`ar-QA`	Male	`ar-QA-MoazNeural` New	General
Arabic (Saudi Arabia)	`ar-SA`	Female	`ar-SA-ZariyahNeural`	General
Arabic (Saudi Arabia)	`ar-SA`	Male	`ar-SA-HamedNeural`	General
Arabic (Syria)	`ar-SY`	Female	`ar-SY-AmanyNeural` New	General
Arabic (Syria)	`ar-SY`	Male	`ar-SY-LaithNeural` New	General
Arabic (Tunisia)	`ar-TN`	Female	`ar-TN-ReemNeural` New	General
Arabic (Tunisia)	`ar-TN`	Male	`ar-TN-HediNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Female	`ar-AE-FatimaNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Male	`ar-AE-HamdanNeural` New	General
Arabic (Yemen)	`ar-YE`	Female	`ar-YE-MaryamNeural` New	General
Arabic (Yemen)	`ar-YE`	Male	`ar-YE-SalehNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Female	`bn-BD-NabanitaNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Male	`bn-BD-PradeepNeural` New	General
Bulgarian (Bulgaria)	`bg-BG`	Female	`bg-BG-KalinaNeural`	General
Bulgarian (Bulgaria)	`bg-BG`	Male	`bg-BG-BorislavNeural`	General
Burmese (Myanmar)	`my-MM`	Female	`my-MM-NilarNeural` New	General
Burmese (Myanmar)	`my-MM`	Male	`my-MM-ThihaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Catalan (Spain)	`ca-ES`	Female	`ca-ES-AlbaNeural`	General
Catalan (Spain)	`ca-ES`	Female	`ca-ES-JoanaNeural`	General
Catalan (Spain)	`ca-ES`	Male	`ca-ES-EnricNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuGaaiNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuMaanNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Male	`zh-HK-WanLungNeural`	General
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaohanNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaomoNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoruiNeural`	Senior voice, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxiaoNeural`	General, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxuanNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyouNeural`	Child voice, optimized for story narrating
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunxiNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyangNeural`	Optimized for news reading, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyeNeural`	Optimized for story narrating
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoChenNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoYuNeural`	General
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-YunJheNeural`	General
Croatian (Croatia)	`hr-HR`	Female	`hr-HR-GabrijelaNeural`	General
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-SreckoNeural`	General
Czech (Czech)	`cs-CZ`	Female	`cs-CZ-VlastaNeural`	General
Czech (Czech)	`cs-CZ`	Male	`cs-CZ-AntoninNeural`	General
Danish (Denmark)	`da-DK`	Female	`da-DK-ChristelNeural`	General
Danish (Denmark)	`da-DK`	Male	`da-DK-JeppeNeural`	General
Dutch (Belgium)	`nl-BE`	Female	`nl-BE-DenaNeural`	General
Dutch (Belgium)	`nl-BE`	Male	`nl-BE-ArnaudNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-ColetteNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-FennaNeural`	General
Dutch (Netherlands)	`nl-NL`	Male	`nl-NL-MaartenNeural`	General
English (Australia)	`en-AU`	Female	`en-AU-NatashaNeural`	General
English (Australia)	`en-AU`	Male	`en-AU-WilliamNeural`	General
English (Canada)	`en-CA`	Female	`en-CA-ClaraNeural`	General
English (Canada)	`en-CA`	Male	`en-CA-LiamNeural`	General
English (Hongkong)	`en-HK`	Female	`en-HK-YanNeural`	General
English (Hongkong)	`en-HK`	Male	`en-HK-SamNeural`	General
English (India)	`en-IN`	Female	`en-IN-NeerjaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (India)	`en-IN`	Male	`en-IN-PrabhatNeural`	General
English (Ireland)	`en-IE`	Female	`en-IE-EmilyNeural`	General
English (Ireland)	`en-IE`	Male	`en-IE-ConnorNeural`	General
English (Kenya)	`en-KE`	Female	`en-KE-AsiliaNeural` New	General
English (Kenya)	`en-KE`	Male	`en-KE-ChilembaNeural` New	General
English (New Zealand)	`en-NZ`	Female	`en-NZ-MollyNeural`	General
English (New Zealand)	`en-NZ`	Male	`en-NZ-MitchellNeural`	General
English (Nigeria)	`en-NG`	Female	`en-NG-EzinneNeural` New	General
English (Nigeria)	`en-NG`	Male	`en-NG-AbeoNeural` New	General
English (Philippines)	`en-PH`	Female	`en-PH-RosaNeural`	General
English (Philippines)	`en-PH`	Male	`en-PH-JamesNeural`	General
English (Singapore)	`en-SG`	Female	`en-SG-LunaNeural`	General
English (Singapore)	`en-SG`	Male	`en-SG-WayneNeural`	General
English (South Africa)	`en-ZA`	Female	`en-ZA-LeahNeural`	General
English (South Africa)	`en-ZA`	Male	`en-ZA-LukeNeural`	General
English (Tanzania)	`en-TZ`	Female	`en-TZ-ImaniNeural` New	General
English (Tanzania)	`en-TZ`	Male	`en-TZ-ElimuNeural` New	General
English (United Kingdom)	`en-GB`	Female	`en-GB-LibbyNeural`	General
English (United Kingdom)	`en-GB`	Female	`en-GB-SoniaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United Kingdom)	`en-GB`	Female	`en-GB-MiaNeural` Retired on 30 October 2021, see below	General
English (United Kingdom)	`en-GB`	Male	`en-GB-RyanNeural`	General
English (United States)	`en-US`	Female	`en-US-AmberNeural`	General
English (United States)	`en-US`	Female	`en-US-AriaNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-AshleyNeural`	General
English (United States)	`en-US`	Female	`en-US-CoraNeural`	General
English (United States)	`en-US`	Female	`en-US-ElizabethNeural`	General
English (United States)	`en-US`	Female	`en-US-JennyNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-MichelleNeural`	General
English (United States)	`en-US`	Female	`en-US-MonicaNeural`	General
English (United States)	`en-US`	Female	`en-US-SaraNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Kid	`en-US-AnaNeural`	General
English (United States)	`en-US`	Male	`en-US-BrandonNeural`	General
English (United States)	`en-US`	Male	`en-US-ChristopherNeural`	General
English (United States)	`en-US`	Male	`en-US-EricNeural`	General
English (United States)	`en-US`	Male	`en-US-GuyNeural`	General, multiple voice styles available using SSML

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Male	`en-US-JacobNeural`	General
Estonian (Estonia)	`et-EE`	Female	`et-EE-AnuNeural`	General
Estonian (Estonia)	`et-EE`	Male	`et-EE-KertNeural`	General
Filipino (Philippines)	`fil-PH`	Female	`fil-PH-BlessicaNeural` New	General
Filipino (Philippines)	`fil-PH`	Male	`fil-PH-AngeloNeural` New	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-NooraNeural`	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-SelmaNeural`	General
Finnish (Finland)	`fi-FI`	Male	`fi-FI-HarriNeural`	General
French (Belgium)	`fr-BE`	Female	`fr-BE-CharlineNeural`	General
French (Belgium)	`fr-BE`	Male	`fr-BE-GerardNeural`	General
French (Canada)	`fr-CA`	Female	`fr-CA-SylvieNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-AntoineNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-JeanNeural`	General
French (France)	`fr-FR`	Female	`fr-FR-DeniseNeural`	General
French (France)	`fr-FR`	Male	`fr-FR-HenriNeural`	General
French (Switzerland)	`fr-CH`	Female	`fr-CH-ArianeNeural`	General
French (Switzerland)	`fr-CH`	Male	`fr-CH-FabriceNeural`	General
Galician (Spain)	`gl-ES`	Female	`gl-ES-SabelaNeural` New	General
Galician (Spain)	`gl-ES`	Male	`gl-ES-RoiNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
German (Austria)	`de-AT`	Female	`de-AT-IngridNeural`	General
German (Austria)	`de-AT`	Male	`de-AT-JonasNeural`	General
German (Germany)	`de-DE`	Female	`de-DE-KatjaNeural`	General
German (Germany)	`de-DE`	Male	`de-DE-ConradNeural`	General
German (Switzerland)	`de-CH`	Female	`de-CH-LeniNeural`	General
German (Switzerland)	`de-CH`	Male	`de-CH-JanNeural`	General
Greek (Greece)	`el-GR`	Female	`el-GR-AthinaNeural`	General
Greek (Greece)	`el-GR`	Male	`el-GR-NestorasNeural`	General
Gujarati (India)	`gu-IN`	Female	`gu-IN-DhwaniNeural`	General
Gujarati (India)	`gu-IN`	Male	`gu-IN-NiranjanNeural`	General
Hebrew (Israel)	`he-IL`	Female	`he-IL-HilaNeural`	General
Hebrew (Israel)	`he-IL`	Male	`he-IL-AvriNeural`	General
Hindi (India)	`hi-IN`	Female	`hi-IN-SwaraNeural`	General
Hindi (India)	`hi-IN`	Male	`hi-IN-MadhurNeural`	General
Hungarian (Hungary)	`hu-HU`	Female	`hu-HU-NoemiNeural`	General
Hungarian (Hungary)	`hu-HU`	Male	`hu-HU-TamasNeural`	General
Indonesian (Indonesia)	`id-ID`	Female	`id-ID-GadisNeural`	General
Indonesian (Indonesia)	`id-ID`	Male	`id-ID-ArdiNeural`	General
Irish (Ireland)	`ga-IE`	Female	`ga-IE-OrlaNeural`	General
Irish (Ireland)	`ga-IE`	Male	`ga-IE-ColmNeural`	General
Italian (Italy)	`it-IT`	Female	`it-IT-ElsaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Italian (Italy)	`it-IT`	Female	`it-IT-IsabellaNeural`	General
Italian (Italy)	`it-IT`	Male	`it-IT-DiegoNeural`	General
Japanese (Japan)	`ja-JP`	Female	`ja-JP-NanamiNeural`	General
Japanese (Japan)	`ja-JP`	Male	`ja-JP-KeitaNeural`	General
Javanese (Indonesia)	`jv-ID`	Female	`jv-ID-SitiNeural` New	General
Javanese (Indonesia)	`jv-ID`	Male	`jv-ID-DimasNeural` New	General
Khmer (Cambodia)	`km-KH`	Female	`km-KH-SreymomNeural` New	General
Khmer (Cambodia)	`km-KH`	Male	`km-KH-PisethNeural` New	General
Korean (Korea)	`ko-KR`	Female	`ko-KR-SunHiNeural`	General
Korean (Korea)	`ko-KR`	Male	`ko-KR-InJoonNeural`	General
Latvian (Latvia)	`lv-LV`	Female	`lv-LV-EveritaNeural`	General
Latvian (Latvia)	`lv-LV`	Male	`lv-LV-NilsNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Female	`lt-LT-OnaNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Male	`lt-LT-LeonasNeural`	General
Malay (Malaysia)	`ms-MY`	Female	`ms-MY-YasminNeural`	General
Malay (Malaysia)	`ms-MY`	Male	`ms-MY-OsmanNeural`	General
Maltese (Malta)	`mt-MT`	Female	`mt-MT-GraceNeural`	General
Maltese (Malta)	`mt-MT`	Male	`mt-MT-JosephNeural`	General
Marathi (India)	`mr-IN`	Female	`mr-IN-AarohiNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Marathi (India)	`mr-IN`	Male	`mr-IN-ManoharNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-IselinNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-PernilleNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Male	`nb-NO-FinnNeural`	General
Persian (Iran)	`fa-IR`	Female	`fa-IR-DilaraNeural` New	General
Persian (Iran)	`fa-IR`	Male	`fa-IR-FaridNeural` New	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-AgnieszkaNeural`	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-ZofiaNeural`	General
Polish (Poland)	`pl-PL`	Male	`pl-PL-MarekNeural`	General
Portuguese (Brazil)	`pt-BR`	Female	`pt-BR-FranciscaNeural`	General, multiple voice styles available using SSML
Portuguese (Brazil)	`pt-BR`	Male	`pt-BR-AntonioNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-FernandaNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-RaquelNeural`	General
Portuguese (Portugal)	`pt-PT`	Male	`pt-PT-DuarteNeural`	General
Romanian (Romania)	`ro-RO`	Female	`ro-RO-AlinaNeural`	General
Romanian (Romania)	`ro-RO`	Male	`ro-RO-EmilNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-DariyaNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-SvetlanaNeural`	General
Russian (Russia)	`ru-RU`	Male	`ru-RU-DmitryNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Slovak (Slovakia)	`sk-SK`	Female	`sk-SK-ViktoriaNeural`	General
Slovak (Slovakia)	`sk-SK`	Male	`sk-SK-LukasNeural`	General
Slovenian (Slovenia)	`sl-SI`	Female	`sl-SI-PetraNeural`	General
Slovenian (Slovenia)	`sl-SI`	Male	`sl-SI-RokNeural`	General
Somali (Somalia)	`so-SO`	Female	`so-SO-UbaxNeural` New	General
Somali (Somalia)	`so-SO`	Male	`so-SO-MuuseNeural` New	General
Spanish (Argentina)	`es-AR`	Female	`es-AR-ElenaNeural`	General
Spanish (Argentina)	`es-AR`	Male	`es-AR-TomasNeural`	General
Spanish (Bolivia)	`es-BO`	Female	`es-BO-SofiaNeural` New	General
Spanish (Bolivia)	`es-BO`	Male	`es-BO-MarceloNeural` New	General
Spanish (Chile)	`es-CL`	Female	`es-CL-CatalinaNeural` New	General
Spanish (Chile)	`es-CL`	Male	`es-CL-LorenzoNeural` New	General
Spanish (Colombia)	`es-CO`	Female	`es-CO-SalomeNeural`	General
Spanish (Colombia)	`es-CO`	Male	`es-CO-GonzaloNeural`	General
Spanish (Costa Rica)	`es-CR`	Female	`es-CR-MariaNeural` New	General
Spanish (Costa Rica)	`es-CR`	Male	`es-CR-JuanNeural` New	General
Spanish (Cuba)	`es-CU`	Female	`es-CU-BelkysNeural` New	General
Spanish (Cuba)	`es-CU`	Male	`es-CU-ManuelNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Dominican Republic)	`es-DO`	Female	`es-DO-RamonaNeural` New	General
Spanish (Dominican Republic)	`es-DO`	Male	`es-DO-EmilioNeural` New	General
Spanish (Ecuador)	`es-EC`	Female	`es-EC-AndreaNeural` New	General
Spanish (Ecuador)	`es-EC`	Male	`es-EC-LuisNeural` New	General
Spanish (El Salvador)	`es-SV`	Female	`es-SV-LorenaNeural` New	General
Spanish (El Salvador)	`es-SV`	Male	`es-SV-RodrigoNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Female	`es-GQ-TeresaNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Male	`es-GQ-JavierNeural` New	General
Spanish (Guatemala)	`es-GT`	Female	`es-GT-MartaNeural` New	General
Spanish (Guatemala)	`es-GT`	Male	`es-GT-AndresNeural` New	General
Spanish (Honduras)	`es-HN`	Female	`es-HN-KarlaNeural` New	General
Spanish (Honduras)	`es-HN`	Male	`es-HN-CarlosNeural` New	General
Spanish (Mexico)	`es-MX`	Female	`es-MX-DaliaNeural`	General
Spanish (Mexico)	`es-MX`	Male	`es-MX-JorgeNeural`	General
Spanish (Nicaragua)	`es-NI`	Female	`es-NI-YolandaNeural` New	General
Spanish (Nicaragua)	`es-NI`	Male	`es-NI-FedericoNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Panama)	`es-PA`	Female	`es-PA-MargaritaNeural` New	General
Spanish (Panama)	`es-PA`	Male	`es-PA-RobertoNeural` New	General
Spanish (Paraguay)	`es-PY`	Female	`es-PY-TaniaNeural` New	General
Spanish (Paraguay)	`es-PY`	Male	`es-PY-MarioNeural` New	General
Spanish (Peru)	`es-PE`	Female	`es-PE-CamilaNeural` New	General
Spanish (Peru)	`es-PE`	Male	`es-PE-AlexNeural` New	General
Spanish (Puerto Rico)	`es-PR`	Female	`es-PR-Karina Neural` New	General
Spanish (Puerto Rico)	`es-PR`	Male	`es-PR-Victor Neural` New	General
Spanish (Spain)	`es-ES`	Female	`es-ES-ElviraNeural`	General
Spanish (Spain)	`es-ES`	Male	`es-ES-AlvaroNeural`	General
Spanish (Uruguay)	`es-UY`	Female	`es-UY-ValentinaNeural` New	General
Spanish (Uruguay)	`es-UY`	Male	`es-UY-MateoNeural` New	General
Spanish (US)	`es-US`	Female	`es-US-PalomaNeural`	General
Spanish (US)	`es-US`	Male	`es-US-AlonsoNeural`	General
Spanish (Venezuela)	`es-VE`	Female	`es-VE-PaolaNeural` New	General
Spanish (Venezuela)	`es-VE`	Male	`es-VE-SebastianNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Sundanese (Indonesia)	`su-ID`	Female	`su-ID-TutiNeural` New	General
Sundanese (Indonesia)	`su-ID`	Male	`su-ID-JajangNeural` New	General
Swahili (Kenya)	`sw-KE`	Female	`sw-KE-ZuriNeural`	General
Swahili (Kenya)	`sw-KE`	Male	`sw-KE-RafikiNeural`	General
Swahili (Tanzania)	`sw-TZ`	Female	`sw-TZ-RehemaNeural` New	General
Swahili (Tanzania)	`sw-TZ`	Male	`sw-TZ-DaudiNeural` New	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-HilleviNeural`	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-SofieNeural`	General
Swedish (Sweden)	`sv-SE`	Male	`sv-SE-MattiasNeural`	General
Tamil (India)	`ta-IN`	Female	`ta-IN-PallaviNeural`	General
Tamil (India)	`ta-IN`	Male	`ta-IN-ValluvarNeural`	General
Tamil (Singapore)	`ta-SG`	Female	`ta-SG-VenbaNeural` New	General
Tamil (Singapore)	`ta-SG`	Male	`ta-SG-AnbuNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Female	`ta-LK-SaranyaNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Male	`ta-LK-KumarNeural` New	General
Telugu (India)	`te-IN`	Female	`te-IN-ShrutiNeural`	General
Telugu (India)	`te-IN`	Male	`te-IN-MohanNeural`	General
Thai (Thailand)	`th-TH`	Female	`th-TH-AcharaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Thai (Thailand)	th-TH	Female	th-TH-PremwadeeNeural	General
Thai (Thailand)	th-TH	Male	th-TH-NiwatNeural	General
Turkish (Turkey)	tr-TR	Female	tr-TR-EmelNeural	General
Turkish (Turkey)	tr-TR	Male	tr-TR-AhmetNeural	General
Ukrainian (Ukraine)	uk-UA	Female	uk-UA-PolinaNeural	General
Ukrainian (Ukraine)	uk-UA	Male	uk-UA-OstapNeural	General
Urdu (India)	ur-IN	Female	ur-IN-GulNeural New	General
Urdu (India)	ur-IN	Male	ur-IN-SalmanNeural New	General
Urdu (Pakistan)	ur-PK	Female	ur-PK-UzmaNeural	General
Urdu (Pakistan)	ur-PK	Male	ur-PK-AsadNeural	General
Uzbek (Uzbekistan)	uz-UZ	Female	uz-UZ-MadinaNeural New	General
Uzbek (Uzbekistan)	uz-UZ	Male	uz-UZ-SardorNeural New	General
Vietnamese (Vietnam)	vi-VN	Female	vi-VN-HoaiMyNeural	General
Vietnamese (Vietnam)	vi-VN	Male	vi-VN-NamMinhNeural	General
Welsh (United Kingdom)	cy-GB	Female	cy-GB-NiaNeural	General
Welsh (United Kingdom)	cy-GB	Male	cy-GB-AledNeural	General
Zulu (South Africa)	zu-ZA	Female	zu-ZA-ThandoNeural New	General
Zulu (South Africa)	zu-ZA	Male	zu-ZA-ThembaNeural New	General

> **IMPORTANT**
>
> The English (United Kingdom) voice `en-GB-MiaNeural` retired on **30 October 2021**. All service requests to `en-GB-MiaNeural` now will be re-directed to `en-GB-SoniaNeural` automatically since **30 October 2021**. If you are using container Neural TTS, please download and deploy the latest version, starting from **30 October 2021**, all requests with previous versions will be rejected.

Neural voices in preview

Below neural voices are in public preview.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Female	`en-US-JennyMultilingualNeural` New	General, multi-lingual capabilities available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaochenNeural` New	Optimized for spontaneous conversation
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyanNeural` New	Optimized for customer service
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoshuangNeural` New	Child voice, optimized for child story and chat; multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoqiuNeural` New	Optimized for narrating

> **IMPORTANT**
>
> Voices in public preview are only available in 3 service regions: East US, West Europe and Southeast Asia.

> **TIP**
>
> `en-US-JennyNeuralMultilingual` supports multiple languages. Check the voices list API for supported languages list.

For more information about regional availability, see regions.

To learn how you can configure and adjust neural voices, such as Speaking Styles, see Speech Synthesis Markup Language.

> **IMPORTANT**
>
> The `en-US-JessaNeural` voice has changed to `en-US-AriaNeural`. If you were using "Jessa" before, convert over to "Aria".

> **TIP**
> You can continue to use the full service name mapping like "Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)" in your speech synthesis requests.

Standard voices

More than 75 standard voices are available in over 45 languages and locales, which allow you to convert text into synthesized speech. For more information about regional availability, see regions.

> **IMPORTANT**
> We are retiring the standard voices on **31st August 2024** and they will no longer be supported after that date.We announced this in emails sent to all existing Speech subscriptions created before **31st August 2021**. During the retiring period (**31st August 2021 - 31st August 2024**), existing standard voice users can continue to use standard voices, but all new users/new speech resources must choose neural voices.

> **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 (Egypt)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-Zhiwei`
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-Matej`
Czech (Czech)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
French (France)	`fr-FR`	Female	`fr-FR-Julie`
French (France)	`fr-FR`	Male	`fr-FR-Paul`
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-HeloisaRUS`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-HeliaRUS`
Romanian (Romania)	`ro-RO`	Male	`ro-RO-Andrei`
Russian (Russia)	`ru-RU`	Female	`ru-RU-EkaterinaRUS`
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".

TIP

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.

Customization

Custom Voice is available in the neural tier (a.k.a, Custom Neural Voice). Based on the Neural TTS technology and the multi-lingual multi-speaker universal model, Custom Neural Voice lets you create synthetic voices that

are rich in speaking styles, or adaptable cross languages. Check below for the languages supported.

> **IMPORTANT**
> The standard tier including the statistical parametric and the concatenative training methods of custom voice is being deprecated and will be retired on 2/29/2024. If you are using non-neural/standard Custom Voice, migrate to Custom Neural Voice immediately to enjoy the better quality and deploy the voices responsibly.

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Arabic (Egypt)	`ar-EG`	Yes	No
Bulgarian (Bulgaria)	`bg-BG`	Yes	No
Chinese (Mandarin, Simplified)	`zh-CN`	Yes	Yes
Chinese (Mandarin, Simplified), English bilingual	`zh-CN` bilingual	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Yes	No
Czech (Czech)	`cs-CZ`	Yes	No
Dutch (Netherlands)	`nl-NL`	Yes	No
English (Australia)	`en-AU`	Yes	Yes
English (Canada)	`en-CA`	Yes	No
English (India)	`en-IN`	Yes	No
English (Ireland)	`en-IE`	Yes	No
English (United Kingdom)	`en-GB`	Yes	Yes
English (United States)	`en-US`	Yes	Yes
French (Canada)	`fr-CA`	Yes	Yes
French (France)	`fr-FR`	Yes	Yes
German (Austria)	`de-AT`	Yes	No
German (Germany)	`de-DE`	Yes	Yes
Hungarian (Hungary)	`hu-HU`	Yes	No
Italian (Italy)	`it-IT`	Yes	Yes

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Japanese (Japan)	`ja-JP`	Yes	Yes
Korean (Korea)	`ko-KR`	Yes	Yes
Norwegian (Bokmål, Norway)	`nb-NO`	Yes	No
Portuguese (Brazil)	`pt-BR`	Yes	Yes
Portuguese (Portugal)	`pt-PT`	Yes	No
Russian (Russia)	`ru-RU`	Yes	Yes
Slovak (Slovakia)	`sk-SK`	Yes	No
Spanish (Mexico)	`es-MX`	Yes	Yes
Spanish (Spain)	`es-ES`	Yes	Yes
Turkish (Turkey)	`tr-TR`	Yes	No
Vietnamese (Vietnam)	`vi-VN`	Yes	No

Select the right locale that matches the training data you have to train a custom voice model. For example, if the recording data you have is spoken in English with a British accent, select `en-GB`.

> **NOTE**
> We do not support bi-lingual model training in Custom Voice, except for the Chinese-English bi-lingual. Select "Chinese-English bilingual" if you want to train a Chinese voice that can speak English as well. Chinese-English bilingual model training using the standard method is available in North Europe and North Central US only. Custom Neural Voice training is available in UK South and East US.

Speech translation

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

Text languages

TEXT LANGUAGE	LANGUAGE CODE
Afrikaans	`af`
Albanian	`sq`
Amharic	`am`

TEXT LANGUAGE	LANGUAGE CODE
Arabic	`ar`
Armenian	`hy`
Assamese	`as`
Azerbaijani	`az`
Bangla	`bn`
Bosnian (Latin)	`bs`
Bulgarian	`bg`
Cantonese (Traditional)	`yue`
Catalan	`ca`
Chinese (Literary)	`lzh`
Chinese Simplified	`zh-Hans`
Chinese Traditional	`zh-Hant`
Croatian	`hr`
Czech	`cs`
Danish	`da`
Dari	`prs`
Dutch	`nl`
English	`en`
Estonian	`et`
Fijian	`fj`
Filipino	`fil`
Finnish	`fi`
French	`fr`
French (Canada)	`fr-ca`

TEXT LANGUAGE	LANGUAGE CODE
German	`de`
Greek	`el`
Gujarati	`gu`
Haitian Creole	`ht`
Hebrew	`he`
Hindi	`hi`
Hmong Daw	`mww`
Hungarian	`hu`
Icelandic	`is`
Indonesian	`id`
Inuktitut	`iu`
Irish	`ga`
Italian	`it`
Japanese	`ja`
Kannada	`kn`
Kazakh	`kk`
Khmer	`km`
Klingon	`tlh-Latn`
Klingon (plqaD)	`tlh-Piqd`
Korean	`ko`
Kurdish (Central)	`ku`
Kurdish (Northern)	`kmr`
Lao	`lo`
Latvian	`lv`

TEXT LANGUAGE	LANGUAGE CODE
Lithuanian	`lt`
Malagasy	`mg`
Malay	`ms`
Malayalam	`ml`
Maltese	`mt`
Maori	`mi`
Marathi	`mr`
Myanmar	`my`
Nepali	`ne`
Norwegian	`nb`
Odia	`or`
Pashto	`ps`
Persian	`fa`
Polish	`pl`
Portuguese (Brazil)	`pt`
Portuguese (Portugal)	`pt-pt`
Punjabi	`pa`
Queretaro Otomi	`otq`
Romanian	`ro`
Russian	`ru`
Samoan	`sm`
Serbian (Cyrillic)	`sr-Cyrl`
Serbian (Latin)	`sr-Latn`
Slovak	`sk`

TEXT LANGUAGE	LANGUAGE CODE
Slovenian	`sl`
Spanish	`es`
Swahili	`sw`
Swedish	`sv`
Tahitian	`ty`
Tamil	`ta`
Telugu	`te`
Thai	`th`
Tigrinya	`ti`
Tongan	`to`
Turkish	`tr`
Ukrainian	`uk`
Urdu	`ur`
Vietnamese	`vi`
Welsh	`cy`
Yucatec Maya	`yua`

Speaker Recognition

Speaker recognition is mostly language agnostic. We built a universal model for text-independent speaker recognition by combining various data sources from multiple languages. We have tuned and evaluated the model on the languages and locales that appear in the following table. See the overview for additional information on Speaker Recognition.

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (US)	`en-US`	yes	yes	yes
Chinese (Mandarin, simplified)	`zh-CN`	n/a	yes	yes
English (Australia)	`en-AU`	n/a	yes	yes

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (Canada)	`en-CA`	n/a	yes	yes
English (India)	`en-IN`	n/a	yes	yes
English (UK)	`en-GB`	n/a	yes	yes
French (Canada)	`fr-CA`	n/a	yes	yes
French (France)	`fr-FR`	n/a	yes	yes
German (Germany)	`de-DE`	n/a	yes	yes
Italian	`it-IT`	n/a	yes	yes
Japanese	`ja-JP`	n/a	yes	yes
Portuguese (Brazil)	`pt-BR`	n/a	yes	yes
Spanish (Mexico)	`es-MX`	n/a	yes	yes
Spanish (Spain)	`es-ES`	n/a	yes	yes

Custom Keyword and Keyword Verification

The following table outlines supported languages for Custom Keyword and Keyword Verification.

LANGUAGE	LOCALE (BCP-47)	CUSTOM KEYWORD	KEYWORD VERIFICATION
Chinese (Mandarin, Simplified)	zh-CN	Yes	Yes
English (United States)	en-US	Yes	Yes
Japanese (Japan)	ja-JP	No	Yes
Portuguese (Brazil)	pt-BR	No	Yes

Next steps

- Create a free Azure account
- See how to recognize speech in C#

Speech service supported regions

11/23/2021 • 5 minutes to read • Edit Online

The Speech service allows your application to convert audio to text, perform speech translation, and covert text to speech. The service is available in multiple regions with unique endpoints for the Speech SDK and REST APIs.

The Speech portal, where you can perform custom configurations to your speech experience for all regions, is available at speech.microsoft.com.

Keep in mind the following points when considering regions:

- If your application uses a Speech SDK, you provide the region identifier, such as `westus`, when creating a speech configuration. Make sure the region matches the region of your subscription.
- If your application uses one of the Speech service's REST APIs, the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

> **NOTE**
>
> Speech Services doesn't store/process customer data outside the region the customer deploys the service instance in.

Speech SDK

In the Speech SDK, the region is specified as a parameter (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

Speech-to-text, text-to-speech, and translation

The Speech service is available in these regions for **speech recognition**, **text-to-speech**, and **translation**:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`

GEOGRAPHY	REGION	REGION IDENTIFIER
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

If you plan to train a custom model with audio data, use one of the regions with dedicated hardware for faster training. You can use the REST API to copy the fully trained model to another region later.

Intent recognition

Available regions for **intent recognition** via the Speech SDK are the following:

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 in these regions:

GLOBAL REGION	REGION	REGION IDENTIFIER
North America	West US	`westus`
North America	West US 2	`westus2`
North America	East US	`eastus`
North America	East US 2	`eastus2`
North America	West Central US	`westcentralus`
North America	South Central US	`southcentralus`
Europe	West Europe	`westeurope`
Europe	North Europe	`northeurope`
Asia	East Asia	`eastasia`
Asia	Southeast Asia	`southeastasia`
India	Central India	`centralindia`

Speaker Recognition

Available regions for **Speaker Recognition** are the following:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Central India	`centralindia`
Australia	Australia East	`australiaeast`
Europe	North Europe	`northeurope`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	West Europe	`westeurope`
Europe	UK South	`uksouth`

REST APIs

The Speech service also exposes REST endpoints for speech-to-text, text-to-speech and speaker recognition requests.

Speech-to-text

For speech-to-text reference documentation, see Speech-to-text REST API.

The endpoint for the REST API has this format:

```
https://<REGION_IDENTIFIER>.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

> **NOTE**
>
> The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is:
>
> ```
> https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
> ```
>
> .

Text-to-speech

For text-to-speech reference documentation, see Text-to-speech REST API.

Neural and standard voices

Use this table to determine **availability of neural and standard voices** by region/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`
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/v`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`

REGION	ENDPOINT
South Central US	`https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v`
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`

> **TIP**
>
> Voices in preview are only available in these 3 regions: East US, West Europe and Southeast Asia.

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	`https://australiaeast.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Brazil South	`https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
Canada Central	`https://canadacentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Central US	`https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East Asia	`https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US	`https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US 2	`https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
France Central	`https://francecentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
India Central	`https://centralindia.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
Japan East	`https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
Japan West	`https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

REGION	ENDPOINT
Korea Central	`https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
North Central US	`https://northcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
North Europe	`https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
South Central US	`https://southcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
Southeast Asia	`https://southeastasia.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
UK South	`https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West Europe	`https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US	`https://westus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US 2	`https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

Custom Neural Voice

The following table details regional support for Custom Neural Voice features.

FEATURE	SUPPORTED REGIONS
Voice model hosting	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Realtime Characters	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Long Audio Characters	East US, West Europe, Southeast Asia, UK South, India Central
Custom Neural Training	East US, Southeast Asia, UK South

Long audio API

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.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`

Speaker Recognition

For speaker recognition reference documentation, see Speaker Recognition REST API. Available regions are the same as Speaker Recognition SDK.

Ingestion Client for the Speech service

11/23/2021 • 2 minutes to read • Edit Online

The Ingestion Client is a tool released on GitHub that enables customers to transcribe audio files through Speech services quickly with little or no development effort. It works by wiring up a dedicated Azure storage account to custom Azure Functions that use either the REST API or the SDK in a serverless fashion to pass transcription requests to the service.

Architecture

The tool helps those customers that want to get an idea of the quality of the transcript without making development investments up front. The tool connects a few resources to transcribe audio files that land in the dedicated Azure Storage container.

Internally, the tool uses our V3.0 Batch API or SDK, and follows best practices to handle scale-up, retries and failover. The following schematic describes the resources and connections.

The Getting Started Guide for the Ingestion Client describes how to setup and use the tool.

> **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.

Both, the Microsoft Speech SDK and the Speech-to-text REST API v3.0, can be used to obtain transcripts. The decision does impact overall costs as it is explained in the guide.

> **TIP**
>
> You can use the tool and resulting solution in production to process a high volume of audio.

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

- Get your Speech service trial subscription
- Learn more about Speech SDK
- Learn about the Speech CLI tool

Universal Phone Set

11/23/2021 • 2 minutes to read • Edit Online

This page provides the Universal Phone Set used to customize the pronunciation of words in Custom Speech. See Customize pronunciation to learn more about customizing pronunciation using phone sets.

English (United States)

Consonants

UPS PHONES	IPA	EXAMPLE
B	b	big
CH	t.ʃ / ʧ	chin
D	d	dig
DH	ð	then
DX	ɾ	butter
F	f	fork
G	g	gut
H	h	help
JH	d.ʒ / ʤ	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

UPS PHONES	IPA	EXAMPLE
TH	θ	thin
V	v	vat
W	w	with
J	j	yard
Z	z	zap
ZH	ʒ	pleasure

Vowels

UPS PHONE	IPA	EXAMPLE
AA	ɑ	father
AE	æ	cat
AH	ʌ	cut
AO	ɔ	dog
AOX	ɔ.ə	four
AU	ɑ.ʊ	foul
AX	ə	ago
AX rho	ɚ	minor
AI	ɑ.ɪ	bite
EH	ɛ	pet
EHX	ɛ.ə	stairs
ER	ɜ	fur
ER rho	ɝ	urban
EI	e.ɪ	ate
IH	ɪ	fill
I	i	feel
O	o	go

UPS PHONE	IPA	EXAMPLE
OI	ɔ.ɪ	toy
OWX	o.ə	boa
Q	ɒ	hot
UH	ʊ	book
U	u	too, blue
UWX	u.ə	lure

Next steps

- Customizing Pronunciation
- Prepare data for Custom Speech

What is neural text-to-speech?

11/23/2021 • 5 minutes to read • Edit Online

Microsoft neural text-to-speech uses deep neural networks to make the voices of computers nearly indistinguishable from recordings of people. With the human-like natural prosody and clear articulation of words, neural text-to-speech has significantly reduced listening fatigue when you 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. Microsoft neural text-to-speech capability does prosody prediction and voice synthesis simultaneously, uses deep neural networks to overcome the limits of traditional text-to-speech systems in matching the patterns of stress and intonation in spoken language, and synthesizes the units of speech into a computer voice. The result is a more fluid and natural-sounding voice.

In this overview, you learn about the benefits and capabilities of the text-to-speech service, which enables your applications, tools, or devices to convert text into human-like synthesized speech. Use human-like neural voices, or create a custom voice unique to your product or brand. For a full list of supported voices, languages, and locales, see supported languages.

This documentation contains the following article types:

- **Quickstarts** are getting-started instructions to guide you through making requests to the service.
- **How-to guides** contain instructions for using the service in more specific or customized ways.
- **Concepts** provide in-depth explanations of the service functionality and features.
- **Tutorials** are longer guides that show you how to use the service as a component in broader business solutions.

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

Core features

- Speech synthesis - Use the Speech SDK or REST API to convert text-to-speech using platform neural voices or custom neural voices.
- Asynchronous synthesis of long audio - Use the Long Audio API to asynchronously synthesize text-to-speech files longer than 10 minutes (for example audio books or lectures). Unlike synthesis performed using the Speech SDK or speech-to-text REST API, responses aren't returned in real time. The expectation is that requests are sent asynchronously, responses are polled for, and that the synthesized audio is downloaded when made available from the service.
- Platform neural voices - Deep neural networks are used to overcome the limits of traditional speech synthesis with regard to stress and intonation in spoken language. Prosody prediction and voice synthesis are performed simultaneously, which results in more fluid and natural-sounding outputs. Neural voices can be used to make interactions with chatbots and voice assistants more natural and engaging, convert digital texts such as e-books into audiobooks, and enhance in-car navigation systems. With the human-like natural prosody and clear articulation of words, neural voices significantly reduce

listening fatigue when you interact with AI systems. For a full list of platform neural voices, see supported languages.

- Fine-tune TTS output with SSML - Speech Synthesis Markup Language (SSML) is an XML-based markup language used to customize text-to-speech outputs. With SSML, you can not only adjust pitch, add pauses, improve pronunciation, change speaking rate, adjust volume, and attribute multiple voices to a single document, but also define your own lexicons or switch to different speaking styles. With the multi-lingual voices, you can also adjust the speaking languages via SSML. See how to use SSML to fine-tune the voice output for your scenario.
- Visemes - Visemes are the key poses in observed speech, including the position of the lips, jaw and tongue when producing a particular phoneme. Visemes have a strong correlation with voices and phonemes. Using viseme events in Speech SDK, you can generate facial animation data, which can be used to animate faces in lip-reading communication, education, entertainment, and customer service. Viseme is currently only supported for the `en-US` English (United States) neural voices.

Get started

See the quickstart to get started with text-to-speech. The text-to-speech service is available via the Speech SDK, the REST API, and the Speech CLI

Sample code

Sample code for text-to-speech is available on GitHub. These samples cover text-to-speech conversion in most popular programming languages.

- Text-to-speech samples (SDK)
- Text-to-speech samples (REST)

Customization

In addition to neural voices, you can create and fine-tune custom voices unique to your product or brand. All it takes to get started are a handful of audio files and the associated transcriptions. For more information, see Get started with Custom Neural Voice

Pricing note

When using the text-to-speech service, you are billed for each character that is converted to speech, including punctuation. While the SSML document itself is not billable, optional elements that are used to adjust how the text is converted to speech, like phonemes and pitch, are counted as billable characters. Here's a list of what's billable:

- Text passed to the text-to-speech service in the SSML body of the request
- All markup within the text field of the request body in the SSML format, except for `<speak>` and `<voice>` tags
- Letters, punctuation, spaces, tabs, markup, and all white-space characters
- Every code point defined in Unicode

For detailed information, see Pricing.

> **IMPORTANT**
>
> Each Chinese, Japanese, and Korean language character is counted as two characters for billing.

Migrate to Neural Voice

We are retiring the standard voices on **31st August 2024** and they will no longer be supported after that date. The announcement has been sent out to all existing Speech subscriptions before **31st August 2021**. During the retiring period (**31st August 2021- 31st August 2024**), existing standard voice users can continue to use their standard voices, all new users/new speech resources should move to the neural voices.

Action required

1. Review the price structure and listen to the neural voice samples at the bottom of the page to determine the right voice for your business needs.
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 by 31 August 2024. **Starting 1st September 2024**, standard voices will no longer be supported, please use neural voices for your speech synthesis request, on cloud or on prem. For on-prem container, please use the neural voice containers and follow the instructions.

Reference docs

- Speech SDK
- REST API: Text-to-speech

Next steps

- Get a free Speech service subscription
- Get the Speech SDK

Get started with text-to-speech

11/23/2021 • 53 minutes to read • Edit Online

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK. You start by doing basic configuration and synthesis, and move on to more advanced examples for custom application development including:

- Getting responses as in-memory streams
- Customizing output sample rate and bit rate
- Submitting synthesis requests using SSML (speech synthesis markup language)
- Using neural voices

Skip to samples on GitHub

If you want to skip straight to sample code, see the C# quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to 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.

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

Create a speech configuration

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

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

There are a few ways that you can initialize a `SpeechConfig`:

- With a subscription: pass in a key and the associated location/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 location/region.

In this example, you create a `SpeechConfig` using a speech key and location/region. Get these credentials by following steps in Try the Speech service for free. You also create some basic boilerplate code to use for the rest of this article, which you modify for different customizations.

```
public class Program
{
    static async Task Main()
    {
        await SynthesizeAudioAsync();
    }

    static async Task SynthesizeAudioAsync()
    {
        var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    }
}
```

Select synthesis language and voice

The Azure Text to Speech service supports more than 250 voices and over 70 languages and variants. You can get the full list, or try them in text to speech demo. Specify the language or voice of `SpeechConfig` to match your input text and use the wanted voice.

```
static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    // Note: if only language is set, the default voice of that language is chosen.
    config.SpeechSynthesisLanguage = "<your-synthesis-language>"; // e.g. "de-DE"
    // The voice setting will overwrite language setting.
    // The voice setting will not overwrite the voice element in input SSML.
    config.SpeechSynthesisVoiceName = "<your-wanted-voice>";
}
```

Synthesize speech to a file

Next, you create a `SpeechSynthesizer` object, which executes text-to-speech conversions and outputs to speakers, files, or other output streams. The `SpeechSynthesizer` accepts as params the `SpeechConfig` object created in the previous step, and an `AudioConfig` object that specifies how output results should be handled.

To start, create an `AudioConfig` to automatically write the output to a `.wav` file, using the `FromWavFileOutput()` function, and 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.

```
static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    using var audioConfig = AudioConfig.FromWavFileOutput("path/to/write/file.wav");
}
```

Next, instantiate a `SpeechSynthesizer` with another `using` statement. Pass your `config` object and the `audioConfig` object as params. Then, executing speech synthesis and writing to a file is as simple as running `SpeakTextAsync()` with a string of text.

```
static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    using var audioConfig = AudioConfig.FromWavFileOutput("path/to/write/file.wav");
    using var synthesizer = new SpeechSynthesizer(config, audioConfig);
    await synthesizer.SpeakTextAsync("A simple test to write to a file.");
}
```

Run the program, and a synthesized `.wav` file is written to the location you specified. This is a good example of the most basic usage, but 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

In some cases, you may want to directly output synthesized speech directly to a speaker. To do this, omit the `AudioConfig` parameter when creating the `SpeechSynthesizer` in the example above. This synthesizes to the current active output device.

```
static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    using var synthesizer = new SpeechSynthesizer(config);
    await synthesizer.SpeakTextAsync("Synthesizing directly to speaker output.");
}
```

Get result as an in-memory stream

For many scenarios in speech application development, you likely need the resulting audio data as an in-memory stream rather than directly writing to a file. This will allow you to build custom behavior including:

- Abstract the resulting byte array as a seek-able stream for custom downstream services.
- Integrate the result with other API's or services.
- Modify the audio data, write custom `.wav` headers, etc.

It's simple to make this change from the previous example. First, remove the `AudioConfig` block, as you will manage the output behavior manually from this point onward for increased control. Then pass `null` for the `AudioConfig` in the `SpeechSynthesizer` constructor.

> **NOTE**
> Passing `null` for the `AudioConfig`, rather than omitting it like in the speaker output example above, will not play the audio by default on the current active output device.

This time, you save the result to a `SpeechSynthesisResult` variable. The `AudioData` property contains a `byte []` of the output data. You can work with this `byte []` 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.

```
static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    using var synthesizer = new SpeechSynthesizer(config, null);

    var result = await synthesizer.SpeakTextAsync("Getting the response as an in-memory stream.");
    using var stream = AudioDataStream.FromResult(result);
}
```

From here you can implement any custom behavior using the resulting `stream` object.

Customize audio format

The following section shows how to 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` of type `SpeechSynthesisOutputFormat`, which you use to select the output format. See the reference docs for a list of audio formats that are available.

There are various options for different file types depending on your requirements. Note that by definition, raw formats like `Raw24Khz16BitMonoPcm` do not include audio headers. Use raw formats only when you know your downstream implementation can decode a raw bitstream, or if you plan on manually building headers based on bit-depth, sample-rate, number of channels, etc.

In this example, you specify a high-fidelity RIFF format `Riff24Khz16BitMonoPcm` by setting the `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.

```
static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    config.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm);

    using var synthesizer = new SpeechSynthesizer(config, null);
    var result = await synthesizer.SpeakTextAsync("Customizing audio output format.");

    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

Speech Synthesis Markup Language (SSML) allows you to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output by submitting your requests from an XML schema. This section shows an example of changing the voice, but 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 config in your root project directory, in this example `ssml.xml`. The root element is always `<speak>`, and wrapping the text in a `<voice>` element allows you to change the voice using the `name` param. See the full list of supported **neural** voices.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-ChristopherNeural">
    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 config as a string using `File.ReadAllText()`. From here, the result object is exactly the same as previous examples.

> **NOTE**
> If you're using Visual Studio, your build config likely will not 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*.

```
public static async Task SynthesizeAudioAsync()
{
    var config = SpeechConfig.FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    using var synthesizer = new SpeechSynthesizer(config, 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 the `SpeechConfig` by using `SpeechConfig.SpeechSynthesisVoiceName = "en-US-ChristopherNeural";`

Get facial pose events

Speech can be a good way to drive the animation of facial expressions. Often visemes are used to represent the key poses in observed speech, such as the position of the lips, jaw and tongue when producing a particular phoneme. You can subscribe the viseme event in Speech SDK. Then, you can apply viseme events to animate the face of a character as speech audio plays. Learn how to get viseme events.

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK. You start by doing basic configuration and synthesis, and move on to more advanced examples for custom application development including:

- Getting responses as in-memory streams
- Customizing output sample rate and bit rate
- Submitting synthesis requests using SSML (speech synthesis markup language)
- Using neural voices

Skip to samples on GitHub

If you want to skip straight to sample code, see the C++ quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, use the following instructions:

- Linux
- macOS
- Windows

Import dependencies

To run the examples in this article, include the following import and `using` statements at the top of your script.

```
#include <iostream>
#include <fstream>
#include <string>
#include <speechapi_cxx.h>

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

Create a speech configuration

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

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

There are a few ways that you can initialize a `SpeechConfig`:

- With a subscription: pass in a key and the associated location/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 location/region.

In this example, you create a `SpeechConfig` using a subscription key and region. Get these credentials by following steps in Try the Speech service for free. You also create some basic boilerplate code to use for the rest of this article, which you modify for different customizations.

```
int wmain()
{
    try
    {
        synthesizeSpeech();
    }
    catch (exception e)
    {
        cout << e.what();
    }
    return 0;
}

void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
}
```

Select synthesis language and voice

The Azure Text to Speech service supports more than 250 voices and over 70 languages and variants. You can get the full list, or try them in text to speech demo. Specify the language or voice of `SpeechConfig` to match your input text and use the wanted voice.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    // Note: if only language is set, the default voice of that language is chosen.
    config->SetSpeechSynthesisLanguage("<your-synthesis-language>"); // e.g. "de-DE"
    // The voice setting will overwrite language setting.
    // The voice setting will not overwrite the voice element in input SSML.
    config->SetSpeechSynthesisVoiceName("<your-wanted-voice>");
}
```

Synthesize speech to a file

Next, you create a `SpeechSynthesizer` object, which executes text-to-speech conversions and outputs to speakers, files, or other output streams. The `SpeechSynthesizer` accepts as params the `SpeechConfig` object created in the previous step, and an `AudioConfig` object that specifies how output results should be handled.

To start, create an `AudioConfig` to automatically write the output to a `.wav` file, using the `FromWavFileOutput()` function.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    auto audioConfig = AudioConfig::FromWavFileOutput("path/to/write/file.wav");
}
```

Next, instantiate a `SpeechSynthesizer`, passing your `config` object and the `audioConfig` object as params. Then, executing speech synthesis and writing to a file is as simple as running `SpeakTextAsync()` with a string of text.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    auto audioConfig = AudioConfig::FromWavFileOutput("path/to/write/file.wav");
    auto synthesizer = SpeechSynthesizer::FromConfig(config, audioConfig);
    auto result = synthesizer->SpeakTextAsync("A simple test to write to a file.").get();
}
```

Run the program, and a synthesized `.wav` file is written to the location you specified. This is a good example of the most basic usage, but 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

In some cases, you may want to directly output synthesized speech directly to a speaker. To do this, omit the `AudioConfig` parameter when creating the `SpeechSynthesizer` in the example above. This synthesizes to the current active output device.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    auto synthesizer = SpeechSynthesizer::FromConfig(config);
    auto result = synthesizer->SpeakTextAsync("Synthesizing directly to speaker output.").get();
}
```

Get result as an in-memory stream

For many scenarios in speech application development, you likely need the resulting audio data as an in-memory stream rather than directly writing to a file. This will allow you to build custom behavior including:

- Abstract the resulting byte array as a seek-able stream for custom downstream services.
- Integrate the result with other API's or services.
- Modify the audio data, write custom `.wav` headers, etc.

It's simple to make this change from the previous example. First, remove the `AudioConfig`, as you will manage the output behavior manually from this point onward for increased control. Then pass `NULL` for the `AudioConfig` in the `SpeechSynthesizer` constructor.

> **NOTE**
>
> Passing `NULL` for the `AudioConfig`, rather than omitting it like in the speaker output example above, will not play the audio by default on the current active output device.

This time, you save the result to a `SpeechSynthesisResult` variable. The `GetAudioData` getter returns a `byte []` of the output data. You can work with this `byte []` 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.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    auto synthesizer = SpeechSynthesizer::FromConfig(config, NULL);

    auto result = synthesizer->SpeakTextAsync("Getting the response as an in-memory stream.").get();
    auto stream = AudioDataStream::FromResult(result);
}
```

From here you can implement any custom behavior using the resulting `stream` object.

Customize audio format

The following section shows how to 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` of type `SpeechSynthesisOutputFormat`, which you use to select the output format. See the reference docs for a list of audio formats that are available.

There are various options for different file types depending on your requirements. Note that by definition, raw formats like `Raw24Khz16BitMonoPcm` do not include audio headers. Use raw formats only when you know your downstream implementation can decode a raw bitstream, or if you plan on manually building headers based on bit-depth, sample-rate, number of channels, etc.

In this example, you specify a high-fidelity RIFF format `Riff24Khz16BitMonoPcm` by setting the `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.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    config->SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat::Riff24Khz16BitMonoPcm);

    auto synthesizer = SpeechSynthesizer::FromConfig(config, NULL);
    auto result = synthesizer->SpeakTextAsync("A simple test to write to a file.").get();

    auto stream = AudioDataStream::FromResult(result);
    stream->SaveToWavFileAsync("path/to/write/file.wav").get();
}
```

Running your program again will write a `.wav` file to the specified path.

Use SSML to customize speech characteristics

Speech Synthesis Markup Language (SSML) allows you to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output by submitting your requests from an XML schema. This section

shows an example of changing the voice, but 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 config in your root project directory, in this example `ssml.xml`. The root element is always `<speak>`, and wrapping the text in a `<voice>` element allows you to change the voice using the `name` param. See the full list of supported **neural** voices.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-ChristopherNeural">
    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 config as a string. From here, the result object is exactly the same as previous examples.

```
void synthesizeSpeech()
{
    auto config = SpeechConfig::FromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");
    auto synthesizer = SpeechSynthesizer::FromConfig(config, NULL);

    std::ifstream file("./ssml.xml");
    std::string ssml, line;
    while (std::getline(file, line))
    {
        ssml += line;
        ssml.push_back('\n');
    }
    auto result = synthesizer->SpeakSsmlAsync(ssml).get();

    auto stream = AudioDataStream::FromResult(result);
    stream->SaveToWavFileAsync("path/to/write/file.wav").get();
}
```

NOTE

To change the voice without using SSML, you can set the property on the `SpeechConfig` by using `SpeechConfig.SetSpeechSynthesisVoiceName("en-US-ChristopherNeural")`

Get facial pose events

Speech can be a good way to drive the animation of facial expressions. Often visemes are used to represent the key poses in observed speech, such as the position of the lips, jaw and tongue when producing a particular phoneme. You can subscribe the viseme event in Speech SDK. Then, you can apply viseme events to animate the face of a character as speech audio plays. Learn how to get viseme events.

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK.

Skip to samples on GitHub

If you want to skip straight to sample code, see the Go quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK for Go.

Text-to-speech to speaker

Use the following code sample to run speech synthesis to your default audio output device. Replace the variables `subscription` and `region` with your speech key and location/region. Running the script will speak your input text to default speaker.

```
package main

import (
 "bufio"
 "fmt"
 "os"
 "strings"
 "time"

 "github.com/Microsoft/cognitive-services-speech-sdk-go/audio"
 "github.com/Microsoft/cognitive-services-speech-sdk-go/common"
 "github.com/Microsoft/cognitive-services-speech-sdk-go/speech"
)

func synthesizeStartedHandler(event speech.SpeechSynthesisEventArgs) {
 defer event.Close()
 fmt.Println("Synthesis started.")
}

func synthesizingHandler(event speech.SpeechSynthesisEventArgs) {
 defer event.Close()
 fmt.Printf("Synthesizing, audio chunk size %d.\n", len(event.Result.AudioData))
}

func synthesizedHandler(event speech.SpeechSynthesisEventArgs) {
 defer event.Close()
 fmt.Printf("Synthesized, audio length %d.\n", len(event.Result.AudioData))
}

func cancelledHandler(event speech.SpeechSynthesisEventArgs) {
 defer event.Close()
 fmt.Println("Received a cancellation.")
}

func main() {
    subscription := "<paste-your-speech-key-here>"
    region := "<paste-your-speech-location/region-here>"

 audioConfig, err := audio.NewAudioConfigFromDefaultSpeakerOutput()
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer audioConfig.Close()
 config, err := speech.NewSpeechConfigFromSubscription(subscription, region)
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer config.Close()
 speechSynthesizer, err := speech.NewSpeechSynthesizerFromConfig(config, audioConfig)
 if err != nil {
```

```
    fmt.Println("Got an error: ", err)
    return
  }
  defer speechSynthesizer.Close()

  speechSynthesizer.SynthesisStarted(synthesizeStartedHandler)
  speechSynthesizer.Synthesizing(synthesizingHandler)
  speechSynthesizer.SynthesisCompleted(synthesizedHandler)
  speechSynthesizer.SynthesisCanceled(cancelledHandler)

  for {
    fmt.Printf("Enter some text that you want to speak, or enter empty text to exit.\n> ")
    text, _ := bufio.NewReader(os.Stdin).ReadString('\n')
    text = strings.TrimSuffix(text, "\n")
    if len(text) == 0 {
      break
    }

    task := speechSynthesizer.SpeakTextAsync(text)
    var outcome speech.SpeechSynthesisOutcome
    select {
    case outcome = <-task:
    case <-time.After(60 * time.Second):
      fmt.Println("Timed out")
      return
    }
    defer outcome.Close()
    if outcome.Error != nil {
      fmt.Println("Got an error: ", outcome.Error)
      return
    }

    if outcome.Result.Reason == common.SynthesizingAudioCompleted {
      fmt.Printf("Speech synthesized to speaker for text [%s].\n", text)
    } else {
      cancellation, _ := speech.NewCancellationDetailsFromSpeechSynthesisResult(outcome.Result)
      fmt.Printf("CANCELED: Reason=%d.\n", cancellation.Reason)

      if cancellation.Reason == common.Error {
        fmt.Printf("CANCELED: ErrorCode=%d\nCANCELED: ErrorDetails=[%s]\nCANCELED: Did you update the
subscription info?\n",
          cancellation.ErrorCode,
          cancellation.ErrorDetails)
      }
    }
  }
}
```

Run the following commands to create a `go.mod` file that links to components hosted on GitHub.

```
go mod init quickstart
go get github.com/Microsoft/cognitive-services-speech-sdk-go
```

Now build and run the code.

```
go build
go run quickstart
```

See the reference docs for detailed information on the `SpeechConfig` and `SpeechSynthesizer` classes.

Text-to-speech to in-memory stream

For many scenarios in speech application development, you likely need the resulting audio data as an in-

memory stream rather than directly writing to a file. This will allow you to build custom behavior including:

- Abstract the resulting byte array as a seek-able stream for custom downstream services.
- Integrate the result with other API's or services.
- Modify the audio data, write custom `.wav` headers, etc.

It's simple to make this change from the previous example. First, remove the `AudioConfig`, as you will manage the output behavior manually from this point onward for increased control. Then pass `nil` for the `AudioConfig` in the `SpeechSynthesizer` constructor.

> **NOTE**
> Passing `nil` for the `AudioConfig`, rather than omitting it like in the speaker output example above, will not play the audio by default on the current active output device.

This time, you save the result to a `SpeechSynthesisResult` variable. The `AudioData` property returns a `[]byte` of the output data. You can work with this `[]byte` manually, or you can use the `AudioDataStream` class to manage the in-memory stream. In this example, you use the `NewAudioDataStreamFromSpeechSynthesisResult()` static function to get a stream from the result.

Replace the variables `subscription` and `region` with your speech key and location/region.

```
package main

import (
 "bufio"
 "fmt"
 "io"
 "os"
 "strings"
 "time"

 "github.com/Microsoft/cognitive-services-speech-sdk-go/speech"
)

func main(subscription string, region string) {
    subscription := "<paste-your-speech-key-here>"
    region := "<paste-your-speech-location/region-here>"

 config, err := speech.NewSpeechConfigFromSubscription(subscription, region)
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer config.Close()
 speechSynthesizer, err := speech.NewSpeechSynthesizerFromConfig(config, nil)
 if err != nil {
  fmt.Println("Got an error: ", err)
  return
 }
 defer speechSynthesizer.Close()

 speechSynthesizer.SynthesisStarted(synthesizeStartedHandler)
 speechSynthesizer.Synthesizing(synthesizingHandler)
 speechSynthesizer.SynthesisCompleted(synthesizedHandler)
 speechSynthesizer.SynthesisCanceled(cancelledHandler)

 for {
  fmt.Printf("Enter some text that you want to speak, or enter empty text to exit.\n> ")
  text, _ := bufio.NewReader(os.Stdin).ReadString('\n')
  text = strings.TrimSuffix(text, "\n")
  if len(text) == 0 {
```

```
    break
   }

   // StartSpeakingTextAsync sends the result to channel when the synthesis starts.
   task := speechSynthesizer.StartSpeakingTextAsync(text)
   var outcome speech.SpeechSynthesisOutcome
   select {
   case outcome = <-task:
   case <-time.After(60 * time.Second):
    fmt.Println("Timed out")
    return
   }
   defer outcome.Close()
   if outcome.Error != nil {
    fmt.Println("Got an error: ", outcome.Error)
    return
   }

   // in most case we want to streaming receive the audio to lower the latency,
   // we can use AudioDataStream to do so.
   stream, err := speech.NewAudioDataStreamFromSpeechSynthesisResult(outcome.Result)
   defer stream.Close()
   if err != nil {
    fmt.Println("Got an error: ", err)
    return
   }

   var all_audio []byte
   audio_chunk := make([]byte, 2048)
   for {
    n, err := stream.Read(audio_chunk)

    if err == io.EOF {
     break
    }

    all_audio = append(all_audio, audio_chunk[:n]...)
   }

   fmt.Printf("Read [%d] bytes from audio data stream.\n", len(all_audio))
  }
 }
```

Run the following commands to create a `go.mod` file that links to components hosted on GitHub.

```
go mod init quickstart
go get github.com/Microsoft/cognitive-services-speech-sdk-go
```

Now build and run the code.

```
go build
go run quickstart
```

See the reference docs for detailed information on the `SpeechConfig` and `SpeechSynthesizer` classes.

Use SSML to customize speech characteristics

Speech Synthesis Markup Language (SSML) allows you to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output by submitting your requests from an XML schema. This section shows an example of changing the voice, but 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 config in your root project directory, in this example `ssml.xml`. The root element is always `<speak>`, and wrapping the text in a `<voice>` element allows you to change the voice using the `name` param. See the full list of supported **neural** voices.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-ChristopherNeural">
    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 config as a string. From here, the result object is exactly the same as previous examples.

> **NOTE**
> To change the voice without using SSML, you can set the property on the `SpeechConfig` by using `speechConfig.SetSpeechSynthesisVoiceName("en-US-ChristopherNeural")`

Get facial pose events

Speech can be a good way to drive the animation of facial expressions. Often visemes are used to represent the key poses in observed speech, such as the position of the lips, jaw and tongue when producing a particular phoneme. You can subscribe the viseme event in Speech SDK. Then, you can apply viseme events to animate the face of a character as speech audio plays. Learn how to get viseme events.

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK. You start by doing basic configuration and synthesis, and move on to more advanced examples for custom application development including:

- Getting responses as in-memory streams
- Customizing output sample rate and bit rate
- Submitting synthesis requests using SSML (speech synthesis markup language)
- Using neural voices

Skip to samples on GitHub

If you want to skip straight to sample code, see the Java quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, use the following instructions:

- Java Runtime
- Android

Import dependencies

To run the examples in this article, include the following import statements at the top of your script.

```
import com.microsoft.cognitiveservices.speech.AudioDataStream;
import com.microsoft.cognitiveservices.speech.SpeechConfig;
import com.microsoft.cognitiveservices.speech.SpeechSynthesizer;
import com.microsoft.cognitiveservices.speech.SpeechSynthesisOutputFormat;
import com.microsoft.cognitiveservices.speech.SpeechSynthesisResult;
import com.microsoft.cognitiveservices.speech.audio.AudioConfig;

import java.io.*;
import java.util.Scanner;
```

Create a speech configuration

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

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

There are a few ways that you can initialize a `SpeechConfig`:

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

In this example, you create a `SpeechConfig` using a speech key and location/region. Get these credentials by following steps in Try the Speech service for free. You also create some basic boilerplate code to use for the rest of this article, which you modify for different customizations.

```
public class Program
{
    public static void main(String[] args) {
        SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-
your-speech-location/region-here>");
    }
}
```

Select synthesis language and voice

The Azure Text to Speech service supports more than 250 voices and over 70 languages and variants. You can get the full list, or try them in text to speech demo. Specify the language or voice of `SpeechConfig` to match your input text and use the wanted voice.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    // Note: if only language is set, the default voice of that language is chosen.
    config.setSpeechSynthesisLanguage("<your-synthesis-language>"); // e.g. "de-DE"
    // The voice setting will overwrite language setting.
    // The voice setting will not overwrite the voice element in input SSML.
    config.setSpeechSynthesisVoiceName("<your-wanted-voice>");
}
```

Synthesize speech to a file

Next, you create a `SpeechSynthesizer` object, which executes text-to-speech conversions and outputs to speakers, files, or other output streams. The `SpeechSynthesizer` accepts as params the `SpeechConfig` object created in the previous step, and an `AudioConfig` object that specifies how output results should be handled.

To start, create an `AudioConfig` to automatically write the output to a `.wav` file using the `fromWavFileOutput()` static function.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    AudioConfig audioConfig = AudioConfig.fromWavFileOutput("path/to/write/file.wav");
}
```

Next, instantiate a `SpeechSynthesizer` passing your `speechConfig` object and the `audioConfig` object as params. Then, executing speech synthesis and writing to a file is as simple as running `SpeakText()` with a string of text.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    AudioConfig audioConfig = AudioConfig.fromWavFileOutput("path/to/write/file.wav");

    SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, audioConfig);
    synthesizer.SpeakText("A simple test to write to a file.");
}
```

Run the program, and a synthesized `.wav` file is written to the location you specified. This is a good example of the most basic usage, but 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

In some cases, you may want to directly output synthesized speech directly to a speaker. To do this, instantiate the `AudioConfig` using the `fromDefaultSpeakerOutput()` static function. This outputs to the current active output device.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    AudioConfig audioConfig = AudioConfig.fromDefaultSpeakerOutput();

    SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, audioConfig);
    synthesizer.SpeakText("Synthesizing directly to speaker output.");
}
```

Get result as an in-memory stream

For many scenarios in speech application development, you likely need the resulting audio data as an in-memory stream rather than directly writing to a file. This will allow you to build custom behavior including:

- Abstract the resulting byte array as a seek-able stream for custom downstream services.
- Integrate the result with other API's or services.
- Modify the audio data, write custom `.wav` headers, etc.

It's simple to make this change from the previous example. First, remove the `AudioConfig` block, as you will manage the output behavior manually from this point onward for increased control. Then pass `null` for the `AudioConfig` in the `SpeechSynthesizer` constructor.

> **NOTE**
> Passing `null` for the `AudioConfig`, rather than omitting it like in the speaker output example above, will not play the audio by default on the current active output device.

This time, you save the result to a `SpeechSynthesisResult` variable. The `SpeechSynthesisResult.getAudioData()` function returns a `byte []` of the output data. You can work with this `byte []` 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.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, null);

    SpeechSynthesisResult result = synthesizer.SpeakText("Getting the response as an in-memory stream.");
    AudioDataStream stream = AudioDataStream.fromResult(result);
    System.out.print(stream.getStatus());
}
```

From here you can implement any custom behavior using the resulting `stream` object.

Customize audio format

The following section shows how to 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` of type `SpeechSynthesisOutputFormat`, which you use to select the output format. See the reference docs for a list of audio formats that are available.

There are various options for different file types depending on your requirements. Note that by definition, raw formats like `Raw24Khz16BitMonoPcm` do not include audio headers. Use raw formats only when you know your downstream implementation can decode a raw bitstream, or if you plan on manually building headers based on bit-depth, sample-rate, number of channels, etc.

In this example, you specify a high-fidelity RIFF format `Riff24Khz16BitMonoPcm` by setting the `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.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");

    // set the output format
    speechConfig.setSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm);

    SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, null);
    SpeechSynthesisResult result = synthesizer.SpeakText("Customizing audio output format.");
    AudioDataStream stream = AudioDataStream.fromResult(result);
    stream.saveToWavFile("path/to/write/file.wav");
}
```

Running your program again will write a `.wav` file to the specified path.

Use SSML to customize speech characteristics

Speech Synthesis Markup Language (SSML) allows you to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output by submitting your requests from an XML schema. This section shows an example of changing the voice, but 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 config in your root project directory, in this example `ssml.xml`. The root element is always `<speak>`, and wrapping the text in a `<voice>` element allows you to change the voice using the `name` param. See the full list of supported **neural** voices.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-ChristopherNeural">
    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 `SpeakText()` function, you use `SpeakSsml()`. This function expects an XML string, so first you create a function to load an XML file and return it as a string.

```
private static String xmlToString(String filePath) {
    File file = new File(filePath);
    StringBuilder fileContents = new StringBuilder((int)file.length());

    try (Scanner scanner = new Scanner(file)) {
        while(scanner.hasNextLine()) {
            fileContents.append(scanner.nextLine() + System.lineSeparator());
        }
        return fileContents.toString().trim();
    } catch (FileNotFoundException ex) {
        return "File not found.";
    }
}
```

From here, the result object is exactly the same as previous examples.

```
public static void main(String[] args) {
    SpeechConfig speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, null);

    String ssml = xmlToString("ssml.xml");
    SpeechSynthesisResult result = synthesizer.SpeakSsml(ssml);
    AudioDataStream stream = AudioDataStream.fromResult(result);
    stream.saveToWavFile("path/to/write/file.wav");
}
```

NOTE

To change the voice without using SSML, you can set the property on the `SpeechConfig` by using `SpeechConfig.setSpeechSynthesisVoiceName("en-US-ChristopherNeural");`

Get facial pose events

Speech can be a good way to drive the animation of facial expressions. Often visemes are used to represent the key poses in observed speech, such as the position of the lips, jaw and tongue when producing a particular phoneme. You can subscribe to viseme events in Speech SDK to get facial animation data, and then apply the data to a character during facial animation. Learn how to get viseme events.

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK. You start by doing basic configuration and synthesis, and move on to more advanced examples for custom application development including:

- Getting responses as in-memory streams
- Customizing output sample rate and bit rate
- Submitting synthesis requests using SSML (speech synthesis markup language)
- Using neural voices

Skip to samples on GitHub

If you want to skip straight to sample code, see the JavaScript quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service resource. If you don't have an account and resource, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK for JavaScript . Depending on your platform, use the following instructions:

- Node.js
- Web browser

Additionally, depending on the target environment use one of the following:

- script
- import
- require

Download and extract the Speech SDK for JavaScript *microsoft.cognitiveservices.speech.sdk.bundle.js* file, and place it in a folder accessible to your HTML file.

```
<script src="microsoft.cognitiveservices.speech.sdk.bundle.js"></script>;
```

> **TIP**
> If you're targeting a web browser, and using the `<script>` tag; the `sdk` prefix is not needed. The `sdk` prefix is an alias used to name the `require` module.

Create a speech configuration

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

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

There are a few ways that you can initialize a `SpeechConfig`:

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

In this example, you create a `SpeechConfig` using a speech key and location/region. Get these credentials by following steps in Try the Speech service for free. You also create some basic boilerplate code to use for the rest of this article, which you modify for different customizations.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
}

synthesizeSpeech();
```

Select synthesis language and voice

The Azure Text to Speech service supports more than 250 voices and over 70 languages and variants. You can get the full list, or try them in text to speech demo. Specify the language or voice of `SpeechConfig` to match your input text and use the wanted voice.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    // Note: if only language is set, the default voice of that language is chosen.
    speechConfig.speechSynthesisLanguage = "<your-synthesis-language>"; // e.g. "de-DE"
    // The voice setting will overwrite language setting.
    // The voice setting will not overwrite the voice element in input SSML.
    speechConfig.speechSynthesisVoiceName = "<your-wanted-voice>";
}

synthesizeSpeech();
```

Synthesize speech to a file

Next, you create a `SpeechSynthesizer` object, which executes text-to-speech conversions and outputs to speakers, files, or other output streams. The `SpeechSynthesizer` accepts as params the `SpeechConfig` object created in the previous step, and an `AudioConfig` object that specifies how output results should be handled.

To start, create an `AudioConfig` to automatically write the output to a `.wav` file using the `fromAudioFileOutput()` static function.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    const audioConfig = sdk.AudioConfig.fromAudioFileOutput("path/to/file.wav");
}
```

Next, instantiate a `SpeechSynthesizer` passing your `speechConfig` object and the `audioConfig` object as params. Then, executing speech synthesis and writing to a file is as simple as running `speakTextAsync()` with a string of text. The result callback is a great place to call `synthesizer.close()`, in fact - this call is needed in order for synthesis to function correctly.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    const audioConfig = sdk.AudioConfig.fromAudioFileOutput("path-to-file.wav");

    const synthesizer = new SpeechSynthesizer(speechConfig, audioConfig);
    synthesizer.speakTextAsync(
        "A simple test to write to a file.",
        result => {
            synthesizer.close();
            if (result) {
                // return result as stream
                return fs.createReadStream("path-to-file.wav");
            }
        },
        error => {
            console.log(error);
            synthesizer.close();
        });
}
```

Run the program, and a synthesized `.wav` file is written to the location you specified. This is a good example of the most basic usage, but 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 (browser only)

In some cases, you may want to directly output synthesized speech directly to a speaker. To do this, instantiate the `AudioConfig` using the `fromDefaultSpeakerOutput()` static function. This outputs to the current active output device.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    const audioConfig = sdk.AudioConfig.fromDefaultSpeakerOutput();

    const synthesizer = new SpeechSynthesizer(speechConfig, audioConfig);
    synthesizer.speakTextAsync(
        "Synthesizing directly to speaker output.",
        result => {
            if (result) {
                synthesizer.close();
                return result.audioData;
            }
        },
        error => {
            console.log(error);
            synthesizer.close();
        });
}
```

Get result as an in-memory stream

For many scenarios in speech application development, you likely need the resulting audio data as an in-memory stream rather than directly writing to a file. This will allow you to build custom behavior including:

- Abstract the resulting byte array as a seek-able stream for custom downstream services.
- Integrate the result with other API's or services.
- Modify the audio data, write custom `.wav` headers, etc.

It's simple to make this change from the previous example. First, remove the `AudioConfig` block, as you will manage the output behavior manually from this point onward for increased control. Then pass `undefined` for the `AudioConfig` in the `SpeechSynthesizer` constructor.

> **NOTE**
> Passing `undefined` for the `AudioConfig`, rather than omitting it like in the speaker output example above, will not play the audio by default on the current active output device.

This time, you save the result to a `SpeechSynthesisResult` variable. The `SpeechSynthesisResult.audioData` property returns an `ArrayBuffer` of the output data, the default browser stream type. For server-code, convert the arrayBuffer to a buffer stream.

The following code works for client-side code.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    const synthesizer = new sdk.SpeechSynthesizer(speechConfig);

    synthesizer.speakTextAsync(
        "Getting the response as an in-memory stream.",
        result => {
            synthesizer.close();
            return result.audioData;
        },
        error => {
            console.log(error);
            synthesizer.close();
        });
}
```

From here you can implement any custom behavior using the resulting `ArrayBuffer` object. The ArrayBuffer is a common type to receive in a browser and play from this format.

For any server-based code, if you need to work with the data as a stream, instead of an ArrayBuffer, you need to convert the object into a stream.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-
speech-location/region-here>");
    const synthesizer = new sdk.SpeechSynthesizer(speechConfig);

    synthesizer.speakTextAsync(
        "Getting the response as an in-memory stream.",
        result => {
            const { audioData } = result;

            synthesizer.close();

            // convert arrayBuffer to stream
            // return stream
            const bufferStream = new PassThrough();
            bufferStream.end(Buffer.from(audioData));
            return bufferStream;
        },
        error => {
            console.log(error);
            synthesizer.close();
        });
}
```

Customize audio format

The following section shows how to customize audio output attributes including:

- Audio file type
- Sample-rate
- Bit-depth

To change the audio format, you use the `speechSynthesisOutputFormat` property on the `SpeechConfig` object. This property expects an `enum` of type `SpeechSynthesisOutputFormat`, which you use to select the output format. See the reference docs for a list of audio formats that are available.

There are various options for different file types depending on your requirements. Note that by definition, raw

formats like `Raw24Khz16BitMonoPcm` do not include audio headers. Use raw formats only when you know your downstream implementation can decode a raw bitstream, or if you plan on manually building headers based on bit-depth, sample-rate, number of channels, etc.

In this example, you specify a high-fidelity RIFF format `Riff24Khz16BitMonoPcm` by setting the `speechSynthesisOutputFormat` on the `SpeechConfig` object. Similar to the example in the previous section, get the audio `ArrayBuffer` data and interact with it.

```
function synthesizeSpeech() {
    const speechConfig = SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-
location/region-here>");

    // Set the output format
    speechConfig.speechSynthesisOutputFormat = SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm;

    const synthesizer = new sdk.SpeechSynthesizer(speechConfig, undefined);
    synthesizer.speakTextAsync(
        "Customizing audio output format.",
        result => {
            // Interact with the audio ArrayBuffer data
            const audioData = result.audioData;
            console.log(`Audio data byte size: ${audioData.byteLength}.`)

            synthesizer.close();
        },
        error => {
            console.log(error);
            synthesizer.close();
        });
}
```

Running your program again will write a `.wav` file to the specified path.

Use SSML to customize speech characteristics

Speech Synthesis Markup Language (SSML) allows you to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output by submitting your requests from an XML schema. This section shows an example of changing the voice, but 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 config in your root project directory, in this example `ssml.xml`. The root element is always `<speak>`, and wrapping the text in a `<voice>` element allows you to change the voice using the `name` param. See the full list of supported **neural** voices.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-ChristopherNeural">
    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 first you create a function to load an XML file and return it as a string.

```
function xmlToString(filePath) {
    const xml = readFileSync(filePath, "utf8");
    return xml;
}
```

For more information on `readFileSync`, see Node.js file system. From here, the result object is exactly the same as previous examples.

```
function synthesizeSpeech() {
    const speechConfig = sdk.SpeechConfig.fromSubscription("<paste-your-speech-key-here>", "<paste-your-speech-location/region-here>");
    const synthesizer = new sdk.SpeechSynthesizer(speechConfig, undefined);

    const ssml = xmlToString("ssml.xml");
    synthesizer.speakSsmlAsync(
        ssml,
        result => {
            if (result.errorDetails) {
                console.error(result.errorDetails);
            } else {
                console.log(JSON.stringify(result));
            }

            synthesizer.close();
        },
        error => {
            console.log(error);
            synthesizer.close();
        });
}
```

NOTE

To change the voice without using SSML, you can set the property on the `SpeechConfig` by using `SpeechConfig.speechSynthesisVoiceName = "en-US-ChristopherNeural";`

Get facial pose events

Speech can be a good way to drive the animation of facial expressions. Often visemes are used to represent the key poses in observed speech, such as the position of the lips, jaw and tongue when producing a particular phoneme. You can subscribe the viseme event in Speech SDK. Then, you can apply viseme events to animate the face of a character as speech audio plays. Learn how to get viseme events.

You can synthesize speech from text using the Speech SDK for Swift and Objective-C.

Prerequisites

The following samples assume that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install Speech SDK and samples

The Cognitive Services Speech SDK contains samples written in in Swift and Objective-C for iOS and Mac. Click a link to see installation instructions for each sample:

- Synthesize speech in Objective-C on macOS
- Synthesize speech in Swift on macOS
- Synthesize speech in Objective-C on iOS
- Synthesize speech in Swift on iOS
- Additional samples for Objective-C on iOS

We also provide an online Speech SDK for Objective-C Reference.

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK. You start by doing basic configuration and synthesis, and move on to more advanced examples for custom application development including:

- Getting responses as in-memory streams
- Customizing output sample rate and bit rate
- Submitting synthesis requests using SSML (speech synthesis markup language)
- Using neural voices

Skip to samples on GitHub

If you want to skip straight to sample code, see the Python quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK.

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

If you're on macOS and run into install issues, you may need to run this command first.

```
python3 -m pip install --upgrade pip
```

After the Speech SDK is installed, include the following import statements at the top of your script.

```
from azure.cognitiveservices.speech import AudioDataStream, SpeechConfig, SpeechSynthesizer,
SpeechSynthesisOutputFormat
from azure.cognitiveservices.speech.audio import AudioOutputConfig
```

Create a speech configuration

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

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

There are a few ways that you can initialize a `SpeechConfig`:

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

In this example, you create a `SpeechConfig` using a speech key and location/region. Get these credentials by following steps in Try the Speech service for free.

```
speech_config = SpeechConfig(subscription="<paste-your-speech-key-here>", region="<paste-your-speech-location/region-here>")
```

Select synthesis language and voice

The Azure Text to Speech service supports more than 250 voices and over 70 languages and variants. You can get the full list, or try them in text to speech demo. Specify the language or voice of `SpeechConfig` to match your input text and use the wanted voice.

```
# Note: if only language is set, the default voice of that language is chosen.
speech_config.speech_synthesis_language = "<your-synthesis-language>" # e.g. "de-DE"
# The voice setting will overwrite language setting.
# The voice setting will not overwrite the voice element in input SSML.
speech_config.speech_synthesis_voice_name ="<your-wanted-voice>"
```

Synthesize speech to a file

Next, you create a `SpeechSynthesizer` object, which executes text-to-speech conversions and outputs to speakers, files, or other output streams. The `SpeechSynthesizer` accepts as params the `SpeechConfig` object created in the previous step, and an `AudioOutputConfig` object that specifies how output results should be handled.

To start, create an `AudioOutputConfig` to automatically write the output to a `.wav` file, using the `filename` constructor param.

```
audio_config = AudioOutputConfig(filename="path/to/write/file.wav")
```

Next, instantiate a `SpeechSynthesizer` by passing your `speech_config` object and the `audio_config` object as params. Then, executing speech synthesis and writing to a file is as simple as running `speak_text_async()` with a string of text.

```
synthesizer = SpeechSynthesizer(speech_config=speech_config, audio_config=audio_config)
synthesizer.speak_text_async("A simple test to write to a file.")
```

Run the program, and a synthesized `.wav` file is written to the location you specified. This is a good example of the most basic usage, but 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

In some cases, you may want to directly output synthesized speech directly to a speaker. To do this, use the example in the previous section, but change the `AudioOutputConfig` by removing the `filename` param, and set `use_default_speaker=True`. This outputs to the current active output device.

```
audio_config = AudioOutputConfig(use_default_speaker=True)
```

Get result as an in-memory stream

For many scenarios in speech application development, you likely need the resulting audio data as an in-memory stream rather than directly writing to a file. This will allow you to build custom behavior including:

- Abstract the resulting byte array as a seek-able stream for custom downstream services.
- Integrate the result with other API's or services.
- Modify the audio data, write custom `.wav` headers, etc.

It's simple to make this change from the previous example. First, remove the `AudioConfig`, as you will manage the output behavior manually from this point onward for increased control. Then pass `None` for the `AudioConfig` in the `SpeechSynthesizer` constructor.

> **NOTE**
> Passing `None` for the `AudioConfig`, rather than omitting it like in the speaker output example above, will not play the audio by default on the current active output device.

This time, you save the result to a `SpeechSynthesisResult` variable. The `audio_data` property contains a `bytes` object of the output data. You can work with this object manually, or you can use the `AudioDataStream` class to manage the in-memory stream. In this example you use the `AudioDataStream` constructor to get a stream from the result.

```
synthesizer = SpeechSynthesizer(speech_config=speech_config, audio_config=None)
result = synthesizer.speak_text_async("Getting the response as an in-memory stream.").get()
stream = AudioDataStream(result)
```

From here you can implement any custom behavior using the resulting `stream` object.

Customize audio format

The following section shows how to customize audio output attributes including:

- Audio file type
- Sample-rate
- Bit-depth

To change the audio format, you use the `set_speech_synthesis_output_format()` function on the `SpeechConfig` object. This function expects an `enum` of type `SpeechSynthesisOutputFormat`, which you use to select the output format. See the reference docs for a list of audio formats that are available.

There are various options for different file types depending on your requirements. Note that by definition, raw formats like `Raw24Khz16BitMonoPcm` do not include audio headers. Use raw formats only when you know your downstream implementation can decode a raw bitstream, or if you plan on manually building headers based on bit-depth, sample-rate, number of channels, etc.

In this example, you specify a high-fidelity RIFF format `Riff24Khz16BitMonoPcm` by setting the `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.

```
speech_config.set_speech_synthesis_output_format(SpeechSynthesisOutputFormat["Riff24Khz16BitMonoPcm"])
synthesizer = SpeechSynthesizer(speech_config=speech_config, audio_config=None)

result = synthesizer.speak_text_async("Customizing audio output format.").get()
stream = AudioDataStream(result)
stream.save_to_wav_file("path/to/write/file.wav")
```

Running your program again will write a customized `.wav` file to the specified path.

Use SSML to customize speech characteristics

Speech Synthesis Markup Language (SSML) allows you to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output by submitting your requests from an XML schema. This section shows an example of changing the voice, but 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 config in your root project directory, in this example `ssml.xml`. The root element is always `<speak>`, and wrapping the text in a `<voice>` element allows you to change the voice using the `name` param. See the full list of supported **neural** voices.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-ChristopherNeural">
    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 `speak_text_async()` function, you use `speak_ssml_async()`. This function expects an XML string, so you first read your SSML config as a string. From here, the result object is exactly the same as previous examples.

> **NOTE**
> If your `ssml_string` contains `ï»¿` at the beginning of the string, you need to strip off the BOM format or the service will return an error. You do this by setting the `encoding` parameter as follows: `open("ssml.xml", "r", encoding="utf-8-sig")`.

```
synthesizer = SpeechSynthesizer(speech_config=speech_config, audio_config=None)

ssml_string = open("ssml.xml", "r").read()
result = synthesizer.speak_ssml_async(ssml_string).get()

stream = AudioDataStream(result)
stream.save_to_wav_file("path/to/write/file.wav")
```

> **NOTE**
> To change the voice without using SSML, you can set the property on the `SpeechConfig` by using `SpeechConfig.speech_synthesis_voice_name = "en-US-ChristopherNeural"`

Get facial pose events

Speech can be a good way to drive the animation of facial expressions. Often visemes are used to represent the key poses in observed speech, such as the position of the lips, jaw and tongue when producing a particular phoneme. You can subscribe the viseme event in Speech SDK. Then, you can apply viseme events to animate the face of a character as speech audio plays. Learn how to get viseme events.

In this quickstart, you learn how to convert text to speech using the Speech service and cURL.

For a high-level look at Text-To-Speech concepts, see the overview article.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Convert text to speech

At a command prompt, run the following command. You will need to insert the following values into the command.

- Your Speech service subscription key.
- Your Speech service region.

You might also wish to change the following values.

- The `X-Microsoft-OutputFormat` header value, which controls the audio output format. You can find a list of supported audio output formats in the text-to-speech REST API reference.
- The output voice. To get a list of voices available for your Speech endpoint, see the next section.
- The output file. In this example, we direct the response from the server into a file named `output.wav`.

```
curl --location --request POST 'https://INSERT_REGION_HERE.tts.speech.microsoft.com/cognitiveservices/v1' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: application/ssml+xml' \
--header 'X-Microsoft-OutputFormat: audio-16khz-128kbitrate-mono-mp3' \
--header 'User-Agent: curl' \
--data-raw '<speak version='\''1.0'\'' xml:lang='\''en-US'\''>
    <voice xml:lang='\''en-US'\'' xml:gender='\''Female'\'' name='\''en-US-JennyNeural'\''>
        my voice is my passport verify me
    </voice>
</speak>' > output.mp3
```

List available voices for your Speech endpoint

To list the available voices for your Speech endpoint, run the following command.

```
curl --location --request GET
'https://INSERT_ENDPOINT_HERE.tts.speech.microsoft.com/cognitiveservices/voices/list' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE'
```

You should receive a response like the following one.

```
[
    {
        "Name": "Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)",
        "DisplayName": "Hoda",
        "LocalName": "هدى",
        "ShortName": "ar-EG-Hoda",
        "Gender": "Female",
        "Locale": "ar-EG",
        "SampleRateHertz": "16000",
        "VoiceType": "Standard"
    },
    {
        "Name": "Microsoft Server Speech Text to Speech Voice (ar-SA, Naayf)",
        "DisplayName": "Naayf",
        "LocalName": "نايف",
        "ShortName": "ar-SA-Naayf",
        "Gender": "Male",
        "Locale": "ar-SA",
        "SampleRateHertz": "16000",
        "VoiceType": "Standard"
    },
    {
        "Name": "Microsoft Server Speech Text to Speech Voice (bg-BG, Ivan)",
        "DisplayName": "Ivan",
        "LocalName": "Иван",
        "ShortName": "bg-BG-Ivan",
        "Gender": "Male",
        "Locale": "bg-BG",
        "SampleRateHertz": "16000",
        "VoiceType": "Standard"
    },
    {
        // This response is truncated. The response will include
        // a complete list of supported languages and specific
        // details like short name, gender, etc.
    }
]
```

In this quickstart, you learn common design patterns for doing text-to-speech synthesis using the Speech SDK. You start by doing basic configuration and synthesis, and move on to more advanced examples for custom application development including:

- Getting responses as in-memory streams
- Customizing output sample rate and bit rate
- Submitting synthesis requests using SSML (speech synthesis markup language)
- Using neural voices

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Download and install

- Windows Install
- Linux Install
- Docker Install (Windows, Linux, macOS)

Follow these steps to install the Speech CLI on Windows:

1. On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
2. Install .NET Core 3.1 SDK.
3. Install the Speech CLI using NuGet by entering this command:

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

Type `spx` to see help for the Speech CLI.

> **NOTE**
>
> As an alternative to NuGet, you can download and extract the Speech CLI for Windows as a zip file.

Font limitations

On Windows, the Speech CLI can only show fonts available to the command prompt on the local computer. Windows Terminal supports all fonts produced interactively by the Speech CLI.

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

Synthesize speech to a speaker

Now you're ready to run the Speech CLI to synthesize speech from text. From the command line, change to the directory that contains the Speech CLI binary file. Then run the following command.

```
spx synthesize --text "The speech synthesizer greets you!"
```

The Speech CLI will produce natural language in English through the computer speaker.

Synthesize speech to a file

Run the following command to change the output from your speaker to a `.wav` file.

```
spx synthesize --text "The speech synthesizer greets you!" --audio output greetings.wav
```

The Speech CLI will produce natural language in English into the `greetings.wav` audio file. In Windows, you can play the audio file by entering `start greetings.wav`.

Get position information

Your project may need to know when a word is spoken by text-to-speech so that it can take specific action based on that timing. As an example, if you wanted to highlight words as they were spoken, you would need to know what to highlight, when to highlight it, and for how long to highlight it.

You can accomplish this using the `WordBoundary` event available within `SpeechSynthesizer`. This event is raised at the beginning of each new spoken word and will provide a time offset within the spoken stream and a text offset within the input prompt.

- `AudioOffset` reports the output audio's elapsed time between the beginning of synthesis and the start of the next word. This is measured in hundred-nanosecond units (HNS) with 10,000 HNS equivalent to 1 millisecond.
- `WordOffset` reports the character position in the input string (original text or SSML) immediately before the

word that's about to be spoken.

> **NOTE**
>
> `WordBoundary` events are raised as the output audio data becomes available, which will be faster than playback to an output device. Appropriately synchronizing stream timing to "real time" must be accomplished by the caller.

You can find examples of using `WordBoundary` in the text-to-speech samples on GitHub.

Next steps

- Get started with Custom Voice
- Improve synthesis with SSML
- Learn how to use the Long Audio API for large text samples like books and news articles
- See the quickstart samples on GitHub

Improve synthesis with Speech Synthesis Markup Language (SSML)

11/23/2021 • 35 minutes to read • Edit Online

Speech Synthesis Markup Language (SSML) is an XML-based markup language that lets developers specify how input text is converted into synthesized speech using the text-to-speech service. Compared to plain text, SSML allows developers to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output. Normal punctuation, such as pausing after a period, or using the correct intonation when a sentence ends with a question mark are automatically handled.

The Speech service implementation of SSML is based on World Wide Web Consortium's Speech Synthesis Markup Language Version 1.0.

> **IMPORTANT**
>
> Chinese, Japanese, and Korean characters count as two characters for billing. For more information, see Pricing.

Neural and custom voices

Use a human-like neural voice, or create your own custom voice unique to your product or brand. For a complete list of supported languages, locales, and voices, see language support. To learn more about neural and custom voices, see Text-to-speech overview.

> **NOTE**
>
> You can hear voices in different styles and pitches reading example text using the Text to Speech page.

Special characters

While using SSML, keep in mind that special characters, such as quotation marks, apostrophes, and brackets must be escaped. For more information, see Extensible Markup Language (XML) 1.0: Appendix D.

Supported SSML elements

Each SSML document is created with SSML elements (or tags). These elements are used to adjust pitch, prosody, volume, and more. The following sections detail how each element is used, and when an element is required or optional.

> **IMPORTANT**
>
> Don't forget to use double quotes around attribute values. Standards for well-formed, valid XML requires attribute values to be enclosed in double quotation marks. For example, `<prosody volume="90">` is a well-formed, valid element, but `<prosody volume=90>` is not. SSML might not recognize attribute values that are not in quotes.

Create an SSML document

`speak` is the root element, and is **required** for all SSML documents. The `speak` element contains important

information, such as version, language, and the markup vocabulary definition.

Syntax

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="string"></speak>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`version`	Indicates the version of the SSML specification used to interpret the document markup. The current version is 1.0.	Required
`xml:lang`	Specifies the language of the root document. The value can contain a lowercase, two-letter language code (for example, `en`), or the language code and uppercase country/region (for example, `en-US`).	Required
`xmlns`	Specifies the URI to the document that defines the markup vocabulary (the element types and attribute names) of the SSML document. The current URI is http://www.w3.org/2001/10/synthesis.	Required

Choose a voice for text-to-speech

The `voice` element is required. It is used to specify the voice that is used for text-to-speech.

Syntax

```
<voice name="string">
    This text will get converted into synthesized speech.
</voice>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`name`	Identifies the voice used for text-to-speech output. For a complete list of supported voices, see Language support.	Required

Example

> **NOTE**
>
> This example uses the `en-US-ChristopherNeural` voice. For a complete list of supported voices, see Language support.

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        This is the text that is spoken.
    </voice>
</speak>
```

Use multiple voices

Within the `speak` element, you can specify multiple voices for text-to-speech output. These voices can be in different languages. For each voice, the text must be wrapped in a `voice` element.

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`name`	Identifies the voice used for text-to-speech output. For a complete list of supported voices, see Language support.	Required

Example

```
<speak version="1.0" xmlns="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>
```

Adjust speaking styles

By default, the text-to-speech service synthesizes text using a neutral speaking style for neural voices. You can adjust the speaking style to express different emotions like cheerfulness, empathy, and calm, or optimize the voice for different scenarios like customer service, newscasting and voice assistant, using the `mstts:express-as` element. This is an optional element unique to the Speech service.

Currently, speaking style adjustments are supported for the following neural voices:

- `en-US-AriaNeural`
- `en-US-JennyNeural`
- `en-US-GuyNeural`
- `en-US-SaraNeural`
- `ja-JP-NanamiNeural`
- `pt-BR-FranciscaNeural`
- `zh-CN-XiaoxiaoNeural`
- `zh-CN-YunyangNeural`
- `zh-CN-YunyeNeural`
- `zh-CN-YunxiNeural`
- `zh-CN-XiaohanNeural`
- `zh-CN-XiaomoNeural`
- `zh-CN-XiaoxuanNeural`

- `zh-CN-XiaoruiNeural`
- `zh-CN-XiaoshuangNeural`

The intensity of speaking style can be further changed to better fit your use case. You can specify a stronger or softer style with `styledegree` to make the speech more expressive or subdued. Currently, speaking style adjustments are supported for Chinese (Mandarin, Simplified) neural voices.

Apart from adjusting the speaking styles and style degree, you can also adjust the `role` parameter so that the voice will imitate 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 will not be changed. Currently, role adjustments are supported for these Chinese (Mandarin, Simplified) neural voices:

- `zh-CN-XiaomoNeural`
- `zh-CN-XiaoxuanNeural`

Above changes are applied at the sentence level, and styles and role-plays vary by voice. If a style or role-play isn't supported, the service will return speech in the default neutral speaking way. You can see what styles and roles are supported for each voice through the voice list API or through the code-free Audio Content Creation platform.

Syntax

```
<mstts:express-as style="string"></mstts:express-as>
```

```
<mstts:express-as style="string" styledegree="value"></mstts:express-as>
```

```
<mstts:express-as role="string" style="string"></mstts:express-as>
```

> **NOTE**
> At the moment, `styledegree` only supports Chinese (Mandarin, Simplified) neural voices. `role` only supports zh-CN-XiaomoNeural and zh-CN-XiaoxuanNeural.

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`style`	Specifies the speaking style. Currently, speaking styles are voice-specific.	Required if adjusting the speaking style for a neural voice. If using `mstts:express-as`, then style must be provided. If an invalid value is provided, this element will be ignored.
`styledegree`	Specifies the intensity of speaking style. **Accepted values**: 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 slightly tendency for the target style. A value of 2 results in a doubling of the default style intensity.	Optional (At the moment, `styledegree` only supports Chinese (Mandarin, Simplified) neural voices.)

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`role`	Specifies the speaking role-play. The voice will act as a different age and gender, but the voice name will not be changed.	Optional (At the moment, `role` only supports zh-CN-XiaomoNeural and zh-CN-XiaoxuanNeural.)

Use this table to determine which speaking styles are supported for each neural voice.

VOICE	STYLE	DESCRIPTION
`en-US-AriaNeural`	`style="newscast-formal"`	Expresses a formal, confident and authoritative tone for news delivery
	`style="newscast-casual"`	Expresses a versatile and casual tone for general news delivery
	`style="narration-professional"`	Express a professional, objective tone for content reading
	`style="customerservice"`	Expresses a friendly and helpful tone for customer support
	`style="chat"`	Expresses a casual and relaxed tone
	`style="cheerful"`	Expresses a positive and happy tone
	`style="empathetic"`	Expresses a sense of caring and understanding
`en-US-JennyNeural`	`style="customerservice"`	Expresses a friendly and helpful tone for customer support
	`style="chat"`	Expresses a casual and relaxed tone
	`style="assistant"`	Expresses a warm and relaxed tone for digital assistants
	`style="newscast"`	Expresses a versatile and casual tone for general news delivery
`en-US-GuyNeural`	`style="newscast"`	Expresses a formal and professional tone for narrating news
`en-US-SaraNeural`	`style="cheerful"`	Expresses a positive and happy tone
	`style="sad"`	Expresses a sorrowful tone
	`style="angry"`	Expresses an angry and annoyed tone
`ja-JP-NanamiNeural`	`style="cheerful"`	Expresses a positive and happy tone
	`style="chat"`	Expresses a casual and relaxed tone

VOICE	STYLE	DESCRIPTION
	`style="customerservice"`	Expresses a friendly and helpful tone for customer support
`pt-BR-FranciscaNeural`	`style="calm"`	Expresses a cool, collected, and composed attitude when speaking. Tone, pitch, prosody is much more uniform compared to other types of speech.
`zh-CN-XiaoxiaoNeural`	`style="newscast"`	Expresses a formal and professional tone for narrating news
	`style="customerservice"`	Expresses a friendly and helpful tone for customer support
	`style="assistant"`	Expresses a warm and relaxed tone for digital assistants
	`style="chat"`	Expresses a casual and relaxed tone for chit-chat
	`style="calm"`	Expresses a cool, collected, and composed attitude when speaking. Tone, pitch, prosody is much more uniform compared to other types of speech.
	`style="cheerful"`	Expresses an upbeat and enthusiastic tone, with higher pitch and vocal energy
	`style="sad"`	Expresses a sorrowful tone, with higher pitch, less intensity, and lower vocal energy. Common indicators of this emotion would be whimpers or crying during speech.
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
	`style="disgruntled"`	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
	`style="serious"`	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.

VOICE	STYLE	DESCRIPTION
	`style="affectionate"`	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.
	`style="gentle"`	Expresses a mild, polite, and pleasant tone, with lower pitch and vocal energy
	`style="lyrical"`	Expresses emotions in a melodic and sentimental way
`zh-CN-YunyangNeural`	`style="customerservice"`	Expresses a friendly and helpful tone for customer support
`zh-CN-YunyeNeural`	`style="calm"`	Expresses a cool, collected, and composed attitude when speaking. Tone, pitch, prosody is much more uniform compared to other types of speech.
	`style="cheerful"`	Expresses an upbeat and enthusiastic tone, with higher pitch and vocal energy
	`style="sad"`	Expresses a sorrowful tone, with higher pitch, less intensity, and lower vocal energy. Common indicators of this emotion would be whimpers or crying during speech.
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
	`style="disgruntled"`	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
	`style="serious"`	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.
`zh-CN-YunxiNeural`	`style="assistant"`	Expresses a warm and relaxed tone for digital assistants

VOICE	STYLE	DESCRIPTION
	`style="cheerful"`	Expresses an upbeat and enthusiastic tone, with higher pitch and vocal energy
	`style="sad"`	Expresses a sorrowful tone, with higher pitch, less intensity, and lower vocal energy. Common indicators of this emotion would be whimpers or crying during speech.
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
	`style="disgruntled"`	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
	`style="serious"`	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.
	`style="depressed"`	Expresses a melancholic and despondent tone with lower pitch and energy
	`style="embarrassed"`	Expresses an uncertain and hesitant tone when the speaker is feeling uncomfortable
`zh-CN-XiaohanNeural`	`style="cheerful"`	Expresses an upbeat and enthusiastic tone, with higher pitch and vocal energy
	`style="sad"`	Expresses a sorrowful tone, with higher pitch, less intensity, and lower vocal energy. Common indicators of this emotion would be whimpers or crying during speech.
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.

VOICE	STYLE	DESCRIPTION
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
	`style="disgruntled"`	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
	`style="serious"`	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.
	`style="embarrassed"`	Expresses an uncertain and hesitant tone when the speaker is feeling uncomfortable
	`style="affectionate"`	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.
	`style="gentle"`	Expresses a mild, polite, and pleasant tone, with lower pitch and vocal energy
`zh-CN-XiaomoNeural`	`style="calm"`	Expresses a cool, collected, and composed attitude when speaking. Tone, pitch, prosody is much more uniform compared to other types of speech.
	`style="cheerful"`	Expresses an upbeat and enthusiastic tone, with higher pitch and vocal energy
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
	`style="disgruntled"`	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
	`style="serious"`	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.

VOICE	STYLE	DESCRIPTION
	`style="depressed"`	Expresses a melancholic and despondent tone with lower pitch and energy
	`style="gentle"`	Expresses a mild, polite, and pleasant tone, with lower pitch and vocal energy
`zh-CN-XiaoxuanNeural`	`style="calm"`	Expresses a cool, collected, and composed attitude when speaking. Tone, pitch, prosody is much more uniform compared to other types of speech.
	`style="cheerful"`	Expresses an upbeat and enthusiastic tone, with higher pitch and vocal energy
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
	`style="disgruntled"`	Expresses a disdainful and complaining tone. Speech of this emotion displays displeasure and contempt.
	`style="serious"`	Expresses a strict and commanding tone. Speaker often sounds stiffer and much less relaxed with firm cadence.
	`style="depressed"`	Expresses a melancholic and despondent tone with lower pitch and energy
	`style="gentle"`	Expresses a mild, polite, and pleasant tone, with lower pitch and vocal energy
`zh-CN-XiaoruiNeural`	`style="sad"`	Expresses a sorrowful tone, with higher pitch, less intensity, and lower vocal energy. Common indicators of this emotion would be whimpers or crying during speech.
	`style="angry"`	Expresses an angry and annoyed tone, with lower pitch, higher intensity, and higher vocal energy. The speaker is in a state of being irate, displeased, and offended.

VOICE	STYLE	DESCRIPTION
	`style="fearful"`	Expresses a scared and nervous tone, with higher pitch, higher vocal energy, and faster rate. The speaker is in a state of tenseness and uneasiness.
`zh-CN-XiaoshuangNeural`	`style="chat"`	Expresses a casual and relaxed tone.

Use this table to check the supported roles and their definitions.

ROLE	DESCRIPTION
`role="Girl"`	The voice imitates to a girl.
`role="Boy"`	The voice imitates to a boy.
`role="YoungAdultFemale"`	The voice imitates to a young adult female.
`role="YoungAdultMale"`	The voice imitates to a young adult male.
`role="OlderAdultFemale"`	The voice imitates to an older adult female.
`role="OlderAdultMale"`	The voice imitates to an older adult male.
`role="SeniorFemale"`	The voice imitates to a senior female.
`role="SeniorMale"`	The voice imitates to a senior male.

Example

This SSML snippet illustrates how the `<mstts:express-as>` element is used to change the speaking style to `cheerful`.

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
    <voice name="en-US-AriaNeural">
        <mstts:express-as style="cheerful">
            That'd be just amazing!
        </mstts:express-as>
    </voice>
</speak>
```

This SSML snippet illustrates how the `styledegree` attribute is used to change the intensity of speaking style for XiaoxiaoNeural.

```
<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-XiaoxiaoNeural">
        <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>
```

This SSML snippet illustrates how the `role` attribute is used to change the role-play for XiaomoNeural.

```
<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>
```

Adjust speaking languages

You can adjust speaking languages for neural voices. Enable one voice to speak different languages fluently (like English, Spanish, and Chinese) using the `<lang xml:lang>` element. This is an optional element unique to the Speech service. Without this element, the voice will speak its primary language. Currently, speaking language adjustments are supported for these neural voices: `en-US-JennyMultilingualNeural`. Above changes are applied at the sentence level and word level. If a language isn't supported, the service will return no audio stream.

> **NOTE**
> Currently, the `<lang xml:lang>` element is incompatible with `prosody` and `break` element, you cannot adjust pause and prosody like pitch, contour, rate, duration, volume in this element.

Syntax

```
<lang xml:lang="string"></lang>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`lang`	Specifies the speaking languages. Currently, speaking different languages are voice-specific.	Required if adjusting the speaking language for a neural voice. If using `lang xml:lang`, then locale must be provided.

Use this table to determine which speaking languages are supported for each neural voice.

VOICE	LOCALE LANGUAGE	DESCRIPTION
`en-US-JennyMultilingualNeural`	`lang="en-US"`	Speak en-US locale, which is the primary locale of this voice
	`lang="en-CA"`	Speak en-CA locale language
	`lang="en-AU"`	Speak en-AU locale language
	`lang="en-GB"`	Speak en-GB locale language

VOICE	LOCALE LANGUAGE	DESCRIPTION
	`lang="de-DE"`	Speak de-DE locale language
	`lang="fr-FR"`	Speak fr-FR locale language
	`lang="fr-CA"`	Speak fr-CA locale language
	`lang="es-ES"`	Speak es-ES locale language
	`lang="es-MX"`	Speak es-MX locale language
	`lang="zh-CN"`	Speak zh-CN locale language
	`lang="ko-KR"`	Speak ko-KR locale language
	`lang="ja-JP"`	Speak ja-JP locale language
	`lang="it-IT"`	Speak it-IT locale language
	`lang="pt-BR"`	Speak pt-BR locale language

Example

This SSML snippet shows how to use `<lang xml:lang>` to change the speaking languages to `en-US`, `es-MX` and `de-DE`.

```
<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">
        I am looking forward to the exciting things.
        <lang xml:lang="es-MX">
            Estoy deseando que lleguen las cosas emocionantes.
        </lang>
        <lang xml:lang="de-DE">
            Ich freue mich auf die spannenden Dinge.
        </lang>
    </voice>
</speak>
```

Add or remove a break/pause

Use the `break` element to insert pauses (or breaks) between words, or prevent pauses automatically added by the text-to-speech service.

> **NOTE**
> Use this element to override the default behavior of text-to-speech (TTS) for a word or phrase if the synthesized speech for that word or phrase sounds unnatural. Set `strength` to `none` to prevent a prosodic break, which is automatically inserted by the text-to-speech service.

Syntax

```
<break strength="string" />
<break time="string" />
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`strength`	Specifies the relative duration of a pause using one of the following values: • none • x-weak • weak • medium (default) • strong • x-strong	Optional
`time`	Specifies the absolute duration of a pause in seconds or milliseconds,this value should be set less than 5000ms. Examples of valid values are `2s` and `500ms`	Optional

STRENGTH	DESCRIPTION
None, or if no value provided	0 ms
x-weak	250 ms
weak	500 ms
medium	750 ms
strong	1000 ms
x-strong	1250 ms

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        Welcome to Microsoft Cognitive Services <break time="100ms" /> Text-to-Speech API.
    </voice>
</speak>
```

Add silence

Use the `mstts:silence` element to insert pauses before or after text, or between the 2 adjacent sentences.

> **NOTE**
> The difference between `mstts:silence` and `break` is that `break` can be added to any place in the text, but silence only works at the beginning or end of input text, or at the boundary of 2 adjacent sentences.

Syntax

```
<mstts:silence  type="string"  value="string"/>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`type`	Specifies the location of silence be added: • `Leading` – at the beginning of text • `Tailing` – in the end of text • `Sentenceboundary` – between adjacent sentences	Required
`Value`	Specifies the absolute duration of a pause in seconds or milliseconds,this value should be set less than 5000ms. Examples of valid values are `2s` and `500ms`	Required

Example In this example, `mtts:silence` is used to add 200 ms of silence between two sentences.

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
<voice name="en-US-ChristopherNeural">
<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>
```

Specify paragraphs and sentences

`p` and `s` elements are used to denote paragraphs and sentences, respectively. In the absence of these elements, the text-to-speech service automatically determines the structure of the SSML document.

The `p` element can contain text and the following elements: `audio`, `break`, `phoneme`, `prosody`, `say-as`, `sub`, `mstts:express-as`, and `s`.

The `s` element can contain text and the following elements: `audio`, `break`, `phoneme`, `prosody`, `say-as`, `mstts:express-as`, and `sub`.

Syntax

```
<p></p>
<s></s>
```

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <p>
            <s>Introducing the sentence element.</s>
            <s>Used to mark individual sentences.</s>
        </p>
        <p>
            Another simple paragraph.
            Sentence structure in this paragraph is not explicitly marked.
        </p>
    </voice>
</speak>
```

Use phonemes to improve pronunciation

The `ph` element is used to for phonetic pronunciation in SSML documents. The `ph` element can only contain text, no other elements. Always provide human-readable speech as a fallback.

Phonetic alphabets are composed of phones, which are made up of letters, numbers, or characters, sometimes in combination. Each phone describes a unique sound of speech. This is in contrast to the Latin alphabet, where any letter might represent multiple spoken sounds. Consider the different pronunciations of the letter "c" in the words "candy" and "cease", or the different pronunciations of the letter combination "th" in the words "thing" and "those".

> **NOTE**
> Phonemes tag is not supported for these 5 voices (et-EE-AnuNeural, ga-IE-OrlaNeural, lt-LT-OnaNeural, lv-LV-EveritaNeural and mt-MT-GarceNeural) at the moment.

Syntax

```
<phoneme alphabet="string" ph="string"></phoneme>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`alphabet`	Specifies the phonetic alphabet to use when synthesizing the pronunciation of the string in the `ph` attribute. The string specifying the alphabet must be specified in lowercase letters. The following are the possible alphabets that you can specify. • `ipa` – International Phonetic Alphabet • `sapi` – Speech service phonetic alphabet • `ups` – Universal Phone Set The alphabet applies only to the `phoneme` in the element..	Optional

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`ph`	A string containing phones that specify the pronunciation of the word in the `phoneme` element. If the specified string contains unrecognized phones, the text-to-speech (TTS) service rejects the entire SSML document and produces none of the speech output specified in the document.	Required if using phonemes.

Examples

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <phoneme alphabet="ipa" ph="təˈmeɪtoʊ"> tomato </phoneme>
    </voice>
</speak>
```

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <phoneme alphabet="sapi" ph="iy eh n y uw eh s"> en-US </phoneme>
    </voice>
</speak>
```

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <s>His name is Mike <phoneme alphabet="ups" ph="JH AU"> Zhou </phoneme></s>
    </voice>
</speak>
```

Use custom lexicon to improve pronunciation

Sometimes the text-to-speech service cannot accurately pronounce a word. For example, the name of a company, a medical term or an emoji. Developers can define how single entities are read in SSML using the `phoneme` and `sub` tags. However, if you need to define how multiple entities are read, you can create a custom lexicon using the `lexicon` tag.

> **NOTE**
> Custom lexicon currently supports UTF-8 encoding.

> **NOTE**
> Custom lexicon is not supported for these 5 voices (et-EE-AnuNeural, ga-IE-OrlaNeural, lt-LT-OnaNeural, lv-LV-EveritaNeural and mt-MT-GarceNeural) at the moment.

Syntax

```
<lexicon uri="string"/>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`uri`	The address of the external PLS document.	Required.

Usage

To define how multiple entities are read, you can create a custom lexicon, which is stored as an .xml or .pls file. The following is a sample .xml file.

```
<?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> bɛˈniːnji</phoneme>
  </lexeme>
  <lexeme>
    <grapheme>⍰</grapheme>
    <alias>test emoji</alias>
  </lexeme>
</lexicon>
```

The `lexicon` element contains at least one `lexeme` element. Each `lexeme` element contains at least one `grapheme` element and one or more `grapheme`, `alias`, and `phoneme` elements. The `grapheme` element contains text describing the orthography . The `alias` elements are used to indicate the pronunciation of an acronym or an abbreviated term. The `phoneme` element provides text describing how the `lexeme` is pronounced. When `alias` and `phoneme` element are provided with the same `grapheme` element, `alias` has higher priority.

> **IMPORTANT**
> The `lexeme` element is case sensitive in custom lexicon. For example, if you only provide a phoneme for `lexeme` 'Hello', it will not work for `lexeme` 'hello'.

Lexicon contains necessary `xml:lang` attribute to indicate which locale it should be applied for. One custom lexicon is limited to one locale by design, so apply it for a different locale it won't work.

It's important to note, that you cannot directly set the pronunciation of a phrase using the custom lexicon. If you need to set the pronunciation for an acronym or an abbreviated term, first provide an `alias`, then associate the `phoneme` with that `alias`. For example:

```
  <lexeme>
    <grapheme>Scotland MV</grapheme>
    <alias>ScotlandMV</alias>
  </lexeme>
  <lexeme>
    <grapheme>ScotlandMV</grapheme>
    <phoneme>ˈskɒtlənd.ˈmiːdiəm.weɪv</phoneme>
  </lexeme>
```

> **NOTE**
> The syllable boundary is '.' in the International Phonetic Alphabet.

You could also directly provide your expected `alias` for the acronym or abbreviated term. For example:

```
<lexeme>
  <grapheme>Scotland MV</grapheme>
  <alias>Scotland Media Wave</alias>
</lexeme>
```

> **IMPORTANT**
> The `phoneme` element cannot contain white spaces when using IPA.

For more information about custom lexicon file, see Pronunciation Lexicon Specification (PLS) Version 1.0.

Next, publish your custom lexicon file. While we don't have restrictions on where this file can be stored, we do recommend using Azure Blob Storage.

After you've published your custom lexicon, you can reference it from your SSML.

> **NOTE**
> The `lexicon` element must be inside the `voice` element.

```
<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-ChristopherNeural">
        <lexicon uri="http://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>
```

When using this custom lexicon, "BTW" will be read as "By the way". "Benigni" will be read with the provided IPA "bɛˈniːnji".

Since it's easy to make mistakes in custom lexicon, Microsoft has provided validation tool for custom lexicon. It provides detailed error messages that help you find errors. Before you send SSML with custom lexicon to the Speech service, you should check your custom lexicon with this tool.

Limitations

- File size: custom lexicon file size maximum limit is 100KB, if beyond this size, synthesis request will fail.
- Lexicon cache refresh: custom lexicon will be cached with URI as key on TTS Service when it's first loaded. Lexicon with same URI won't be reloaded within 15 mins, so custom lexicon change needs to wait at most 15 mins to take effect.

Speech service phonetic sets

In the sample above, we're using the International Phonetic Alphabet, also known as the IPA phone set. We suggest developers use the IPA, because it is the international standard. For some IPA characters, they have the 'precomposed' and 'decomposed' version when being represented with Unicode. Custom lexicon only support

the decomposed unicodes.

Considering that the IPA is not easy to remember, the Speech service defines a phonetic set for seven languages (`en-US`, `fr-FR`, `de-DE`, `es-ES`, `ja-JP`, `zh-CN`, and `zh-TW`).

You can use the `x-microsoft-sapi` as the value for the `alphabet` attribute with custom lexicons as demonstrated below:

```
<?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>
```

For more information on the detailed Speech service phonetic alphabet, see the Speech service phonetic sets.

Adjust prosody

The `prosody` element is used to specify changes to pitch, contour, range, rate, duration, and volume for the text-to-speech output. The `prosody` element 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. The text-to-speech service limits or substitutes values that are not supported. Examples of unsupported values are a pitch of 1 MHz or a volume of 120.

Syntax

```
<prosody pitch="value" contour="value" range="value" rate="value" duration="value" volume="value"></prosody>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL

`pitch`	Indicates the baseline pitch for the text. You can express the pitch as:
	• An absolute value, expressed as a number followed by "Hz" (Hertz). For example, `<prosody pitch="600Hz">some text</prosody>`.
	• A relative value, expressed as a number preceded by "+" or "-" and followed by "Hz" or "st", that specifies an amount to change the pitch. For example: `<prosody pitch="+80Hz">some text</prosody>` or `<prosody pitch="-2st">some text</prosody>`. The "st" indicates the change unit is semitone, which is half of a tone (a half step) on the standard diatonic scale.
	• A constant value:
	  ○ x-low
	  ○ low
	  ○ medium
	  ○ high
	  ○ x-high
	  ○ default
		Optional

`contour`	Contour now supports both neural and standard voices. 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: `<prosody contour="(0%,+20Hz) (10%,-2st) (40%,+10Hz)">` The first value in each set of parameters specifies the location of the pitch change as a percentage of the duration of the text. The second value specifies the amount to raise or lower the pitch, using a relative value or an enumeration value for pitch (see `pitch`).	Optional
`range`	A value that represents the range of pitch for the text. You can express `range` using the same absolute values, relative values, or enumeration values used to describe `pitch`.	Optional

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL

`rate`	Indicates the speaking rate of the text. You can express `rate` as:
	• A relative value, expressed as a number that acts as a multiplier of the default. For example, a value of *1* results in no change in the rate. A value of *0.5* results in a halving of the rate. A value of *3* results in a tripling of the rate.
	• A constant value:
	  ◦ x-slow
	  ◦ slow
	  ◦ medium
	  ◦ fast
	  ◦ x-fast
	  ◦ default
		Optional

`duration`	The period of time that should elapse while the speech synthesis (TTS) service reads the text, in seconds or milliseconds. For example, *2s* or *1800ms*. Duration supports standard voices only.	Optional

`volume`	Indicates the volume level of the speaking voice. You can express the volume as:
	• An absolute value, expressed as a number in the range of 0.0 to 100.0, from *quietest* to *loudest*. For example, 75. The default is 100.0.
	• A relative value, expressed as a number preceded by "+" or "-" that specifies an amount to change the volume. For example, +10 or -5.5.
	• A constant value:
	  ◦ silent
	  ◦ x-soft
	  ◦ soft
	  ◦ medium
	  ◦ loud
	  ◦ x-loud
	  ◦ default
		Optional

Change speaking rate

Speaking rate can be applied to Neural voices and standard voices at the word or sentence-level.

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <prosody rate="+30.00%">
            Welcome to Microsoft Cognitive Services Text-to-Speech API.
        </prosody>
    </voice>
</speak>
```

Change volume

Volume changes can be applied to standard voices at the word or sentence-level. Whereas volume changes can only be applied to neural voices at the sentence level.

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <prosody volume="+20.00%">
            Welcome to Microsoft Cognitive Services Text-to-Speech API.
        </prosody>
    </voice>
</speak>
```

Change pitch

Pitch changes can be applied to standard voices at the word or sentence-level. Whereas pitch changes can only be applied to neural voices at the sentence level.

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        Welcome to <prosody pitch="high">Microsoft Cognitive Services Text-to-Speech API.</prosody>
    </voice>
</speak>
```

Change pitch contour

> **IMPORTANT**
>
> Pitch contour changes are now supported with neural voices.

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <prosody contour="(60%,-60%) (100%,+80%)" >
            Were you the only person in the room?
        </prosody>
    </voice>
</speak>
```

say-as element

`say-as` is an optional element that indicates the content type (such as number or date) of the element's text. This provides guidance to the speech synthesis engine about how to pronounce the text.

Syntax

```
<say-as interpret-as="string" format="digit string" detail="string"> </say-as>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`interpret-as`	Indicates the content type of element's text. For a list of types, see the table below.	Required
`format`	Provides additional information about the precise formatting of the element's text for content types that might have ambiguous formats. SSML defines formats for content types that use them (see table below).	Optional
`detail`	Indicates the level of detail to be spoken. For example, this attribute might request that the speech synthesis engine pronounce punctuation marks. There are no standard values defined for `detail`.	Optional

The following are the supported content types for the `interpret-as` and `format` attributes. Include the `format` attribute only if `interpret-as` is set to date and time.

INTERPRET-AS	FORMAT	INTERPRETATION
`address`		The text is spoken as an address. The speech synthesis engine pronounces: `I'm at <say-as interpret-as="address">150th CT NE, Redmond, WA</say-as>` As "I'm at 150th court north east redmond washington."
`cardinal`, `number`		The text is spoken as a cardinal number. The speech synthesis engine pronounces: `There are <say-as interpret-as="cardinal">3</say-as> alternatives` As "There are three alternatives."
`characters`, `spell-out`		The text is spoken as individual letters (spelled out). The speech synthesis engine pronounces: `<say-as interpret-as="characters">test</say-as>` As "T E S T."

INTERPRET-AS	FORMAT	INTERPRETATION
`date`	dmy, mdy, ymd, ydm, ym, my, md, dm, d, m, y	The text is spoken as a date. The `format` attribute specifies the date's format (*d=day, m=month, and y=year*). The speech synthesis engine pronounces: `Today is <say-as interpret-as="date" format="mdy">10-19-2016</say-as>` As "Today is October nineteenth two thousand sixteen."
`digits`, `number_digit`		The text is spoken as a sequence of individual digits. The speech synthesis engine pronounces: `<say-as interpret-as="number_digit">123456789</say-as>` As "1 2 3 4 5 6 7 8 9."
`fraction`		The text is spoken as a fractional number. The speech synthesis engine pronounces: `<say-as interpret-as="fraction">3/8</say-as> of an inch` As "three eighths of an inch."
`ordinal`		The text is spoken as an ordinal number. The speech synthesis engine pronounces: `Select the <say-as interpret-as="ordinal">3rd</say-as> option` As "Select the third option".

INTERPRET-AS	FORMAT	INTERPRETATION
`telephone`		The text is spoken as a telephone number. The `format` attribute can contain digits that represent a country code. For example, "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 `format`. The speech synthesis engine pronounces: `The number is <say-as interpret-as="telephone" format="1">(888) 555-1212</say-as>` As "My number is area code eight eight eight five five five one two one two."
`time`	hms12, hms24	The text is spoken as a time. The `format` attribute specifies whether the time is specified using a 12-hour clock (hms12) or a 24-hour clock (hms24). Use a colon to separate numbers representing hours, minutes, and seconds. The following are valid time examples: 12:35, 1:14:32, 08:15, and 02:50:45. The speech synthesis engine pronounces: `The train departs at <say-as interpret-as="time" format="hms12">4:00am</say-as>` As "The train departs at four A M."

`name`		The text is spoken as a person name. The speech synthesis engine pronounces:
		
		`<say-as interpret-as="name">ED</say-as>`
		
		as [æd].
		In Chinese names, some characters pronounce differently when they appear in a family name. For example, the speech synthesis engine says 仇 in
		
		`<say-as interpret-as="address">仇先生</say-as>`
		
		as [qiú] instead of [chóu].

Usage

The `say-as` element can only contain text.

Example

The speech synthesis engine speaks the following example as "Your first request was for one room on October nineteenth twenty ten with early arrival at twelve thirty five PM."

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <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>
```

Add recorded audio

`audio` is an optional element that allows you to insert MP3 audio into an SSML document. The body of the audio element can contain plain text or SSML markup that's spoken if the audio file is unavailable or unplayable. Additionally, the `audio` element can contain text and the following elements: `audio`, `break`, `p`, `s`, `phoneme`, `prosody`, `say-as`, and `sub`.

Any audio included in the SSML document must meet these requirements:

- The MP3 must be hosted on an Internet-accessible HTTPS endpoint. HTTPS is required, and the domain hosting the MP3 file must present a valid, trusted TLS/SSL certificate.
- The MP3 must be a valid MP3 file (MPEG v2).
- The bit rate must be 48 kbps.
- The sample rate must be 16,000 Hz.
- The combined total time for all text and audio files in a single response cannot exceed ninety (90) seconds.
- The MP3 must not contain any customer-specific or other sensitive information.

Syntax

```
<audio src="string"/></audio>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`src`	Specifies the location/URL of the audio file.	Required if using the audio element in your SSML document.

Example

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">
    <voice name="en-US-ChristopherNeural">
        <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>
```

Add background audio

The `mstts:backgroundaudio` element allows you to add background audio to your SSML documents (or mix an audio file with text-to-speech). With `mstts:backgroundaudio` you can loop an audio file in the background, fade in at the beginning of text-to-speech, and fade out at the end of text-to-speech.

If the background audio provided is shorter than the text-to-speech or the fade out, it will loop. If it is longer than the text-to-speech, it will stop when the fade out has finished.

Only one background audio file is allowed per SSML document. However, you can intersperse `audio` tags within the `voice` element to add additional audio to your SSML document.

Syntax

```
<mstts:backgroundaudio src="string" volume="string" fadein="string" fadeout="string"/>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`src`	Specifies the location/URL of the background audio file.	Required if using background audio in your SSML document.
`volume`	Specifies the volume of the background audio file. **Accepted values**: `0` to `100` inclusive. The default value is `1`.	Optional
`fadein`	Specifies the duration of the background audio "fade in" as milliseconds. The default value is `0`, which is the equivalent to no fade in. **Accepted values**: `0` to `10000` inclusive.	Optional
`fadeout`	Specifies the duration of the background audio fade out in milliseconds. The default value is `0`, which is the equivalent to no fade out. **Accepted values**: `0` to `10000` inclusive.	Optional

Example

```
<speak version="1.0" xml:lang="en-US" xmlns:mstts="http://www.w3.org/2001/mstts">
    <mstts:backgroundaudio src="https://contoso.com/sample.wav" volume="0.7" fadein="3000" fadeout="4000"/>
    <voice name="Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)">
        The text provided in this document will be spoken over the background audio.
    </voice>
</speak>
```

Bookmark element

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. We will not read out the bookmark elements. The bookmark element can be used to reference a specific location in the text or tag sequence.

> **NOTE**
> `bookmark` element only works for `en-US-AriaNeural` voice for now.

Syntax

```
<bookmark mark="string"/>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
`mark`	Specifies the reference text of the `bookmark` element.	Required.

Example

As an example, you might want to know the time offset of each flower word as following

```
<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>
```

Get bookmark using Speech SDK

You can subscribe to the `BookmarkReached` event in Speech SDK to get the bookmark offsets.

> **NOTE**
> `BookmarkReached` event is only available since Speech SDK version 1.16.

`BookmarkReached` events are raised as the output audio data becomes available, which will be faster than playback to an output device.

- `AudioOffset` reports the output audio's elapsed time between the beginning of synthesis and the bookmark element. This is measured in hundred-nanosecond units (HNS) with 10,000 HNS equivalent to 1 millisecond.
- `Text` is the reference text of the bookmark element, which is the string you set in the `mark` attribute.

- C#
- C++
- Java
- Python
- JavaScript
- Objective-C
- Swift

For more information, see `BookmarkReached` .

```
synthesizer.BookmarkReached += (s, e) =>
{
    // The unit of e.AudioOffset is tick (1 tick = 100 nanoseconds), divide by 10,000 to convert to
milliseconds.
    Console.WriteLine($"Bookmark reached. Audio offset: " +
        $"{e.AudioOffset / 10000}ms, bookmark text: {e.Text}.");
};
```

For the example SSML above, the `BookmarkReached` event will be triggered twice, and the console output will be

```
Bookmark reached. Audio offset: 825ms, bookmark text: flower_1.
Bookmark reached. Audio offset: 1462.5ms, bookmark text: flower_2.
```

Next steps

- Language support: voices, locales, languages

Lower speech synthesis latency using Speech SDK

11/23/2021 • 7 minutes to read • Edit Online

> **NOTE**
>
> This article requires Speech SDK 1.17.0 or later.

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.

```
var result = await synthesizer.SpeakTextAsync(text);
Console.WriteLine($"first byte latency:
\t{result.Properties.GetProperty(PropertyId.SpeechServiceResponse_SynthesisFirstByteLatencyMs)} ms");
Console.WriteLine($"finish latency:
\t{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;
```

LATENCY	DESCRIPTION	SPEECHSYNTHESISRESULT PROPERTY KEY
`first byte latency`	Indicates the time delay between the synthesis starts and the first audio chunk is received.	`SpeechServiceResponse_SynthesisFirstByteLatency`
`finish latency`	Indicates the time delay between the synthesis starts and the whole synthesized audio is received.	`SpeechServiceResponse_SynthesisFinishLatencyMs`

The Speech SDK measured the latencies and puts them in the property bag of `SpeechSynthesisResult`. Refer following codes to get them.

```
auto result = synthesizer->SpeakTextAsync(text).get();
auto firstByteLatency = std::stoi(result-
>Properties.GetProperty(PropertyId::SpeechServiceResponse_SynthesisFirstByteLatencyMs));
auto finishedLatency = std::stoi(result-
>Properties.GetProperty(PropertyId::SpeechServiceResponse_SynthesisFinishLatencyMs));
// you can also get the result id, and send to us when you need help for diagnosis
auto resultId = result->ResultId;
```

LATENCY	DESCRIPTION	SPEECHSYNTHESISRESULT PROPERTY KEY
`first byte latency`	Indicates the time delay between the synthesis starts and the first audio chunk is received.	`SpeechServiceResponse_SynthesisFirstByteLatency`
`finish latency`	Indicates the time delay between the synthesis starts and the whole synthesized audio is received.	`SpeechServiceResponse_SynthesisFinishLatencyMs`

The Speech SDK measured the latencies and puts them in the property bag of `SpeechSynthesisResult`. Refer following codes to get them.

```
SpeechSynthesisResult result = synthesizer.SpeakTextAsync(text).get();
System.out.println("first byte latency: \t" +
result.getProperties().getProperty(PropertyId.SpeechServiceResponse_SynthesisFirstByteLatencyMs) + " ms.");
System.out.println("finish latency: \t" +
result.getProperties().getProperty(PropertyId.SpeechServiceResponse_SynthesisFinishLatencyMs) + " ms.");
// you can also get the result id, and send to us when you need help for diagnosis
String resultId = result.getResultId();
```


LATENCY	DESCRIPTION	SPEECHSYNTHESISRESULT PROPERTY KEY
`first byte latency`	Indicates the time delay between the synthesis starts and the first audio chunk is received.	`SpeechServiceResponse_SynthesisFirstByteLatency`
`finish latency`	Indicates the time delay between the synthesis starts and the whole synthesized audio is received.	`SpeechServiceResponse_SynthesisFinishLatencyMs`

The Speech SDK measured the latencies and puts them in the property bag of `SpeechSynthesisResult`. Refer following codes to get them.


```
result = synthesizer.speak_text_async(text).get()
first_byte_latency =
int(result.properties.get_property(speechsdk.PropertyId.SpeechServiceResponse_SynthesisFirstByteLatencyMs))
finished_latency =
int(result.properties.get_property(speechsdk.PropertyId.SpeechServiceResponse_SynthesisFinishLatencyMs))
# you can also get the result id, and send to us when you need help for diagnosis
result_id = result.result_id
```

LATENCY	DESCRIPTION	SPXSPEECHSYNTHESISRESULT PROPERTY KEY
`first byte latency`	Indicates the time delay between the synthesis starts and the first audio chunk is received.	`SPXSpeechServiceResponseSynthesisFirstByteLaten`
`finish latency`	Indicates the time delay between the synthesis starts and the whole synthesized audio is received.	`SPXSpeechServiceResponseSynthesisFinishLatencyM`

The Speech SDK measured the latencies and puts them in the property bag of `SPXSpeechSynthesisResult`. Refer following codes to get them.

```
SPXSpeechSynthesisResult *speechResult = [speechSynthesizer speakText:text];
int firstByteLatency = [intString [speechResult.properties
getPropertyById:SPXSpeechServiceResponseSynthesisFirstByteLatencyMs]];
int finishedLatency = [intString [speechResult.properties
getPropertyById:SPXSpeechServiceResponseSynthesisFinishLatencyMs]];
// you can also get the result id, and send to us when you need help for diagnosis
NSString *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 `AudioDateStream` of the Speech SDK to enable streaming.

Taking `AudioDateStream` as an example:

```
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.");
            }
        }
    }
}
```

You can use the `PullAudioOutputStream`, `PushAudioOutputStream`, the `Synthesizing` event, and `AudioDateStream` of the Speech SDK to enable streaming.

Taking `AudioDateStream` as an example:

```
auto synthesizer = SpeechSynthesizer::FromConfig(config, nullptr);
auto result = synthesizer->SpeakTextAsync(text).get();
auto audioDataStream = AudioDataStream::FromResult(result);
uint8_t buffer[16000];
uint32_t filledSize = 0;
while ((filledSize = audioDataStream->ReadData(buffer, sizeof(buffer))) > 0)
{
    cout << filledSize << " bytes received." << endl;
}
```

You can use the `PullAudioOutputStream`, `PushAudioOutputStream`, the `Synthesizing` event, and `AudioDateStream` of the Speech SDK to enable streaming.

Taking `AudioDateStream` as an example:

```
SpeechSynthesizer synthesizer = new SpeechSynthesizer(config, null);
SpeechSynthesisResult result = synthesizer.StartSpeakingTextAsync(text).get();
AudioDataStream audioDataStream = AudioDataStream.fromResult(result);
byte[] buffer = new byte[16000];
long filledSize = audioDataStream.readData(buffer);
while (filledSize > 0) {
    System.out.println(filledSize + " bytes received.");
    filledSize = audioDataStream.readData(buffer);
}
```

You can use the `PullAudioOutputStream`, `PushAudioOutputStream`, the `Synthesizing` event, and `AudioDateStream` of the Speech SDK to enable streaming.

Taking `AudioDateStream` as an example:

```
speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config, audio_config=None)
result = speech_synthesizer.start_speaking_text_async(text).get()
audio_data_stream = speechsdk.AudioDataStream(result)
audio_buffer = bytes(16000)
filled_size = audio_data_stream.read_data(audio_buffer)
while filled_size > 0:
    print("{} bytes received.".format(filled_size))
    filled_size = audio_data_stream.read_data(audio_buffer)
```

You can use the `SPXPullAudioOutputStream`, `SPXPushAudioOutputStream`, the `Synthesizing` event, and `SPXAudioDataStream` of the Speech SDK to enable streaming.

Taking `AudioDateStream` as an example:

```
SPXSpeechSynthesizer *synthesizer = [[SPXSpeechSynthesizer alloc] initWithSpeechConfiguration:speechConfig
audioConfiguration:nil];
SPXSpeechSynthesisResult *speechResult = [synthesizer startSpeakingText:inputText];
SPXAudioDataStream *stream = [[SPXAudioDataStream alloc] initFromSynthesisResult:speechResult];
NSMutableData* data = [[NSMutableData alloc]initWithCapacity:16000];
while ([stream readData:data length:16000] > 0) {
    // Read data here
}
```

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.

```
using (var synthesizer = new SpeechSynthesizer(uspConfig, null as AudioConfig))
{
    using (var connection = Connection.FromSpeechSynthesizer(synthesizer))
    {
        connection.Open(true);
    }
    await synthesizer.SpeakTextAsync(text);
}
```

```
auto synthesizer = SpeechSynthesizer::FromConfig(config, nullptr);
auto connection = Connection::FromSpeechSynthesizer(synthesizer);
connection->Open(true);
```

```
SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, (AudioConfig) null);
Connection connection = Connection.fromSpeechSynthesizer(synthesizer);
connection.openConnection(true);
```

```
synthesizer = speechsdk.SpeechSynthesizer(config, None)
connection = speechsdk.Connection.from_speech_synthesizer(synthesizer)
connection.open(True)
```

```
SPXSpeechSynthesizer* synthesizer = [[SPXSpeechSynthesizer
alloc]initWithSpeechConfiguration:self.speechConfig audioConfiguration:nil];
SPXConnection* connection = [[SPXConnection alloc]initFromSpeechSynthesizer:synthesizer];
[connection open:true];
```

> **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 and `GStreamer` is properly installed. Refer this instruction to install and configure `GStreamer` for Speech SDK.

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. For example, we fixed a `TCP_NODELAY` setting issue in 1.16.0, which reduced extra one route trip time.

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

Long Audio API

11/23/2021 • 10 minutes to read • Edit Online

The Long Audio API provides asynchronous synthesis of long-form text to speech (for example: audio books, news articles and documents). This API doesn't return synthesized audio in real time. Instead, you poll for the response(s) and consume the output(s) as the service makes them available. Unlike the Text-to-speech API used by the Speech SDK, the Long Audio API can create synthesized audio longer than 10 minutes. This makes it ideal for publishers and audio content platforms to create long audio content like audio books in a batch.

More benefits of the Long Audio API:

- Synthesized speech returned by the service uses the best neural voices.
- There's no need to deploy a voice endpoint.

> **NOTE**
>
> The Long Audio API supports both Public Neural Voices and Custom Neural Voices.

Workflow

When using the Long Audio API, you'll typically submit a text file or files to be synthesized, poll for the status, and download the audio output when the status indicates success.

This diagram provides a high-level overview of the workflow.

Prepare content for synthesis

When preparing your text file, make sure it:

- Is either plain text (.txt) or SSML text (.txt).
- Is encoded as UTF-8 with Byte Order Mark (BOM).
- Is a single file, not a zip.
- Contains more than 400 characters for plain text or 400 billable characters for SSML text, and less than 10,000 paragraphs.
  - For plain text, each paragraph is separated by hitting **Enter/Return**. See plain text input example.
  - For SSML text, each SSML piece is considered a paragraph. Separate SSML pieces by different paragraphs. See SSML text input example.

Sample code

The rest of this page focuses on Python, but sample code for the Long Audio API is available on GitHub for the following programming languages:

- Sample code: Python
- Sample code: C#
- Sample code: Java

Python example

This section contains Python examples that show the basic usage of the Long Audio API. Create a new Python project using your favorite IDE or editor. Then copy this code snippet into a file named `long_audio_synthesis_client.py`.

```
import json
import ntpath
import requests
```

These libraries are used to construct the HTTP request, and call the text-to-speech long audio synthesis REST API.

Get a list of supported voices

To get a list of supported voices, send a GET request to `https://<endpoint>/api/texttospeech/v3.0/longaudiosynthesis/voices`.

This code gets a full list of voices you can use at a specific region/endpoint.

```
def get_voices():
    region = '<region>'
    key = '<your_key>'
    url =
'https://{}.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/longaudiosynthesis/voices'.format(reg
ion)
    header = {
        'Ocp-Apim-Subscription-Key': key
    }

    response = requests.get(url, headers=header)
    print(response.text)

get_voices()
```

Replace the following values:

- Replace `<your_key>` with your Speech service subscription key. This information is available in the **Overview** tab for your resource in the Azure portal.
- Replace `<region>` with the region where your Speech resource was created (for example: `eastus` or `westus`). This information is available in the **Overview** tab for your resource in the Azure portal.

You'll see output that looks like this:

```
{
  "values": [
    {
      "locale": "en-US",
      "voiceName": "en-US-AriaNeural",
      "description": "",
      "gender": "Female",
      "createdDateTime": "2020-05-21T05:57:39.123Z",
      "properties": {
        "publicAvailable": true
      }
    },
    {
      "id": "8fafd8cd-5f95-4a27-a0ce-59260f873141"
      "locale": "en-US",
      "voiceName": "my custom neural voice",
      "description": "",
      "gender": "Male",
      "createdDateTime": "2020-05-21T05:25:40.243Z",
      "properties": {
        "publicAvailable": false
      }
    }
  ]
}
```

If **properties.publicAvailable** is **true**, the voice is a public neural voice. Otherwise, it's a custom neural voice.

Convert text to speech

Prepare an input text file, in either plain text or SSML text, then add the following code to `long_audio_synthesis_client.py`:

> **NOTE**
>
> `concatenateResult` is an optional parameter. If this parameter isn't set, the audio outputs will be generated per paragraph. You can also concatenate the audios into one output by including the parameter. `outputFormat` is also optional. By default, the audio output is set to `riff-16khz-16bit-mono-pcm`. For more information about supported audio output formats, see Audio output formats.

```
def submit_synthesis():
    region = '<region>'
    key = '<your_key>'
    input_file_path = '<input_file_path>'
    locale = '<locale>'
    url =
'https://{}.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/longaudiosynthesis'.format(region)
    header = {
        'Ocp-Apim-Subscription-Key': key
    }

    voice_identities = [
        {
            'voicename': '<voice_name>'
        }
    ]

    payload = {
        'displayname': 'long audio synthesis sample',
        'description': 'sample description',
        'locale': locale,
        'voices': json.dumps(voice_identities),
        'outputformat': 'riff-16khz-16bit-mono-pcm',
        'concatenateresult': True,
    }

    filename = ntpath.basename(input_file_path)
    files = {
        'script': (filename, open(input_file_path, 'rb'), 'text/plain')
    }

    response = requests.post(url, payload, headers=header, files=files)
    print('response.status_code: %d' % response.status_code)
    print(response.headers['Location'])

submit_synthesis()
```

Replace the following values:

- Replace `<your_key>` with your Speech service subscription key. This information is available in the **Overview** tab for your resource in the Azure portal.
- Replace `<region>` with the region where your Speech resource was created (for example: `eastus` or `westus`). This information is available in the **Overview** tab for your resource in the Azure portal.
- Replace `<input_file_path>` with the path to the text file you've prepared for text-to-speech.
- Replace `<locale>` with the desired output locale. For more information, see language support.

Use one of the voices returned by your previous call to the `/voices` endpoint.

- If you are using public neural voice, replace `<voice_name>` with the desired output voice.
- To use a custom neural voice, replace `voice_identities` variable with following, and replace `<voice_id>` with the `id` of your custom neural voice.

```
voice_identities = [
    {
        'id': '<voice_id>'
    }
]
```

You'll see output that looks like this:

```
response.status_code: 202
https://<endpoint>/api/texttospeech/v3.0/longaudiosynthesis/<guid>
```

> **NOTE**
>
> If you have more than one input file, you will need to submit multiple requests, and there are limitations to consider.
>
> - The client can submit up to **5** requests per second for each Azure subscription account. If it exceeds the limitation, a **429 error code (too many requests)** is returned. Reduce the rate of submissions to avoid this limit.
> - The server can queue up to **120** requests for each Azure subscription account. If the queue exceeds this limitation, server will return **429 error code(too many requests)**. Wait for completed requests before submitting additional requests.

You can use the URL in output to get the request status.

Get details about a submitted request

To get status of a submitted synthesis request, send a GET request to the URL returned in previous step.

```
def get_synthesis():
    url = '<url>'
    key = '<your_key>'
    header = {
        'Ocp-Apim-Subscription-Key': key
    }
    response = requests.get(url, headers=header)
    print(response.text)

get_synthesis()
```

Output will be like this:

```
response.status_code: 200
{
  "models": [
    {
      "voiceName": "en-US-AriaNeural"
    }
  ],
  "properties": {
    "outputFormat": "riff-16khz-16bit-mono-pcm",
    "concatenateResult": false,
    "totalDuration": "PT5M57.252S",
    "billableCharacterCount": 3048
  },
  "id": "eb3d7a81-ee3e-4e9a-b725-713383e71677",
  "lastActionDateTime": "2021-01-14T11:12:27.240Z",
  "status": "Succeeded",
  "createdDateTime": "2021-01-14T11:11:02.557Z",
  "locale": "en-US",
  "displayName": "long audio synthesis sample",
  "description": "sample description"
}
```

The `status` property changes from `NotStarted` status, to `Running`, and finally to `Succeeded` or `Failed`. You can poll this API in a loop until the status becomes `Succeeded` or `Failed`.

Download audio result

Once a synthesis request succeeds, you can download the audio result by calling the GET `/files` API.

```
def get_files():
    id = '<request_id>'
    region = '<region>'
    key = '<your_key>'
    url =
'https://{}.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/longaudiosynthesis/{}/files'.format(r
egion, id)
    header = {
        'Ocp-Apim-Subscription-Key': key
    }

    response = requests.get(url, headers=header)
    print('response.status_code: %d' % response.status_code)
    print(response.text)

get_files()
```

Replace `<request_id>` with the ID of request you want to download the result. It can be found in the response of previous step.

Output will be like this:

```
response.status_code: 200
{
  "values": [
    {
      "name": "2779f2aa-4e21-4d13-8afb-6b3104d6661a.txt",
      "kind": "LongAudioSynthesisScript",
      "properties": {
        "size": 4200
      },
      "createdDateTime": "2021-01-14T11:11:02.410Z",
      "links": {
        "contentUrl": "https://customvoice-usw.blob.core.windows.net/artifacts/input.txt?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"
      }
    },
    {
      "name": "voicesynthesis_waves.zip",
      "kind": "LongAudioSynthesisResult",
      "properties": {
        "size": 9290000
      },
      "createdDateTime": "2021-01-14T11:12:27.226Z",
      "links": {
        "contentUrl": "https://customvoice-usw.blob.core.windows.net/artifacts/voicesynthesis_waves.zip?
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"
      }
    }
  ]
}
```

This example output contains information for two files. The one with `"kind": "LongAudioSynthesisScript"` is the input script submitted. The other one with `"kind": "LongAudioSynthesisResult"` is the result of this request.

The result is zip that contains the audio output files generated, along with a copy of the input text.

Both files can be downloaded from the URL in their `links.contentUrl` property.

Get all synthesis requests

The following code lists all submitted requests:

```
def get_synthesis():
    region = '<region>'
    key = '<your_key>'
    url =
'https://{}.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/longaudiosynthesis/'.format(region)
    header = {
        'Ocp-Apim-Subscription-Key': key
    }

    response = requests.get(url, headers=header)
    print('response.status_code: %d' % response.status_code)
    print(response.text)

get_synthesis()
```

Output will be like:

```
response.status_code: 200
{
  "values": [
    {
      "models": [
        {
          "id": "8fafd8cd-5f95-4a27-a0ce-59260f873141",
          "voiceName": "my custom neural voice"
        }
      ],
      "properties": {
        "outputFormat": "riff-16khz-16bit-mono-pcm",
        "concatenateResult": false,
        "totalDuration": "PT1S",
        "billableCharacterCount": 5
      },
      "id": "f9f0bb74-dfa5-423d-95e7-58a5e1479315",
      "lastActionDateTime": "2021-01-05T07:25:42.433Z",
      "status": "Succeeded",
      "createdDateTime": "2021-01-05T07:25:13.600Z",
      "locale": "en-US",
      "displayName": "Long Audio Synthesis",
      "description": "Long audio synthesis sample"
    },
    {
      "models": [
        {
          "voiceName": "en-US-AriaNeural"
        }
      ],
      "properties": {
        "outputFormat": "riff-16khz-16bit-mono-pcm",
        "concatenateResult": false,
        "totalDuration": "PT5M57.252S",
        "billableCharacterCount": 3048
      },
      "id": "eb3d7a81-ee3e-4e9a-b725-713383e71677",
      "lastActionDateTime": "2021-01-14T11:12:27.240Z",
      "status": "Succeeded",
      "createdDateTime": "2021-01-14T11:11:02.557Z",
      "locale": "en-US",
      "displayName": "long audio synthesis sample",
      "description": "sample description"
    }
  ]
}
```

The `values` property lists your synthesis requests. The list is paginated, with a maximum page size of 100. If there are more than 100 requests, a `"@nextLink"` property is provided to get the next page of the paginated list.

```
  "@nextLink": "https://<endpoint>/api/texttospeech/v3.0/longaudiosynthesis/?top=100&skip=100"
```

You can also customize page size and skip number by providing `skip` and `top` in URL parameter.

Remove previous requests

The service will keep up to `20,000` requests for each Azure subscription account. If your request amount exceeds this limitation, remove previous requests before making new ones. If you don't remove existing requests, you'll receive an error notification.

The following code shows how to remove a specific synthesis request.

```
def delete_synthesis():
    id = '<request_id>'
    region = '<region>'
    key = '<your_key>'
    url =
'https://{}.customvoice.api.speech.microsoft.com/api/texttospeech/v3.0/longaudiosynthesis/{}/'.format(region
, id)
    header = {
        'Ocp-Apim-Subscription-Key': key
    }

    response = requests.delete(url, headers=header)
    print('response.status_code: %d' % response.status_code)
```

If the request is successfully removed, the response status code will be HTTP 204 (No Content).

```
response.status_code: 204
```

> **NOTE**
>
> Requests with a status of `NotStarted` or `Running` cannot be removed or deleted.

The completed `long_audio_synthesis_client.py` is available on GitHub.

HTTP status codes

The following table details the HTTP response codes and messages from the REST API.

API	HTTP STATUS CODE	DESCRIPTION	SOLUTION
Create	400	The voice synthesis is not enabled in this region.	Change the speech subscription key with a supported region.
	400	Only the **Standard** speech subscription for this region is valid.	Change the speech subscription key to the "Standard" pricing tier.
	400	Exceed the 20,000 request limit for the Azure account. Remove some requests before submitting new ones.	The server will keep up to 20,000 requests for each Azure account. Delete some requests before submitting new ones.
	400	This model cannot be used in the voice synthesis: {modelID}.	Make sure the {modelID}'s state is correct.
	400	The region for the request does not match the region for the model: {modelID}.	Make sure the {modelID}'s region match with the request's region.
	400	The voice synthesis only supports the text file in the UTF-8 encoding with the byte-order marker.	Make sure the input files are in UTF-8 encoding with the byte-order marker.

API	HTTP STATUS CODE	DESCRIPTION	SOLUTION
	400	Only valid SSML inputs are allowed in the voice synthesis request.	Make sure the input SSML expressions are correct.
	400	The voice name {voiceName} is not found in the input file.	The input SSML voice name is not aligned with the model ID.
	400	The number of paragraphs in the input file should be less than 10,000.	Make sure the number of paragraphs in the file is less than 10,000.
	400	The input file should be more than 400 characters.	Make sure your input file exceeds 400 characters.
	404	The model declared in the voice synthesis definition cannot be found: {modelID}.	Make sure the {modelID} is correct.
	429	Exceed the active voice synthesis limit. Wait until some requests finish.	The server is allowed to run and queue up to 120 requests for each Azure account. Wait and avoid submitting new requests until some requests are completed.
All	429	There are too many requests.	The client is allowed to submit up to 5 requests to server per second for each Azure account. Reduce the request amount per second.
Delete	400	The voice synthesis task is still in use.	You can only delete requests that are **Completed** or **Failed**.
GetByID	404	The specified entity cannot be found.	Make sure the synthesis ID is correct.

Regions and endpoints

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.customvoice.api.speech.microsoft.com`
Southeast Asia	`https://southeastasia.customvoice.api.speech.microsoft.com`
UK South	`https://uksouth.customvoice.api.speech.microsoft.com`

REGION	ENDPOINT
West Europe	`https://westeurope.customvoice.api.speech.microsoft.com`

Audio output formats

We support flexible audio output formats. You can generate audio outputs per paragraph or concatenate the audio outputs into a single output by setting the `concatenateResult` parameter. The following audio output formats are supported by the Long Audio API:

> **NOTE**
> The default audio format is riff-16khz-16bit-mono-pcm.

- riff-8khz-16bit-mono-pcm
- riff-16khz-16bit-mono-pcm
- riff-24khz-16bit-mono-pcm
- riff-48khz-16bit-mono-pcm
- audio-16khz-32kbitrate-mono-mp3
- audio-16khz-64kbitrate-mono-mp3
- audio-16khz-128kbitrate-mono-mp3
- audio-24khz-48kbitrate-mono-mp3
- audio-24khz-96kbitrate-mono-mp3
- audio-24khz-160kbitrate-mono-mp3

Get facial pose events

11/23/2021 • 5 minutes to read • Edit Online

> **NOTE**
>
> Viseme events are only available for `en-US` English (United States) neural voices for now.

A *viseme* is the visual description of a phoneme in spoken language. It defines the position of the face and mouth when speaking a word. Each viseme depicts the key facial poses for a specific set of phonemes. Viseme can be used to control the movement of 2D and 3D avatar models, perfectly matching mouth movements to synthetic speech.

Visemes make avatars easier to use and control. Using visemes, 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 to 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.

See the introduction video of the viseme.

Azure neural TTS can produce visemes with speech

A neural voice turns input text or SSML (Speech Synthesis Markup Language) into synthesized speech. Speech audio output can be accompanied by viseme IDs and their offset timestamps. Each viseme ID specifies a specific pose in observed speech, such as the position of the lips, jaw, and tongue when producing a particular phoneme. 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 flowchart below.

PARAMETER	DESCRIPTION
Viseme ID	Integer number that specifies a viseme. In English (United States), we offer 22 different visemes to depict the mouth shapes for a specific set of phonemes. There is no one-to-one correspondence between visemes and phonemes. Often several phonemes correspond to a single viseme, as several phonemes look the same on the face when produced, such as `s` and `z`. See the mapping table between Viseme ID and phonemes.
Audio offset	The start time of each viseme, in ticks (100 nanoseconds).

Get viseme events with the Speech SDK

To get viseme with your synthesized speech, subscribe to the `VisemeReceived` event in Speech SDK. The following snippet shows how to subscribe to the viseme event.

```
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}.");
    };

    var result = await synthesizer.SpeakSsmlAsync(ssml));
}
```

```
auto synthesizer = SpeechSynthesizer::FromConfig(speechConfig, audioConfig);

// Subscribes to viseme received event
synthesizer->VisemeReceived += [](const SpeechSynthesisVisemeEventArgs& e)
{
    cout << "viseme event received. "
        // The unit of e.AudioOffset is tick (1 tick = 100 nanoseconds), divide by 10,000 to convert to
milliseconds.
        << "Audio offset: " << e.AudioOffset / 10000 << "ms, "
        << "viseme id: " << e.VisemeId << "." << endl;
};

auto result = synthesizer->SpeakSsmlAsync(ssml).get();
```

```
SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, audioConfig);

// Subscribes to viseme received event
synthesizer.VisemeReceived.addEventListener((o, e) -> {
    // The unit of e.AudioOffset is tick (1 tick = 100 nanoseconds), divide by 10,000 to convert to
milliseconds.
    System.out.print("Viseme event received. Audio offset: " + e.getAudioOffset() / 10000 + "ms, ");
    System.out.println("viseme id: " + e.getVisemeId() + ".");
});

SpeechSynthesisResult result = synthesizer.SpeakSsmlAsync(ssml).get();
```

```
speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config, audio_config=audio_config)

# Subscribes to viseme received event
speech_synthesizer.viseme_received.connect(lambda evt: print(
    "Viseme event received: audio offset: {}ms, viseme id: {}.".format(evt.audio_offset / 10000,
evt.viseme_id)))

result = speech_synthesizer.speak_ssml_async(ssml).get()
```

```
var synthesizer = new SpeechSDK.SpeechSynthesizer(speechConfig, audioConfig);

// Subscribes to viseme received event
synthesizer.visemeReceived = function (s, e) {
    window.console.log("(Viseme), Audio offset: " + e.audioOffset / 10000 + "ms. Viseme ID: " + e.visemeId);
}

synthesizer.speakSsmlAsync(ssml);
```

```
SPXSpeechSynthesizer *synthesizer =
    [[SPXSpeechSynthesizer alloc] initWithSpeechConfiguration:speechConfig
                                          audioConfiguration:audioConfig];

// Subscribes to viseme received event
[synthesizer addVisemeReceivedEventHandler: ^ (SPXSpeechSynthesizer *synthesizer,
SPXSpeechSynthesisVisemeEventArgs *eventArgs) {
    NSLog(@"Viseme event received. Audio offset: %fms, viseme id: %lu.", eventArgs.audioOffset/10000.,
eventArgs.visemeId);
}];

[synthesizer speakSsml:ssml];
```

Here is an example of the viseme output.

```
(Viseme), Viseme ID: 1, Audio offset: 200ms.

(Viseme), Viseme ID: 5, Audio offset: 850ms.

……

(Viseme), Viseme ID: 13, Audio offset: 2350ms.
```

After obtaining the viseme output, you can use these events to drive character animation. You can build your own characters and automatically animate the characters.

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 provided in viseme event, these well-designed SVGs will be processed with smoothing modifications, and provide robust animation to the users. For example, below illustration shows a red lip character designed for language learning.

For 3D characters, think of the characters as string puppets. The puppet master pulls the strings from one state to another and the laws of physics do the rest and drive the puppet to move fluidly. The viseme output acts as a puppet master to provide an action timeline. The animation engine defines the physical laws of action. By interpolating frames with easing algorithms, the engine can further generate high-quality animations.

Map phonemes to visemes

Visemes vary by language. Each language has a set of visemes that correspond to its specific phonemes. The following table shows the correspondence between International Phonetic Alphabet (IPA) phonemes and viseme IDs for English (United States).

IPA	EXAMPLE	VISEME ID
i	**ea**t	6
ɪ	**i**f	6
eɪ	**a**te	4
ɛ	**e**very	4
æ	**a**ctive	1
ɑ	**o**bstinate	2
ɔ	**ca**use	3

IPA	EXAMPLE	VISEME ID
ʊ	b**oo**k	4
oʊ	**o**ld	8
u	**U**ber	7
ʌ	**u**ncle	1
aɪ	**i**ce	11
aʊ	**ou**t	9
ɔɪ	**oi**l	10
ju	**Yu**ma	[6, 7]
ə	**a**go	1
ɪɹ	**ear**s	[6, 13]
ɛɹ	**air**plane	[4, 13]
ʊɹ	c**ur**e	[4, 13]
aɪ(ə)ɹ	**Ire**land	[11, 13]
aʊ(ə)ɹ	**hour**s	[9, 13]
ɔɹ	**or**ange	[3, 13]
ɑɹ	**ar**tist	[2, 13]
ɝ	**ear**th	[5, 13]
ɚ	all**er**gy	[1, 13]
w	**w**ith, s**ue**de	7
j	**y**ard, f**e**w	6
p	**p**ut	21
b	**b**ig	21
t	**t**alk	19
d	**d**ig	19
k	**c**ut	20

IPA	EXAMPLE	VISEME ID
g	go	20
m	mat, smash	21
n	no, snow	19
ŋ	link	20
f	fork	18
v	value	18
θ	thin	17
ð	then	17
s	sit	15
z	zap	15
ʃ	she	16
ʒ	Jacques	16
h	help	12
tʃ	chin	16
dʒ	joy	16
l	lid, glad	14
ɹ	red, bring	13

Next steps

- Speech SDK reference documentation

What is Custom Neural Voice?

11/23/2021 • 4 minutes to read • Edit Online

Custom Neural Voice is a text-to-speech (TTS) 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 by providing your audio samples as training data. Based on the Neural TTS technology and the multi-lingual multi-speaker universal model, Custom Neural Voice lets you 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 and cross-lingual feature.

> **NOTE**
>
> The Custom Neural Voice feature requires registration, and access to it is limited based upon Microsoft's eligibility and use criteria. Customers who wish to use this feature are required to register their use cases through the intake form.

The basics of Custom Neural Voice

The underlying Neural TTS technology used for Custom Neural Voice consists of three major components: Text Analyzer, Neural Acoustic Model, and Neural Vocoder. To generate natural synthetic speech from text, text is first input into 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, such as 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 TTS voice models are trained using deep neural networks based on the recording samples of human voices. In this blog, we describe how Neural TTS works with state-of-the-art neural speech synthesis models. The blog also explains how a universal base model can be adapted to a target speaker's voice with less than 2 hours of speech data (or less than 2,000 recorded utterances), and additionally transfer the voice to another language or style. To read about how a neural vocoder is trained, see the blog post.

Custom Neural Voice lets you adapt the Neural TTS engine to fit your scenarios. 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. Custom Neural Voice can use text provided by the user to convert text into speech in real-time, or generate audio content offline with text input. This is made available via the REST API, the Speech SDK, or the web portal.

Get started

- To get started with Custom Neural Voice and create a project, see Get started with Custom Neural Voice.

- To prepare and upload your audio data, see Prepare training data.
- To train and deploy your models, see Create and use your voice model.

Terms and definitions

TERM	DEFINITION
Voice model	A text-to-speech model that can mimic the unique vocal characteristics of a target speaker. A *voice model* is also known as a *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 of a human voice.
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.
Standard TTS	The standard, or "traditional," method of TTS that breaks down spoken language into phonetic snippets so that they can be remixed and matched using classical programming or statistical methods.
Neural TTS	Neural TTS synthesizes speech using deep neural networks that have "learned" the way phonetics are combined in natural human speech, rather than using procedural programming or statistical methods. In addition to the recordings of a target voice talent, Neural TTS uses a source library/base model that is built with voice recordings from many different speakers.
Training data	A custom neural voice training dataset that includes the audio recordings of the voice talent, and the associated text transcriptions.
Persona	A persona describes who you want this voice to be. A good persona design will inform all voice creation whether it's choosing an available voice model already created, or starting from scratch by casting and recording a new voice talent.
Script	A script is a text file that contains the utterances to be spoken by your voice talent. (The term "*utterances*" encompasses both full sentences and shorter phrases.)

Responsible use of AI

To learn how to use Custom Neural Voice responsibly, see the transparency note. 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.

Next steps

- Get started with Custom Neural Voice

Get started with Custom Neural Voice

11/23/2021 • 6 minutes to read • Edit Online

Custom Neural Voice is a set of online tools that allow you to create a recognizable, one-of-a-kind voice for your brand. All it takes to get started are a handful of audio files and the associated transcriptions. Follow the links below to start creating a custom text-to-speech experience. See the supported languages and regions for Custom Neural Voice.

> **NOTE**
>
> As part of Microsoft's commitment to designing responsible AI, we have limited the use of Custom Neural Voice. You may gain access to the technology only after your applications are reviewed and you have committed to using it in alignment with our responsible AI principles. Learn more about our policy on the limit access and apply here.

Set up your Azure account

A Speech service subscription is required before you can use Custom Neural Voice. Follow these instructions to create a Speech service subscription in Azure. If you do not have an Azure account, you can sign up for a new one.

Once you've created an Azure account and a Speech service subscription, you'll need to sign in Speech Studio and connect your subscription.

1. Get your Speech service subscription key from the Azure portal.
2. Sign in to Speech Studio, then click **Custom Voice**.
3. Select your subscription and create a speech project.
4. If you'd like to switch to another Speech subscription, use the cog icon located in the top navigation.

> **NOTE**
>
> You must have a F0 or a S0 Speech service key created in Azure before you can use the service. Custom Neural Voice only supports the S0 tier.

Create a project

Content like data, models, tests, and endpoints are organized into **Projects** in Speech Studio. Each project is specific to a country/language and the gender of the voice you want to create. For example, you may create a project for a female voice for your call center's chat bots that use English in the United States ('en-US').

To create your first project, select the **Text-to-Speech/Custom Voice** tab, then click **Create project**. Follow the instructions provided by the wizard to create your project. After you've created a project, you will see four tabs: **Set up voice talent**, **Prepare training data**, **Train model**, and **Deploy model**. Use the links provided in next steps to learn how to use each tab.

Tips for creating a custom neural voice

Creating a great custom voice requires careful quality control in each step, from voice design and data preparation, to the deployment of the voice model to your system. Below are some key steps to take when creating a custom neural voice for your organization.

Persona design

First, design a persona of the voice that represents your brand using a persona brief document that defines elements such as the features of the voice, and the character behind the voice. This will help to guide the process of creating a custom voice model, including defining the scripts, selecting your voice talent, training and voice tuning.

Script selection

Carefully 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 are creating a customer service bot. Include different sentence types in your scripts, including statements, questions, exclamations, etc.

Preparing training data

We recommend that the audio recordings be captured in a professional quality recording studio to achieve a high signal-to-noise ratio. The quality of the voice model heavily depends on your training data. Consistent volume, speaking rate, pitch, and consistency in expressive mannerisms of speech are required.

Once the recordings are ready, follow Prepare training data to prepare the training data in the right format.

Training

Once you have prepared the training data, go to Speech Studio to create your custom neural voice. You need to 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 the errors and submit again.

Testing

Prepare test scripts for your voice model that cover the different use cases for your apps. It's recommended that you use scripts within and outside the training dataset so you can test the quality more broadly for different content.

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 generate synthetic speech. SSML is the markup language used to communicate with the TTS 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.

Migrate to Custom Neural Voice

The standard/non-neural training tier (statistical parametric, concacenative) of Custom Voice is being deprecated. The announcement has been sent out to all existing Speech subscriptions before 2/28/2021. During the deprecation period (3/1/2021 - 2/29/2024), existing standard tier users can continue to use their non-neural models created. All new users/new speech resources should move to the neural tier/Custom Neural Voice. After 2/29/2024, all standard/non-neural custom voices will no longer be supported.

If you are using non-neural/standard Custom Voice, migrate to Custom Neural Voice immediately following the steps below. Moving to Custom Neural Voice will help you develop more realistic voices for even more natural conversational interfaces and enable your customers and end users to benefit from the latest Text-to-Speech technology, in a responsible way.

1. Learn more about our policy on the limit access and apply here. Note that the access to the Custom Neural Voice service is subject to Microsoft's sole discretion based on our eligibility criteria. Customers may gain access to the technology only after their application is reviewed and they have committed to using it in alignment with our Responsible AI principles and the code of conduct.

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.

> **IMPORTANT**
>
> To protect voice talent and prevent training of voice models with unauthorized recording or without the acknowledgement from the voice talent, we require the customer to upload a recorded statement of the voice talent giving their consent. When preparing your recording script, make sure you include this sentence. "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." This sentence must be uploaded to the **Set up voice talent** tab as a verbal consent file. It will be used to verify if the recordings in your training datasets are done by the same person that makes the consent.

3. After the Custom Neural Voice model is created, deploy the voice model to a new endpoint. To create a new custom voice endpoint with your neural voice model, go to **Text-to-Speech > Custom Voice > Deploy model**. Select **Deploy models** and enter a **Name** and **Description** for your custom endpoint. Then select the custom neural voice model you would like to associate with this endpoint and confirm the deployment.
4. Update your code in your apps if you have created a new endpoint with a new model.

Next steps

- Prepare data for Custom Neural Voice
- Train and deploy a Custom Neural Voice
- How to record voice samples

Prepare training data

11/23/2021 • 9 minutes to read • Edit Online

When you're ready to create a custom text-to-speech voice for your application, the first step is to gather audio recordings and associated scripts to start training the voice model. The Speech service uses this data to create a unique voice tuned to match the voice in the recordings. After you've trained the voice, you can start synthesizing speech in your applications.

Voice talent verbal statement

Before you can train your own text-to-speech voice model, you'll need audio recordings and the associated text transcriptions. On this page, we'll review data types, how they are used, and how to manage each.

> **NOTE**
>
> 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 below sentence.

> "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." This sentence will be used to verify if the training data is done by the same person that makes the consent. Read more about the voice talent verification here.

> Custom Neural Voice is available with limited access. Make sure you understand the responsible AI requirements and apply the access here.

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. We provide tools (beta) to help you segment your audio into utterances and prepare transcripts using the Batch Transcription API.

This table lists data types and how each is used to create a custom text-to-speech voice model.

DATA TYPE	DESCRIPTION	WHEN TO USE	ADDITIONAL 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	ADDITIONAL PROCESSING REQUIRED
Long audio + transcript (beta)	A collection (.zip) of long, unsegmented audio files (longer than 20 seconds), paired with a collection (.zip) of transcripts that contains all spoken words.	You have audio files and matching transcripts, but they are not segmented into utterances.	Segmentation (using batch transcription). Audio format transformation where required.
Audio only (beta)	A collection (.zip) of audio files without a transcript.	You only have audio files available, without transcripts.	Segmentation + transcript generation (using batch transcription). Audio format transformation where required.

Files should be grouped by type into a dataset and uploaded as a zip file. Each dataset can only contain a single data type.

> **NOTE**
> The maximum number of datasets allowed to be imported per subscription is 10 zip files for free subscription (F0) users and 500 for standard subscription (S0) users.

Individual utterances + matching transcript

You can prepare recordings of individual utterances and the matching transcript in two ways. Either write a script and have it read by a voice talent or use publicly available audio and transcribe it to text. If you do the latter, edit disfluencies from the audio files, such as "um" and other filler sounds, stutters, mumbled words, or mispronunciations.

To produce a good voice 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.

> **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.

Audio files

Each audio file should contain a single utterance (a single sentence or a single turn of a dialog system), less than 15 seconds long. All files must be in the same spoken language. Multi-language custom text-to-speech voices are not supported, with the exception of the Chinese-English bi-lingual. Each audio file must have a unique numeric filename with the filename extension .wav.

Follow these guidelines when preparing audio.

PROPERTY	VALUE
File format	RIFF (.wav), grouped into a .zip file
Sampling rate	At least 16,000 Hz. For creating a neural voice, 24,000 Hz is required.

PROPERTY	VALUE
Sample format	PCM, 16-bit
File name	Numeric, with .wav extension. No duplicate file names allowed.
Audio length	Shorter than 15 seconds
Archive format	.zip
Maximum archive size	2048 MB

> **NOTE**
> .wav files with a sampling rate lower than 16,000 Hz will be rejected. If a .zip file contains .wav files with different sample rates, only those equal to or higher than 16,000 Hz will be imported. The portal currently imports .zip archives up to 2048 MB. However, multiple archives can be uploaded.

> **NOTE**
> The default sampling rate for a custom neural voice is 24,000 Hz. Your .wav files with a sampling rate lower than 16,000 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.

Transcripts

The transcription file is a plain text file. Use these guidelines to prepare your transcriptions.

PROPERTY	VALUE
File format	Plain text (.txt)
Encoding format	ANSI/ASCII, UTF-8, UTF-8-BOM, UTF-16-LE, or UTF-16-BE. For zh-CN, ANSI/ASCII and UTF-8 encodings are not supported.
# of utterances per line	**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 (beta)

In some cases, you may not have segmented audio available. We provide a service (beta) through the Speech Studio to help you segment long audio files and create transcriptions. Keep in mind, this service will be charged toward your speech-to-text subscription usage.

> **NOTE**
> The long-audio segmentation service will leverage the batch transcription feature of speech-to-text, which only supports standard subscription (S0) users. During the processing of the segmentation, your audio files and the transcripts will also be sent to the Custom Speech service to refine the recognition model so the accuracy can be improved for your data. No data will be retained during this process. After the segmentation is done, only the utterances segmented and their mapping transcripts will be stored for your downloading and training.

Audio files

Follow these guidelines when preparing audio for segmentation.

PROPERTY	VALUE
File format	RIFF (.wav) with a sampling rate of at least 16 khz-16-bit in PCM or .mp3 with a bit rate of at least 256 KBps, grouped into a .zip file
File name	ASCII and Unicode characters supported. No duplicate names allowed.
Audio length	Longer than 20 seconds
Archive format	.zip
Maximum archive size	2048 MB

> **NOTE**
> The default sampling rate for a custom neural voice is 24,000 Hz. Your .wav files with a sampling rate lower than 16,000 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.

Transcripts

Transcripts must be prepared to the specifications listed in this table. Each audio file must be matched with a transcript.

PROPERTY	VALUE
File format	Plain text (.txt), grouped into a .zip
File name	Use the same name as the matching audio file

PROPERTY	VALUE
Encoding format	UTF-8-BOM only
# of utterances per line	No limit
Maximum file size	2048 MB

All transcripts files in this data type should be grouped into a zip file. For example, you have uploaded a zip file containing an audio file named 'kingstory.wav', 45 seconds long, and another one named 'queenstory.mp3', 200 seconds long. You will need to upload another zip file containing two transcripts, one named 'kingstory.txt', the other one 'queenstory.txt'. Within each plain text file, you will provide the full correct transcription for the matching audio.

After your dataset is successfully uploaded, we will help you segment the audio file into utterances based on the transcript provided. You can check the segmented utterances and the matching transcripts by downloading the dataset. Unique IDs will be assigned to the segmented utterances automatically. It's important that you make sure the transcripts you provide are 100% accurate. Errors in the transcripts can reduce the accuracy during the audio segmentation and further introduce quality loss in the training phase that comes later.

Audio only (beta)

If you don't have transcriptions for your audio recordings, use the **Audio only** option to upload your data. Our system can help you segment and transcribe your audio files. Keep in mind, this service will 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) with a sampling rate of at least 16 khz-16-bit in PCM or .mp3 with a bit rate of at least 256 KBps, grouped into a .zip file
File name	ASCII and Unicode characters supported. No duplicate name allowed.
Audio length	Longer than 20 seconds
Archive format	.zip
Maximum archive size	2048 MB

> **NOTE**
> The default sampling rate for a custom neural voice is 24,000 Hz. Your .wav files with a sampling rate lower than 16,000 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 will help you segment the audio file into utterances based on our speech batch transcription service. Unique IDs will be assigned to the segmented utterances automatically. Matching transcripts will be generated through speech recognition. All .mp3 files will be transformed into the .wav format after processing. You can check the segmented utterances and the matching transcripts by downloading the dataset.

Next steps

- Create and use your voice model
- How to record voice samples

Create and use your voice model

11/23/2021 • 14 minutes to read • Edit Online

In Prepare training data, you learned about the different data types you can use to train a custom neural voice and the different format requirements. Once you've prepared your data and the voice talent verbal statement, you can start to upload them to the Speech Studio. In this article, you learn how to train a Custom Neural Voice through the Speech Studio portal. See the supported languages for custom neural voice.

Prerequisites

- Complete get started with Custom Neural Voice
- Prepare training data

Set up voice talent

A voice talent is an individual or target speaker whose voices are recorded and used to create neural voice models. Before you create a voice, define your voice persona and select a right voice talent. For details on recording voice samples, see the tutorial.

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 following sentence:

"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." This sentence is used to verify if the training data matches the audio in the consent statement. > Read more about the voice talent verification here.

> **NOTE**
> Custom Neural Voice is available with limited access. Make sure you understand the responsible AI requirements, and then apply for access.

The following steps assume you've prepared the voice talent verbal consent files. Go to Speech Studio to select a custom neural voice project, then follow the following steps to create a voice talent profile.

1. Navigate to **Text-to-Speech** > **Custom Voice** > **select a project** > **Set up voice talent**.
2. Select **Add voice talent**.
3. Next, to define voice characteristics, select **Target scenario** to be used. Then describe your **Voice characteristics**.

> **NOTE**
> The scenarios you provide must be consistent with what you've applied for in the application form.

4. Then, go to **Upload voice talent statement**, follow the instruction to upload voice talent statement you've prepared beforehand.

> **NOTE**
> Make sure the verbal statement is recorded in the same settings as your training data, including the recording environment and speaking style.

5. Finally, go to **Review and create**, you can review the settings and select **Submit**.

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. Data readiness checking will be done per each training set. You can import multiple data to a training set.

You can do the following to create and review your training data.

1. On the **Prepare training data** tab, select **Add training set** to enter **Name** and **Description** > **Create** to add a new training set.

   When the training set is successfully created, you can start to upload your data.

2. To upload data, select **Upload data** > **Choose data type** > **Upload data** and **Specify the target training set** > Enter **Name** and **Description** for your data > review the settings and select **Submit**.

> **NOTE**
> - Duplicate audio names will be 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 duplicate, they'll be rejected.
> - If you've created data files in the previous version of Speech Studio, you must specify a training set for your data in advance to use them. Or else, an exclamation mark will be appended to the data name, and the data could not be used.

Each 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 Custom Neural Voice service. Go to Prepare training data and make sure your data has been rightly formatted.

> **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 10 .zip files for free subscription (F0) users and 500 for standard subscription (S0) users.

Data files are automatically validated once you hit the **Submit** button. Data validation includes series of checks on the audio files to verify their file format, size, and sampling rate. Fix the errors if any and submit again.

Once the data is uploaded, 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 to 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.

A higher signal-to-noise ratio (SNR) indicates lower noise in your audio. You can typically reach a 50+ SNR by recording at professional studios. Audio with an SNR below 20 can result in obvious noise in your generated

voice.

Consider re-recording any utterances with low pronunciation scores or poor signal-to-noise ratios. If you can't re-record, consider excluding those utterances from your data.

On the **Data details**, you can check the data details of the training set. If there are any typical issues with the data, follow the instructions in the message displayed to fix them before training.

The issues are divided into three types. Referring to the following three tables to check the respective types of errors.

Manually fix the first type of errors listed in the table below, otherwise the data with these errors will be excluded during 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	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 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.
Audio	Invalid audio format	The audio format of the .wav files is invalid. Check the wav file format using an audio tool like SoX.
Audio	Low sampling rate	The sampling rate of the .wav files cannot be lower than 16 KHz.
Audio	Too long audio	Audio duration is longer than 30 seconds. Split the long audio into multiple files. We suggest utterances should be shorter than 15 seconds.
Audio	No valid audio	No valid audio is found in this dataset. Check your audio data and upload again.

The second type of errors listed in the table below will be automatically fixed, but double checking the fixed data is recommended.

CATEGORY	NAME	DESCRIPTION
Audio	Stereo audio auto fixed	Use mono in your audio sample recordings. Stereo audio channels are automatically merged into a mono channel, which can cause content loss. Download the normalized dataset and review it.
Volume	Volume peak auto fixed	The volume peak should be within the range of -3 dB (70% of max volume) to -6 dB (50%). Control the volume peak during the sample recording or data preparation. This audio is linearly scaled to fit the peak range automatically (-4 dB or 65%). Download the normalized dataset and review it.
Mismatch	Silence auto fixed	The start silence is detected to be longer than 200 ms, and has been trimmed to 200 ms automatically. Download the normalized dataset and review it.
Mismatch	Silence auto fixed	The end silence is detected to be longer than 200 ms, and has been trimmed to 200 ms automatically. Download the normalized dataset and review it.
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.

If the third type of errors listed in the table below aren't fixed, although the data with these errors won't be excluded during training, it will affect the quality of training. For higher-quality training, manually fixing these errors is recommended.

CATEGORY	NAME	DESCRIPTION
Script	Non-normalized text	This script contains digit 0-9. Expand them to normalized words and match with the audio. For example, normalize '123' to 'one hundred and twenty-three'.
Script	Non-normalized text	This script contains symbols {}. Normalize the symbols to match the audio. For example, '50%' to 'fifty percent'.

CATEGORY	NAME	DESCRIPTION
Script	Not enough question utterances	At least 10% 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% of the total utterances should be exclamation sentences. This helps the voice model properly express an excited tone.
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. It will be automatically upsampled to 24 KHz if it's lower.
Volume	Overall volume too low	Volume shouldn't be lower than -18 dB (10% 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 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 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.
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.

CATEGORY	NAME	DESCRIPTION
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.

Train your custom neural voice model

After your data files have been validated, you can use them to build your custom neural voice model.

1. On the **Train model** tab, select **Train model** to create a voice model with the data you've uploaded.

2. Select the neural training method for your model and target language.

By default, your voice model is trained in the same language of your training data. You can also select to create a secondary language (preview) for your voice model. Check the languages supported for custom neural voice and cross-lingual feature: language for customization.

Training of custom neural voices isn't free. Check the pricing for details. However, if you have statistical parametric or concatenative voice models deployed before 3/31/2021 with S0 Speech resources, free neural training credits are offered to your Azure subscription, and you can train 5 different versions of neural voices for free.

3. Next, choose the data you want to use for training, and specify a speaker file.

> **NOTE**
>
> - You need to select at least 300 utterances to create a custom neural voice.
> - To train a neural voice, you must specify a voice talent profile with the audio consent file provided of the voice talent acknowledging to use his/her speech data to train a custom voice model. Custom Neural Voice is available with limited access. Make sure you understand the responsible AI requirements and apply the access here.

4. Then, choose your test script.

Each training will generate 100 sample audio files automatically to help you test the model with a default script. You can also provide your own test script as optional. The test script must exclude the filenames (the ID of each utterance), otherwise, these IDs will be spoken. Below is an example of how the utterances are organized in one .txt file:

```
This is the waistline, and it's falling.
We have trouble scoring.
It was Janet Maslin.
```

Each paragraph of the utterance will result in a separate audio. If you want to combine all sentences into one audio, make them in one paragraph.

> **NOTE**
> - The test script must be a txt file, less than 1 MB. Supported encoding format includes ANSI/ASCII, UTF-8, UTF-8-BOM, UTF-16-LE, or UTF-16-BE.
> - The generated audios are a combination of the uploaded test script and the default test script.

5. Enter a **Name** and **Description** to help you identify this model.

Choose a name carefully. The name you enter here will be the name you use to specify the voice in your request for speech synthesis as part of the SSML input. Only letters, numbers, and a few punctuation characters such as -, _, and (', ') are allowed. Use different names for different neural voice models.

A common use of the **Description** field is to record the names of the data that were used to create the model.

6. Review the settings, then select **Submit** to start training the model.

> **NOTE**
> 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.

The **Train model** table displays a new entry that corresponds to this newly created model. The table also displays the status: Processing, Succeeded, Failed.

The status that's shown reflects the process of converting your data to a voice model, as shown here.

STATE	MEANING
Processing	Your voice model is being created.
Succeeded	Your voice model has been created and can be deployed.
Failed	Your voice model has been failed in training because of many reasons, for example, unseen data problems or network issues.

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.

> **NOTE**
> Standard subscription (S0) users can train three voices simultaneously. If you reach the limit, wait until at least one of your voice models finishes training, and then try again.

7. After you finish training the model successfully, you can review the model details.

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, 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, and more. Check here to learn more about the capabilities and limits of our technology and the best practice to improve your model quality.

Create and use a custom neural voice endpoint

After you've successfully created and tested your voice model, you deploy it in a custom Text-to-Speech endpoint. You then use this endpoint in place of the usual endpoint when making Text-to-Speech requests through the REST API. Your custom endpoint can be called only by the subscription that you've used to deploy the model.

You can do the following to create a custom neural voice endpoint.

1. On the **Deploy model** tab, select **Deploy model**.
2. Next, enter a **Name** and **Description** for your custom endpoint.
3. Then, select a voice model you would like to associate with this endpoint.
4. Finally, select **Deploy** to create your endpoint.

After you've clicked the **Deploy** button, in the endpoint table, you'll see an entry for your new endpoint. It may take a few minutes to instantiate a new endpoint. When the status of the deployment is **Succeeded**, the endpoint is ready for use.

You can **Suspend** and **Resume** your endpoint if you don't use it all the time. When an endpoint is reactivated after suspension, the endpoint URL will be kept the same so you don't need to change your code in your apps.

You can also update the endpoint to a new model. To change the model, make sure the new model is named the same as the one you want to update.

> **NOTE**
>
> - Standard subscription (S0) users can create up to 50 endpoints, 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 subscription to pass through the authentication of TTS service.

After your endpoint is deployed, the endpoint name appears as a link. Click the link to display information specific to your endpoint, such as the endpoint key, endpoint URL, and sample code.

The custom endpoint is functionally identical to the standard endpoint that's used for text-to-speech requests. For more information, see Speech SDK or REST API.

We also provide an online tool, Audio Content Creation, that allows you to fine-tune their audio output using a friendly UI.

Next steps

- How to record voice samples
- Text-to-Speech API reference
- Long Audio API

Record voice samples to create a custom voice

11/23/2021 • 22 minutes to read • Edit Online

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.

> **NOTE**
> To train a neural voice, you must specify a voice talent profile with the audio consent file provided of the voice talent acknowledging to use his/her speech data to train a custom neural voice model. When preparing your recording script, make sure you include the below sentence.

> "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." This sentence will be used to verify if the training data is done by the same person that makes the consent. Read more about the voice talent verification here.

> Custom Neural Voice is available with limited access. Make sure you understand the responsible AI requirements and apply the access here.

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 primarily filling the director role and hiring both a voice talent and a recording engineer. If you want to make the recordings yourself, this article includes some information about the recording engineer role. The editor role isn't needed until after the session, so can be performed by the director or the recording engineer.

Choose your voice talent

Actors with experience in voiceover or voice character work make good custom neural voice talent. You can also often find suitable talent among announcers and newsreaders. 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 is the other half of the equation. They must be able to speak with consistent rate, volume level, pitch, and tone. Clear diction is a must. The talent also needs to be able to strictly control their pitch variation, emotional affect, and speech mannerisms. Recording voice samples can be more fatiguing than other kinds of voice work. Most voice talent can record for two or three hours a day. Limit sessions to three or four a week, with a day off in-between if possible.

Work with your voice talent to develop a "persona" that defines the overall sound and emotional tone of the custom neural voice. In the process, you'll pinpoint what "neutral" sounds like for that persona. Using the Custom Neural Voice capability, you can train a model that speaks with emotions. Define the "speaking styles" and ask your voice talent to read the script in a way that resonates the styles you want.

A persona might have, for example, a naturally upbeat personality. So "their" voice might carry a note of optimism even when they speak neutrally. However, such a personality trait should be subtle and consistent. Listen to readings by existing voices to get an idea of what you're aiming for.

> **TIP**
>
> Usually, you'll want to own the voice recordings you make. Your voice talent should be amenable to a work-for-hire contract for the project.

Create a script

The starting point of any custom 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.)

The utterances in your script can come from anywhere: fiction, non-fiction, transcripts of speeches, news reports, and anything else available in printed form. If you want to make sure your voice does well on specific kinds of words (such as medical terminology or programming jargon), you might want to include sentences from scholarly papers or technical documents. For a brief discussion of potential legal issues, see the "Legalities" section. You can also write your own text.

Your utterances don't need to come from the same source, or the same kind of source. They don't even need to have anything to do with each other. However, if you'll 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 your 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. Building a custom neural voice requires at least 300 recorded sentences as training data.

You can select your domain-specific scripts from the sentences that your custom voice will be used to read.

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.

  In general, each sentence should contain 4 words to 30 words. It's required that no duplicate sentences are included in your script.
  For how to balance the different sentence types, refer to the following table.

SENTENCE TYPES	COVERAGE
Statement sentences	Statement sentences are the major part of the script, taking about 70-80% of all.
Question sentences	Question sentences should take about 10%-20% of your domain script, including 5%-10% of rising and 5%-10% of falling tones.
Exclamation sentences	Exclamation sentences should take about 10%-20% of your scripts.
Short word/phrase	Short word/phrase scripts should also take about 10% cases of the total utterances, with 5 to 7 words per case.

> **NOTE**
> Regarding short word/phrase, actually it means that single words or phrases should be included and separated with a comma. It helps a voice talent pause briefly at the comma when reading the scripts.

  Best practices include:

  - Balanced coverage for Part of Speech, like verb, noun, adjective, and so on.
  - Balanced coverage for pronunciations. Include all letters from A to Z so the TTS engine learns how to pronounce each letter in your defined style.
  - Readable, understandable, common-sense for speaker to read out.
  - Avoid too much similar pattern for word/phrase, like "easy" and "easier".
  - Include different format of numbers: address, unit, phone, quantity, date, and so on, in all sentence types.
  - Include spelling sentences if it's something your TTS voice will be used to read. For example, "Spell of Apple is A P P L E".
- Don't put multiple sentences into one line/one utterance. Separate each line per utterances.
- Make sure the sentence is mostly clean. In general, don't include too many non-standard words like numbers or abbreviations as they're usually hard to read. Some application may need to read many numbers or acronyms. In this case, 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", you have "by the way".
  - For lines with digits, instead of "911", you have "nine one one".
  - For lines with acronyms, instead of "ABC", you have "A B C".

With that, make sure your voice talent pronounces these words in the expected way. Keep your script and recordings match consistently during the training process.

> **NOTE**
>
> The scripts prepared for your voice talent need to follow the native reading conventions, such as 50% and $45, while the scripts used for training need to be normalized to make sure that the scripts match the audio content, such as *fifty percent* and *forty-five dollars*. Check the scripts used for training against the recordings of your voice talent, to make sure they match.

- 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 talent, you'll probably catch a few more mistakes.

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 the defects.

The script defects generally fall into the following categories:

CATEGORY	EXAMPLE
Have a meaningless content in a common way.	
Incomplete sentences.	- "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.	- 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:
	  - 45$ (should be $45)
	  - No space or excess space between word/punctuation

Duplication in similar format, one per each pattern is enough.	- "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 the commonly used foreign word is acceptable in our script.	
Emoji or any other uncommon symbols.	

Script format

You can write your script in Microsoft Word. The script is for use during the recording session, so you can set it up any way you find easy to work with. Create the text file that's required by 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 Word paragraph numbering feature to number the rows of the table automatically.
- A blank column where you'll write the take number or time code of each utterance to help you find it in the finished recording.
- The text of the utterance itself.

Custom Voice Script Session Date: 6/1/65 Voice Talent: A. Lincoln

	Time Code	Utterance
1		Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.
2		Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure.
3		We are met on a great battlefield of that war.
4		We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that that nation might live.

> **NOTE**
> Most studios record in short segments known as *takes*. Each take typically contains 10 to 24 utterances. Just noting the take number is sufficient to find an utterance later. If you're recording in a studio that prefers to make longer recordings, you'll want to note the time code instead. The studio will have a prominent time display.

Leave enough space after each row to write notes. Be sure that no utterance is split between pages. Number the pages, and print your script on one side of the paper.

Print three copies of the script: one for the talent, one for the engineer, and one for the director (you). Use a paper clip instead of staples: an experienced voice artist will separate the pages to avoid making noise as the pages are turned.

Legalities

Under copyright law, an actor's reading of copyrighted text might be a performance for which the author of the work should be compensated. This performance 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 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.

TEXT SOURCE	DESCRIPTION
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

PROPERTY	VALUE
Peak volume levels	-3 dB to -6 dB
SNR	> 35 dB
Silence	- 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	- 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 voice training, we'll down sample it to 24 KHz 16 bit PCM automatically.

Typical audio errors

For high-quality training results, avoiding audio errors is highly recommended. The errors of audio normally involve the following categories:

- Audio file name doesn't match the script ID.
- War file has an invalid format and cannot be read.
- Audio sampling rate is lower than 16 KHz. Also, it is recommended that wav file sampling rate should 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. That is, the waveform at its peak value is cut and thus not complete.

- The silence part isn't clean, such as ambient noise, mouth noise and echo.

  For example, below audio contains the environment noise between speeches.

Below sample contains noises 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 should not be longer than 200 ms or shorter than 100 ms.

Do it yourself

If you want to make the recording yourself, rather than going into a recording studio, here's a short primer. Thanks to the rise of home recording and podcasting, it's easier than ever to find good recording advice and resources online.

Your "recording booth" should be a small room with no noticeable echo or "room tone." It should be as quiet and soundproof as possible. Drapes on the walls can be used to reduce echo and neutralize or "deaden" the sound of the room.

Use a high-quality studio condenser microphone ("mic" for short) intended for recording voice. Sennheiser, AKG, and even newer Zoom mics can yield good results. You can buy a mic, or rent one from a local audio-visual rental firm. Look for one with a USB interface. This type of mic conveniently combines the microphone element, preamp, and analog-to-digital converter into one package, simplifying hookup.

You may also use an analog microphone. Many rental houses offer "vintage" microphones renowned for their voice character. Note that professional analog gear uses balanced XLR connectors, rather than the 1/4-inch plug that's used in consumer equipment. If you go analog, you'll also need a preamp and a computer audio interface with these connectors.

Install the microphone on a stand or boom, and install a pop filter in front of the microphone to eliminate noise from "plosive" consonants like "p" and "b." Some microphones come with a suspension mount that isolates them from vibrations in the stand, which is helpful.

The voice talent must stay at a consistent distance from the microphone. Use tape on the floor to mark where they should stand. If the talent prefers to sit, take special care to monitor mic distance and avoid chair noise.

Use a stand to hold the script. Avoid angling the stand so that it can reflect sound toward the microphone.

The person operating the recording equipment—the engineer—should be in a separate room from the talent, with some way to talk to the talent in the recording booth (a *talkback circuit).*

The recording should contain as little noise as possible, with a goal of an 80-dB signal-to-noise ratio or better.

Listen closely to a recording of silence in your "booth," figure out where any noise is coming from, and eliminate the cause. Common sources of noise are air vents, fluorescent light ballasts, traffic on nearby roads, and equipment fans (even notebook PCs might have fans). Microphones and cables can pick up electrical noise from nearby AC wiring, usually a hum or buzz. A buzz can also be caused by a *ground loop*, which is caused by having equipment plugged into more than one electrical circuit.

TIP

In some cases, you might be able to use an equalizer or a noise reduction software plug-in to help remove noise from your recordings, although it is always best to stop it at its source.

Set levels so that most of the available dynamic range of digital recording is used without overdriving. That means set the audio loud, but not so loud that it becomes distorted. An example of the waveform of a good recording is shown in the following image:

Here, most of the range (height) is used, but the highest peaks of the signal do not reach the top or bottom of the window. You can also see that the silence in the recording approximates a thin horizontal line, indicating a low noise floor. This recording has acceptable dynamic range and signal-to-noise ratio.

Record directly into the computer via a high-quality audio interface or a USB port, depending on the mic you're using. For analog, keep the audio chain simple: mic, preamp, audio interface, computer. You can license both Avid Pro Tools and Adobe Audition monthly at a reasonable cost. If your budget is extremely tight, try the free Audacity.

Record at 44.1 KHz 16 bit monophonic (CD quality) or better. Current state-of-the-art is 48 KHz 24 bit, if your equipment supports it. You will down-sample your audio to 24 KHz 16-bit before you submit it to Speech Studio. Still, it pays to have a high-quality original recording in the event edits are needed.

Ideally, have different people serve in the roles of director, engineer, and talent. Don't try to do it all yourself. In a pinch, one person can be both the director and the engineer.

Before the session

To avoid wasting studio time, run through the script with your voice talent before the recording session. While the voice talent becomes familiar with the text, they can clarify the pronunciation of any unfamiliar words.

> **NOTE**
>
> Most recording studios offer electronic display of scripts in the recording booth. In this case, type your run-through notes directly into the script's document. You'll still want a paper copy to take notes on during the session, though. Most engineers will want a hard copy, too. And you'll still want a third printed copy as a backup for the talent in case the computer is down.

Your voice talent might ask which word you want emphasized in an utterance (the "operative word"). Tell them that you want a natural reading with no particular emphasis. Emphasis can be added when speech is synthesized; it should not be a part of the original recording.

Direct the talent to pronounce words distinctly. Every word of the script should be pronounced as written. Sounds should not be omitted or slurred together, as is common in casual speech, *unless they have been written that way in the script*.

WRITTEN TEXT	UNWANTED CASUAL PRONUNCIATION
never going to give you up	never gonna give you up
there are four lights	there're four lights
how's the weather today	how's th' weather today
say hello to my little friend	say hello to my lil' friend

The talent should *not* add distinct pauses between words. The sentence should still flow naturally, even while sounding a little formal. This fine distinction might take practice to get right.

The recording session

Create a reference recording, or *match file,* of a typical utterance at the beginning of the session. Ask the talent to repeat this line every page or so. Each time, compare the new recording to the reference. This practice helps the talent remain consistent in volume, tempo, pitch, and intonation. Meanwhile, the engineer can use the match file as a reference for levels and overall consistency of sound.

The match file is especially important when you resume recording after a break or on another day. 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 really need to capture is the voice talent, so you can make a monophonic (single-channel) recording of just their lines. However, if you record in stereo, you can use the second channel to record the chatter in the control room to capture discussion of particular lines or takes. Remove this track from the version that's uploaded to 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.

Leave only about 0.2 second of silence at the beginning and end of each clip, except for the first. That file should start with a full five seconds of silence. Do not use an audio editor to "zero out" silent parts of the file. Including the "room tone" will help the algorithms compensate for any residual background noise.

Listen to each file carefully. At this stage, you can edit out small unwanted sounds that you missed during recording, like a slight lip smack before a line, but be careful not to remove any actual speech. If you can't fix a file, remove it from your dataset and note that you have done so.

Convert each file to 16 bits and a sample rate of 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. Create and use 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.

Create and use your voice model

Improve synthesis with the Audio Content Creation tool

11/23/2021 • 10 minutes to read • Edit Online

Audio Content Creation is an easy-to-use and powerful tool that lets you build highly natural audio content for a variety of scenarios, like audiobooks, news broadcasts, video narrations, and chat bots. With Audio Content Creation, you can fine-tune text-to-speech voices and design customized audio experiences in an efficient and low-cost way.

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.

You can have easy access to more than 150 pre-built voices across 60+ different languages, including the state-of-the-art neural TTS voices, and your custom voice if you have built one.

See the video tutorial for Audio Content Creation.

How to Get Started?

Audio Content Creation is a free tool, but you will pay for the Azure Speech service you consume. To work with the tool, you need to log in with an Azure account and create a speech resource. For each Azure account, you have monthly free speech quotas which include 500,000 characters for Neural TTS voices (per month), 5 million characters for standard and custom voices (per month), and 1 custom voice endpoint hosting service (per month). The monthly allotted amount is usually enough for a small content team of around 3-5 people. Here are the steps for 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 to have a Microsoft account and an Azure account. Follow these instructions to set up the account.

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 signing up for the Azure account, you need to create a Speech resource under your Azure account to access Speech services. View the instructions for how to create a Speech resource.

It takes a few moments to deploy your new Speech resource. Once the deployment is complete, you can start the Audio Content Creation journey.

> **NOTE**
>
> If you plan to use neural voices, make sure that you create your resource in a region that supports neural voices.

Step 3 - Log into the Audio Content Creation with your Azure account and Speech resource

1. After getting the Azure account and the Speech resource, you can log into Audio Content Creation by clicking **Get started**.
2. The home page lists all the products under Speech Studio. Click **Audio Content Creation** to start.
3. The **Welcome to Speech Studio** page will appear to you to set up the speech service. Select the Azure

subscription and the Speech resource you want to work on. Click **Use resource** to complete the settings. When you log into the Audio Content Creation tool for the Next time, we will link you directly to the audio work files under the current speech resource. You can check your Azure subscriptions details and status in Azure portal. If you do not have available speech resource and you are the owner or admin of an Azure subscription, you can also create a new Speech resource in Speech Studio by clicking **Create a new resource**. If you are a user role for a certain Azure subscription, you may not have the permission to create a new speech resource. Please contact your admin to get the speech resource access.

4. You can modify your Speech resource at any time with the **Settings** option, located in the top nav.
5. If you want to switch directory, please go the **Settings** or your profile to operate.

How to use the tool?

This diagram shows the steps it takes to fine-tune text-to-speech outputs. Use the links below to learn more about each step.

1. Choose the speech resource you want to work on.
2. Create an audio tuning file using plain text or SSML scripts. Type or upload your content in to Audio Content Creation.
3. Choose the voice and the language for your script content. Audio Content Creation includes all of the Microsoft text-to-speech voices. You can use standard, neural, or your own custom voice.

   > **NOTE**
   >
   > Gated access is available for Custom Neural Voices, which allow you to create high-definition voices similar to natural-sounding speech. For additional details, see Gating process.

4. Select the content you want to preview and click the **play** icon (a triangle) to preview the default synthesis output. Please note that if you make any changes on the text, you need to click the **Stop** icon and then click **play** icon again to re-generate the audio with changed scripts.
5. 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. Here is a video to show how to fine-tune speech output with Audio Content Creation.
6. 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

There are two ways to get your content into the Audio Content Creation tool.

Option 1:

1. Click **New** > **file** to create a new audio tuning file.
2. Type or paste your content into the editing window. The characters for each file is up to 20,000. If your script is longer than 20,000 characters, you can use Option 2 to automatically split your content into multiple files.
3. Don't forget to save.

Option 2:

1. Click **Upload** to import one or more text files. Both plain text and SSML are supported. If your script file is more than 20,000 characters, please split the file by paragraphs, by character or by regular expressions.
2. When you upload your text files, make sure that the file meets these requirements.

PROPERTY	VALUE / NOTES
File format	Plain text (.txt) SSML text (.txt) Zip files aren't supported
Encoding format	UTF-8
File name	Each file must have a unique name. Duplicates aren't supported.
Text length	Character limitation of the text file is 20,000. If your files exceed the limitation, please split the files with the instructions in the tool.
SSML restrictions	Each SSML file can only contain a single piece of SSML.

Plain text example

```
Welcome to use Audio Content Creation to customize audio output for your products.
```

SSML text example

```
<speak xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" version="1.0"
xml:lang="en-US">
    <voice name="Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)">
    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. Click **Export** to create an audio creation task. **Export to Audio Library** is recommended as it supports the long audio output and the full audio output experience. You can also download the audio to your local disk directly, but only the first 10 minutes are available.

2. Choose the output format for your tuned audio. A list of supported formats and sample rates is available below.
3. You can view the status of the task on the **Export task** tab. If the task fails, see the detailed information page for a full report.
4. When the task is complete, your audio is available for download on the **Audio Library** tab.
5. Click **Download**. Now you're ready to use your custom tuned audio in your apps or products.

Supported audio formats

FORMAT	16 KHZ SAMPLE RATE	24 KHZ SAMPLE RATE
wav	riff-16khz-16bit-mono-pcm	riff-24khz-16bit-mono-pcm
mp3	audio-16khz-128kbitrate-mono-mp3	audio-24khz-160kbitrate-mono-mp3

How to add/remove Audio Content Creation users?

If more than one user wants to use Audio Content Creation, you can grant user access to the Azure subscription and the speech resource. If you add a user to an Azure subscription, the user can access all the resources under the Azure subscription. But if you only add a user to a speech resource, the user will only have access to the speech resource, and cannot access other resources under this Azure subscription. A user with access to the speech resource can use Audio Content Creation.

The user need to prepare a Microsoft account. If the user do not have a Microsoft account, create one with just a few minutes. The user can use the existing email and link as a Microsoft account, or creat a new outlook email as Microsoft account.

Add users to a speech resource

Follow these steps to add a user to a speech resource so they can use Audio Content Creation.

1. Search for **Cognitive services** in the Azure portal, select the speech resource that you want to add users to.
2. Click **Access control (IAM)**. Select **Add** > **Add role assignment (Preview)** to open the Add role assignment pane.
3. On the **Role** tab, select the **Cognitive Service User** role. If you want to give the user ownership of this speech resource, you can select the **Owner** role.
4. On the **Members** tab, type in user's email address and select the user in the directory. The email address must be a **Microsoft account**, which is trusted by Azure active directory. Users can easily sign up a Microsoft account using a personal email address.
5. On the **Review + assign** tab, select **Review + assign** to assign the role.
6. The user will receive an email invitation. Accept the invitation by clicking **Accept invitation** > **Accept to join Azure** in the email. Then the user will be redirected to the Azure portal. The user does not need to take further action in the Azure portal. After a few moments, the user is assigned the role at the speech resource scope, and will have the access to this speech resource. If the user didn't receive the invitation email, you can search the user's account under "Role assignments" and go inside the user's profile. Find "Identity" -> "Invitation accepted", and click **(manage)** to resend the email invitation. You can also copy the invitation link to the users.
7. The user now visits or refreshes the Audio Content Creation product page, and sign in with the user's Microsoft account. Select **Audio Content Creation** block among all speech products. Choose the speech resource in the pop-up window or in the settings at the upper right of the page. If the user cannot

find available speech resource, check if you are in the right directory. To check the right directory, click the account profile in the upper right corner, and click **Switch** besides the "Current directory". If there are more than one directory available, it means you have access to multiple directories. Switch to different directories and go to settings to see if the right speech resource is available.

Users who are in the same speech resource will see each other's work in Audio Content Creation studio. If you want each individual user to have a unique and private workplace in Audio Content Creation, please 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. Click **Access control (IAM)**. Click the **Role assignments** tab to view all the role assignments for this speech resource.
3. Select the users you want to remove, click **Remove** > **Ok**.

Enable users to grant access

If you want one of the users to give access to other users, you need to give the user 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. See how to add users to a speech resource.

2. In the Azure portal, select the collapsed menu in the upper left. Click **Azure Active Directory**, and then Click **Users**.
3. Search the user's Microsoft account, and go to the user's detail page. Click **Assigned roles**.
4. Click **Add assignments** -> **Directory Readers**. If the button "Add assignments" is grayed out, it means that you do not have the access. Only the global administrator of this directory can add assignment to users.

See also

- Long Audio API

Next steps

Speech Studio

Guidelines for responsible deployment of synthetic voice technology

11/23/2021 • 3 minutes to read • Edit Online

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 will need to ensure that the parents or legal guardians are able to 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.

Obtain meaningful consent from voice talent

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 are 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 those with speech disorders

When working with individuals with speech disorders, to create or deploy synthetic voice technology, the following guidelines apply.

Provide guidelines to establish contracts

Provide guidelines for establishing contracts with individuals who use synthetic voice for assistance in speaking. The contract should consider specifying the parties who own the voice, duration of use, ownership transfer criteria, procedures for deleting the voice font, and how to prevent unauthorized access. Additionally, enable the 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 desire to make updates to their synthetic voice to reflect aging (for example, a child reaching puberty). Individuals may also have stylistic preferences that change over time, and may want to make changes to pitch, accent, or other voice characteristics.

See also

- Disclosure for Voice Talent
- How to Disclose
- Disclosure Design Patterns

Disclosure design guidelines

11/23/2021 • 3 minutes to read • Edit Online

Learn how to build and maintain trust with customers by being transparent about the synthetic nature of your voice experience.

What is disclosure?

Disclosure is a means of letting people know they're interacting with or listening to a voice that is synthetically generated.

Why is disclosure necessary?

The need to disclose the synthetic origins of a computer-generated voice is relatively new. In the past, computer-generated voices were obviously that—no one would ever mistake them for a real person. Every day, however, the realism of synthetic voices improves, and they become increasingly indistinguishable from human voices.

Goals

These are the principles to keep in mind when designing synthetic voice experiences:

Reinforce trust
Design with the intention to fail the Turing Test without degrading the experience. Let the users in on the fact that they're interacting with a synthetic voice while allowing them to engage seamlessly with the experience.

Adapt to context of use
Understand when, where, and how your users will interact with the synthetic voice to provide the right type of disclosure at the right time.

Set clear expectations
Allow users to easily discover and understand the capabilities of the agent. Offer opportunities to learn more about synthetic voice technology upon request.

Embrace failure
Use moments of failure to reinforce the capabilities of the agent.

How to use this guide

This guide helps you determine which disclosure patterns are best fit for your synthetic voice experience. We then offer examples of how and when to use them. Each of these patterns is designed to maximize transparency with users about synthetic speech while staying true to human-centered design.

Considering the vast body of design guidance on voice experiences, we focus here specifically on:

1. **Disclosure assessment**: A process to determine the type of disclosure recommended for your synthetic voice experience
2. **How to disclose**: Examples of disclosure patterns that can be applied to your synthetic voice experience
3. **When to disclose**: Optimal moments to disclose throughout the user journey

Disclosure assessment

Consider your users' expectations about an interaction and the context in which they will experience the voice. If the context makes it clear that a synthetic voice is "speaking," disclosure may be minimal, momentary, or even unnecessary. The main types of context that affect disclosure include persona type, scenario type, and level of exposure. It also helps to consider who might be listening.

Understand context

Use this worksheet to determine the context of your synthetic voice experience. You'll apply this in next step where you'll determine your disclosure level.

	CONTEXT OF USE	POTENTIAL RISKS & CHALLENGES
1. Persona type	**If any of the following apply, your persona fits under the 'Human-like Persona' category:** • 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:** • 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.
3. Level of exposure	**Your voice experience most likely fits under the 'High' category if:** • The user will hear or interact with the synthetic voice frequently or for a long duration of time	The importance of being transparent and building trust with users is even higher when establishing long-term relationships.

Determine disclosure level

Use the following diagram to determine whether your synthetic voice experience requires high or low disclosure based on your context of use.

See also

- Disclosure design patterns
- Disclosure for Voice Talent

- Guidelines for Responsible Deployment of Synthetic Voice Technology

Disclosure design patterns

11/23/2021 • 8 minutes to read • Edit Online

Now that you've determined the right level of disclosure for your synthetic voice experience, it's a good time to explore potential design patterns.

Overview

There's a spectrum of disclosure design patterns you can apply to your synthetic voice experience. If the outcome of your disclosure assessment was 'High Disclosure', we recommend **explicit disclosure**, which means communicating the origins of the synthetic voice outright. **Implicit disclosure** includes cues and interaction patterns that benefit voice experiences whether or not required disclosure levels are high or low.

EXPLICIT DISCLOSURE PATTERNS	IMPLICIT DISCLOSURE PATTERNS
Transparent Introduction Verbal Transparent Introduction Explicit Byline Customization and Calibration Parental Disclosure Providing opportunities to learn more about how the voice was made	Capability Disclosure Implicit Cues and Feedback Conversational Transparency

Use the following chart to refer directly to the patterns that apply to your synthetic voice. Some of the other conditions in this chart may also apply to your scenario:

IF YOUR SYNTHETIC VOICE EXPERIENCE...	RECOMMENDATIONS	DESIGN PATTERNS
Requires High Disclosure	Use at least one explicit pattern and implicit cues up front to helps users build associations.	Explicit Disclosure Implicit Disclosure
Requires Low Disclosure	Disclosure may be minimal or unnecessary, but could benefit from some implicit patterns.	Capability Disclosure Conversational Transparency
Has a high level of engagement	Build for the long term and offer multiple entry points to disclosure along the user journey. It is highly recommended to have an onboarding experience.	Transparent Introduction Customization and Calibration Capability Disclosure

IF YOUR SYNTHETIC VOICE EXPERIENCE...	RECOMMENDATIONS	DESIGN PATTERNS
Includes children as the primary intended audience	Target parents as the primary disclosure audience and ensure that they can effectively communicate disclosure to children.	Parental Disclosure Verbal Transparent Introduction Implicit Disclosure Conversational Transparency
Includes blind users or people with low vision as the primary intended audience	Be inclusive of all users and ensure that any form of visual disclosure has associated alternative text or sound effects. Adhere to accessibility standards for contrast ratio and display size. Use auditory cues to communicate disclosure.	Verbal Transparent Introduction Auditory Cues Haptic Cues Conversational Transparency Accessibility Standards
Is screen-less, device-less or uses voice as the primary or only mode of interaction	Use auditory cues to communicate disclosure.	Verbal Transparent Introduction Auditory Cues
Potentially includes multiple users/listeners (e.g., personal assistant in multiple household)	Be mindful of various user contexts and levels of understanding and offer multiple opportunities for disclosure in the user journey.	Transparent Introduction (Return User) Providing opportunities to learn more about how the voice was made Conversational Transparency

Explicit disclosure

If your synthetic voice experience requires high disclosure, it's best to use at least one of the following explicit patterns to clearly state the synthetic nature.

Transparent Introduction

Before the voice experience begins, introduce the digital assistant by being fully transparent about the origins of its voice and its capabilities. The optimal moment to use this pattern is when onboarding a new user or when introducing new features to a returning user. Implementing implicit cues during an introduction helps users form a mental model about the synthetic nature of the digital agent.

First-time user experience

The synthetic voice is introduced while onboarding a new user.

Recommendations

- Describe that the voice is artificial (e.g. "digital")
- Describe what the agent is capable of doing
- Explicitly state the voice's origins
- Offer an entry point to learn more about the synthetic voice

Returning user experience

If a user skips the onboarding experience, continue to offer entry points to the Transparent Introduction experience until the user triggers the voice for the first time.

Provide a consistent entry point to the synthetic voice experience. Allow the user to return to the onboarding experience when they trigger the voice for the first time at any point in the user journey.

Verbal Transparent Introduction

A spoken prompt stating the origins of the digital assistant's voice is explicit enough on its own to achieve disclosure. This pattern is best for high disclosure scenarios where voice is the only mode of interaction available.

Use a transparent introduction when there are moments in the user experience where you might already introduce or attribute a person's voice.

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.

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, theyhave 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.

User preferences allow users to customize and improve their experience.

Recommendations

- Allow users to customize the voice (e.g., select language and voice type)
- Provide users a way to teach the system to respond to his/her unique voice (e.g., voice calibration, custom commands)
- Optimize for user-generated or contextual interactions (e.g., reminders)

Persona Customization

Offer ways to customize the digital assistant's voice. If the voice is based on a celebrity or a widely recognizable person, consider using both visual and spoken introductions when users preview the voice.

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.

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.

More information can be offered in an external site help site.

Recommendations

- Simplify complex concepts and avoid using legalese and technical jargon
- Don't bury this content in privacy and terms of use statements
- Keep content concise and use imagery when available

Implicit disclosure

Consistency is the key to achieving disclosure implicitly throughout the user journey. Consistent use of visual and auditory cues across devices and modes of interaction can help build associations between implicit patterns and explicit disclosure.

Implicit Cues & Feedback

Anthropomorphism can manifest in different ways, from the actual visual representation of the agent, to the voice, sounds, patterns of light, bouncing shapes, or even the vibration of a device. When defining your persona, leverage implicit cues and feedback patterns rather than aim for a very human-like avatar. This is one way to minimize the need for more explicit disclosure.

These cues help anthropomorphize the agent without being too human-like. They can also become effective disclosure mechanisms on their own when used consistently over time.

Consider the different modes of interactions of your experience when incorporating the following types of cues:

VISUAL CUES	AUDITORY CUES	HAPTIC CUES
Avatar Responsive real-time cues (e.g., animations) Non-screen cues (e.g., lights and patterns on a device)	Sonicon (e.g., a brief distinctive sound, series of musical notes)	Vibration

Capability Disclosure

Disclosure can be achieved implicitly by setting accurate expectations for what the digital assistant is capable of. Provide sample commands so that users can learn how to interact with the digital assistant and offer contextual help to learn more about the synthetic voice during the early stages of the experience.

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.

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.

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 timea person interacts with thesyntheticvoice. 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

- Disclosure for Voice Talent
- Guidelines for Responsible Deployment of Synthetic Voice Technology
- How to Disclose

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

Text-to-speech REST API

11/23/2021 • 10 minutes to read • Edit Online

The Speech service allows you to convert text into synthesized speech and get a list of supported voices for a region using a set of REST APIs. Each available endpoint is associated with a region. A subscription key for the endpoint/region you plan to use is required.

The text-to-speech REST API supports neural and standard text-to-speech voices, each of which supports a specific language and dialect, identified by locale.

- For a complete list of voices, see language support.
- For information about regional availability, see regions.

> **IMPORTANT**
>
> Costs vary for standard, custom, and neural voices. For more information, see Pricing.

Before using this API, understand:

- The text-to-speech REST API requires an Authorization header. This means that you need to complete a token exchange to access the service. For more information, see Authentication.

> **TIP**
>
> See this article for Azure Government and Azure China endpoints.

Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each service:

SUPPORTED AUTHORIZATION HEADERS	SPEECH-TO-TEXT	TEXT-TO-SPEECH
Ocp-Apim-Subscription-Key	Yes	Yes
Authorization: Bearer	Yes	Yes

When using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your subscription key. For example:

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your subscription key for an access token that's valid for 10 minutes. In the next few sections you'll learn how to get a token, and use a token.

How to get an access token

To get an access token, you'll need to make a request to the `issueToken` endpoint using the `Ocp-Apim-Subscription-Key` and your subscription key.

The `issueToken` endpoint has this format:

```
https://<REGION_IDENTIFIER>.api.cognitive.microsoft.com/sts/v1.0/issueToken
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

Use these samples to create your access token request.

HTTP sample

This example is a simple HTTP request to get a token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.api.cognitive.microsoft.com
Content-type: application/x-www-form-urlencoded
Content-Length: 0
```

The body of the response contains the access token in JSON Web Token (JWT) format.

PowerShell sample

This example is a simple PowerShell script to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
$FetchTokenHeader = @{
  'Content-type'='application/x-www-form-urlencoded';
  'Content-Length'= '0';
  'Ocp-Apim-Subscription-Key' = 'YOUR_SUBSCRIPTION_KEY'
}

$OAuthToken = Invoke-RestMethod -Method POST -Uri
https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
 -Headers $FetchTokenHeader

# show the token received
$OAuthToken
```

cURL sample

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). This cURL command illustrates how to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
curl -v -X POST \
 "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
 -H "Content-type: application/x-www-form-urlencoded" \
 -H "Content-Length: 0" \
 -H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

C# sample

This C# class illustrates how to get an access token. Pass your Speech Service subscription key when you instantiate the class. If your subscription isn't in the West US region, change the value of `FetchTokenUri` to match the region for your subscription.

```
public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
    private string subscriptionKey;
    private string token;

    public Authentication(string subscriptionKey)
    {
        this.subscriptionKey = subscriptionKey;
        this.token = FetchTokenAsync(FetchTokenUri, subscriptionKey).Result;
    }

    public string GetAccessToken()
    {
        return this.token;
    }

    private async Task<string> FetchTokenAsync(string fetchUri, string subscriptionKey)
    {
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
            UriBuilder uriBuilder = new UriBuilder(fetchUri);

            var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
            Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
            return await result.Content.ReadAsStringAsync();
        }
    }
}
```

Python sample

```
# Request module must be installed.
# Run pip install requests if necessary.
import requests

subscription_key = 'REPLACE_WITH_YOUR_KEY'


def get_token(subscription_key):
    fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key
    }
    response = requests.post(fetch_token_url, headers=headers)
    access_token = str(response.text)
    print(access_token)
```

How to use an access token

The access token should be sent to the service as the `Authorization: Bearer <TOKEN>` header. Each access token is valid for 10 minutes. You can get a new token at any time, however, to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the Speech-to-text REST API for short audio:

```
POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.stt.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

// Message body here...
```

Get a list of voices

The `voices/list` endpoint allows you to get a full list of voices for a specific region/endpoint.

Regions and endpoints

REGION	ENDPOINT
Australia East	`https://australiaeast.tts.speech.microsoft.com/cognitiveservices/voices`
Brazil South	`https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/voices/l`
Canada Central	`https://canadacentral.tts.speech.microsoft.com/cognitiveservices/voices`
Central US	`https://centralus.tts.speech.microsoft.com/cognitiveservices/voices/lis`
East Asia	`https://eastasia.tts.speech.microsoft.com/cognitiveservices/voices/list`
East US	`https://eastus.tts.speech.microsoft.com/cognitiveservices/voices/list`
East US 2	`https://eastus2.tts.speech.microsoft.com/cognitiveservices/voices/list`
France Central	`https://francecentral.tts.speech.microsoft.com/cognitiveservices/voices`
India Central	`https://centralindia.tts.speech.microsoft.com/cognitiveservices/voices/`
Japan East	`https://japaneast.tts.speech.microsoft.com/cognitiveservices/voices/lis`
Korea Central	`https://koreacentral.tts.speech.microsoft.com/cognitiveservices/voices/`

REGION	ENDPOINT
North Central US	`https://northcentralus.tts.speech.microsoft.com/cognitiveservices/voice`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/voices/l`
South Africa North	`https://southafricanorth.tts.speech.microsoft.com/cognitiveservices/voi`
South Central US	`https://southcentralus.tts.speech.microsoft.com/cognitiveservices/voice`
Southeast Asia	`https://southeastasia.tts.speech.microsoft.com/cognitiveservices/voices`
UK South	`https://uksouth.tts.speech.microsoft.com/cognitiveservices/voices/list`
West Central US	`https://westcentralus.tts.speech.microsoft.com/cognitiveservices/voices`
West Europe	`https://westeurope.tts.speech.microsoft.com/cognitiveservices/voices/li`
West US	`https://westus.tts.speech.microsoft.com/cognitiveservices/voices/list`
West US 2	`https://westus2.tts.speech.microsoft.com/cognitiveservices/voices/list`

> **TIP**
> Voices in preview are only available in these 3 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 / OPTIONAL
`Ocp-Apim-Subscription-Key`	Your Speech service subscription key.	Either this header or `Authorization` is required.
`Authorization`	An authorization token preceded by the word `Bearer`. For more information, see Authentication.	Either this header or `Ocp-Apim-Subscription-Key` is required.

Request body

A body isn't required for `GET` requests to this endpoint.

Sample request

This request only requires an authorization header.

```
GET /cognitiveservices/voices/list HTTP/1.1

Host: westus.tts.speech.microsoft.com
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
```

Sample response

This response has been truncated to illustrate the structure of a response.

> **NOTE**
> Voice availability varies by region/endpoint.

```
[

    {
```

```
    "Name": "Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)",
    "DisplayName": "Christopher",
    "LocalName": "Christopher",
    "ShortName": "en-US-ChristopherNeural",
    "Gender": "Male",
    "Locale": "en-US",
    "StyleList": [
      "chat",
      "customerservice",
      "newscast-casual",
      "newscast-formal",
      "cheerful",
      "empathetic"
    ],
    "SampleRateHertz": "24000",
    "VoiceType": "Neural",
    "Status": "GA"
  },

    ...

     {
    "Name": "Microsoft Server Speech Text to Speech Voice (en-US, JennyMultilingualNeural)",
    "ShortName": "en-US-JennyMultilingualNeural",
    "DisplayName": "Jenny Multilingual",
    "LocalName": "Jenny Multilingual",
    "Gender": "Female",
    "Locale": "en-US",
    "SampleRateHertz": "24000",
    "VoiceType": "Neural",
    "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"
      ],
    "Status": "Preview"
    },

  ...

    {
    "Name": "Microsoft Server Speech Text to Speech Voice (ga-IE, OrlaNeural)",
    "DisplayName": "Orla",
    "LocalName": "Orla",
    "ShortName": "ga-IE-OrlaNeural",
    "Gender": "Female",
    "Locale": "ga-IE",
    "SampleRateHertz": "24000",
    "VoiceType": "Neural",
    "Status": "Preview"
  },

  ...

   {
    "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",
    "StyleList": [
      "Calm",
      "Fearful",
      "Cheerful",
      "Disgruntled",
      "Serious",
      "Angry",
      "Sad",
```

```
      "Depressed",
      "Embarrassed"
    ],
    "SampleRateHertz": "24000",
    "VoiceType": "Neural",
    "Status": "Preview"
  },

    ...

   {
    "Name": "Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)",
    "DisplayName": "Hoda",
    "LocalName": "هدى",
    "ShortName": "ar-EG-Hoda",
    "Gender": "Female",
    "Locale": "ar-EG",
    "SampleRateHertz": "16000",
    "VoiceType": "Standard",
    "Status": "GA"
  },

...

]
```

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
200	OK	The request was successful.
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
429	Too Many Requests	You have exceeded the quota or rate of requests allowed for your subscription.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

Convert text-to-speech

The `v1` endpoint allows you to convert text-to-speech using Speech Synthesis Markup Language (SSML).

Regions and endpoints

These regions are supported for text-to-speech using the REST API. Make sure that you select the endpoint that matches your subscription region.

Neural and standard voices

Use this table to determine **availability of neural and standard voices** by region/endpoint:

REGION	ENDPOINT
Australia East	`https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1`
Brazil South	`https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/v1`

REGION	ENDPOINT
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`

> **TIP**
>
> Voices in preview are only available in these 3 regions: East US, West Europe and Southeast Asia.

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	`https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
Brazil South	`https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
Canada Central	`https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`

REGION	ENDPOINT
Central US	`https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
East Asia	`https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
East US	`https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
East US 2	`https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
France Central	`https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
India Central	`https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
Japan East	`https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
Japan West	`https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
Korea Central	`https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
North Central US	`https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
North Europe	`https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
South Central US	`https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
Southeast Asia	`https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
UK South	`https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
West Europe	`https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
West US	`https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`
West US 2	`https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}`

Custom Neural Voice

The following table details regional support for Custom Neural Voice features.

FEATURE	SUPPORTED REGIONS
Voice model hosting	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Realtime Characters	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East

FEATURE	SUPPORTED REGIONS
Long Audio Characters	East US, West Europe, Southeast Asia, UK South, India Central
Custom Neural Training	East US, Southeast Asia, UK South

Long audio API

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.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 / OPTIONAL
`Authorization`	An authorization token preceded by the word `Bearer`. For more information, see Authentication.	Required
`Content-Type`	Specifies the content type for the provided text. Accepted value: `application/ssml+xml`.	Required
`X-Microsoft-OutputFormat`	Specifies the audio output format. For a complete list of accepted values, see audio outputs.	Required
`User-Agent`	The application name. The value provided must be less than 255 characters.	Required

Audio outputs

This is a list of supported audio formats that are sent in each request as the `X-Microsoft-OutputFormat` header. Each incorporates a bitrate and encoding type. The Speech service supports 24 kHz, 16 kHz, and 8 kHz audio outputs.

```
raw-16khz-16bit-mono-pcm            riff-16khz-16bit-mono-pcm
raw-24khz-16bit-mono-pcm            riff-24khz-16bit-mono-pcm
raw-48khz-16bit-mono-pcm            riff-48khz-16bit-mono-pcm
raw-8khz-8bit-mono-mulaw            riff-8khz-8bit-mono-mulaw
raw-8khz-8bit-mono-alaw             riff-8khz-8bit-mono-alaw
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
audio-48khz-96kbitrate-mono-mp3     audio-48khz-192kbitrate-mono-mp3
raw-16khz-16bit-mono-truesilk       raw-24khz-16bit-mono-truesilk
webm-16khz-16bit-mono-opus          webm-24khz-16bit-mono-opus
ogg-16khz-16bit-mono-opus           ogg-24khz-16bit-mono-opus
ogg-48khz-16bit-mono-opus
```

> **NOTE**
> If your selected voice and output format have different bit rates, the audio is resampled as necessary. ogg-24khz-16bit-mono-opus can be decoded with opus codec

Request body

The body of each `POST` request is sent as Speech Synthesis Markup Language (SSML). SSML allows you to choose the voice and language of the synthesized speech returned by the text-to-speech service. For a complete list of supported voices, see language support.

> **NOTE**
> If using a custom voice, the body of a request can be sent as plain text (ASCII or UTF-8).

Sample request

This HTTP request uses SSML to specify the voice and language. If the body length is long, and the resulting audio exceeds 10 minutes - it is truncated to 10 minutes. In other words, the audio length cannot exceed 10 minutes.

```
POST /cognitiveservices/v1 HTTP/1.1

X-Microsoft-OutputFormat: raw-24khz-16bit-mono-pcm
Content-Type: application/ssml+xml
Host: westus.tts.speech.microsoft.com
Content-Length: 225
Authorization: Bearer [Base64 access_token]

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Male'
    name='en-US-ChristopherNeural'>
        Microsoft Speech Service Text-to-Speech API
</voice></speak>
```

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
200	OK	The request was successful; the response body is an audio file.
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
415	Unsupported Media Type	It's possible that the wrong `Content-Type` was provided. `Content-Type` should be set to `application/ssml+xml`.
429	Too Many Requests	You have exceeded the quota or rate of requests allowed for your subscription.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

If the HTTP status is `200 OK`, the body of the response contains an audio file in the requested format. This file

can be played as it's transferred, saved to a buffer, or saved to a file.

Next steps

- Create a free Azure account
- Asynchronous synthesis for long-form audio
- Get started with Custom Neural Voice

Language and voice support for the Speech service

11/23/2021 • 22 minutes to read • Edit Online

Language support varies by Speech service functionality. The following tables summarize language support for Speech-to-text, Text-to-speech, Speech translation and Speaker Recognition service offerings.

Speech-to-text

Both the Microsoft Speech SDK and the REST API support the following languages (locales).

To improve accuracy, customization is offered for a subset of the languages through uploading **Audio + Human-labeled Transcripts** or **Related Text: Sentences**. Support for customization of the acoustic model with **Audio + Human-labeled Transcripts** is limited to the specific base models listed below. Other base models and languages will only use the text of the transcripts to train custom models just like with **Related Text: Sentences**. To learn more about customization, see Get started with Custom Speech.

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Algeria)	`ar-DZ`	Text		
Arabic (Bahrain), modern standard	`ar-BH`	Text		
Arabic (Egypt)	`ar-EG`	Text	Yes	
Arabic (Iraq)	`ar-IQ`	Text		
Arabic (Israel)	`ar-IL`	Text		
Arabic (Jordan)	`ar-JO`	Text		
Arabic (Kuwait)	`ar-KW`	Text		
Arabic (Lebanon)	`ar-LB`	Text		
Arabic (Libya)	`ar-LY`	Text		
Arabic (Morocco)	`ar-MA`	Text		
Arabic (Oman)	`ar-OM`	Text		
Arabic (Qatar)	`ar-QA`	Text		
Arabic (Saudi Arabia)	`ar-SA`	Text		
Arabic (Palestinian Authority)	`ar-PS`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Syria)	`ar-SY`	Text		
Arabic (Tunisia)	`ar-TN`	Text		
Arabic (United Arab Emirates)	`ar-AE`	Text		
Arabic (Yemen)	`ar-YE`	Text		
Bulgarian (Bulgaria)	`bg-BG`	Text		
Catalan (Spain)	`ca-ES`	Text Pronunciation	Yes	
Chinese (Cantonese, Traditional)	`zh-HK`	Audio (20201015) Text	Yes	
Chinese (Mandarin, Simplified)	`zh-CN`	Audio (20200910) Text	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Audio (20190701, 20201015) Text	Yes	
Croatian (Croatia)	`hr-HR`	Text Pronunciation		
Czech (Czech)	`cs-CZ`	Text Pronunciation		
Danish (Denmark)	`da-DK`	Text Pronunciation	Yes	
Dutch (Netherlands)	`nl-NL`	Audio (20201015) Text Pronunciation	Yes	
English (Australia)	`en-AU`	Audio (20201019) Text Pronunciation	Yes	
English (Canada)	`en-CA`	Audio (20201019) Text Pronunciation	Yes	
English (Ghana)	`en-GH`	Text Pronunciation		
English (Hong Kong)	`en-HK`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
English (India)	`en-IN`	Audio (20200923) Text Pronunciation		
English (Ireland)	`en-IE`	Text Pronunciation		
English (Kenya)	`en-KE`	Text Pronunciation		
English (New Zealand)	`en-NZ`	Audio (20201019) Text Pronunciation		
English (Nigeria)	`en-NG`	Text Pronunciation		
English (Philippines)	`en-PH`	Text Pronunciation		
English (Singapore)	`en-SG`	Text Pronunciation		
English (South Africa)	`en-ZA`	Text Pronunciation		
English (Tanzania)	`en-TZ`	Text Pronunciation		
English (United Kingdom)	`en-GB`	Audio (20201019) Text Pronunciation	Yes	Yes
English (United States)	`en-US`	Audio (20201019, 20210223) Text Pronunciation	Yes	Yes
Estonian(Estonia)	`et-EE`	Text Pronunciation		
Filipino (Philippines)	`fil-PH`	Text Pronunciation		
Finnish (Finland)	`fi-FI`	Text Pronunciation	Yes	
French (Canada)	`fr-CA`	Audio (20201015) Text Pronunciation	Yes	
French (France)	`fr-FR`	Audio (20201015) Text Pronunciation	Yes	

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
French (Switzerland)	`fr-CH`	Text Pronunciation		
German (Austria)	`de-AT`	Text Pronunciation		
German (Switzerland)	`de-CH`	Text Pronunciation		
German (Germany)	`de-DE`	Audio (20190701, 20200619, 20201127) Text Pronunciation	Yes	
Greek (Greece)	`el-GR`	Text	Yes	
Gujarati (Indian)	`gu-IN`	Text		
Hebrew (Israel)	`he-IL`	Text		
Hindi (India)	`hi-IN`	Audio (20200701) Text	Yes	
Hungarian (Hungary)	`hu-HU`	Text Pronunciation		
Indonesian (Indonesia)	`id-ID`	Text Pronunciation		
Irish (Ireland)	`ga-IE`	Text Pronunciation		
Italian (Italy)	`it-IT`	Audio (20201016) Text Pronunciation	Yes	
Japanese (Japan)	`ja-JP`	Text	Yes	
Kannada (India)	`kn-IN`	Text		
Korean (Korea)	`ko-KR`	Audio (20201015) Text	Yes	
Latvian (Latvia)	`lv-LV`	Text Pronunciation		
Lithuanian (Lithuania)	`lt-LT`	Text Pronunciation		
Malay (Malaysia)	`ms-MY`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Maltese (Malta)	`mt-MT`	Text		
Marathi (India)	`mr-IN`	Text		
Norwegian (Bokmål, Norway)	`nb-NO`	Text	Yes	
Persian (Iran)	`fa-IR`	Text		
Polish (Poland)	`pl-PL`	Text Pronunciation	Yes	
Portuguese (Brazil)	`pt-BR`	Audio (20190620, 20201015) Text Pronunciation	Yes	
Portuguese (Portugal)	`pt-PT`	Text Pronunciation	Yes	
Romanian (Romania)	`ro-RO`	Text Pronunciation	Yes	
Russian (Russia)	`ru-RU`	Audio (20200907) Text	Yes	
Slovak (Slovakia)	`sk-SK`	Text Pronunciation		
Slovenian (Slovenia)	`sl-SI`	Text Pronunciation		
Spanish (Argentina)	`es-AR`	Text Pronunciation		
Spanish (Bolivia)	`es-BO`	Text Pronunciation		
Spanish (Chile)	`es-CL`	Text Pronunciation		
Spanish (Colombia)	`es-CO`	Text Pronunciation		
Spanish (Costa Rica)	`es-CR`	Text Pronunciation		
Spanish (Cuba)	`es-CU`	Text Pronunciation		
Spanish (Dominican Republic)	`es-DO`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Spanish (Ecuador)	`es-EC`	Text Pronunciation		
Spanish (El Salvador)	`es-SV`	Text Pronunciation		
Spanish (Equatorial Guinea)	`es-GQ`	Text		
Spanish (Guatemala)	`es-GT`	Text Pronunciation		
Spanish (Honduras)	`es-HN`	Text Pronunciation		
Spanish (Mexico)	`es-MX`	Audio (20200907) Text Pronunciation	Yes	
Spanish (Nicaragua)	`es-NI`	Text Pronunciation		
Spanish (Panama)	`es-PA`	Text Pronunciation		
Spanish (Paraguay)	`es-PY`	Text Pronunciation		
Spanish (Peru)	`es-PE`	Text Pronunciation		
Spanish (Puerto Rico)	`es-PR`	Text Pronunciation		
Spanish (Spain)	`es-ES`	Audio (20201015) Text Pronunciation	Yes	
Spanish (Uruguay)	`es-UY`	Text Pronunciation		
Spanish (USA)	`es-US`	Text Pronunciation		
Spanish (Venezuela)	`es-VE`	Text Pronunciation		
Swahili (Kenya)	`sw-KE`	Text		
Swedish (Sweden)	`sv-SE`	Text Pronunciation	Yes	
Tamil (India)	`ta-IN`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Telugu (India)	`te-IN`	Text		
Thai (Thailand)	`th-TH`	Text	Yes	
Turkish (Turkey)	`tr-TR`	Text		
Vietnamese (Vietnam)	`vi-VN`	Text		

Text-to-speech

Both the Microsoft Speech SDK and REST APIs support these voices, each of which supports a specific language and dialect, identified by locale. You can also get a full list of languages and voices supported for each specific region/endpoint through the voices list API.

> **IMPORTANT**
> Pricing varies for standard, custom and neural voices. Please visit the Pricing page for additional information.

Neural voices

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

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

> **NOTE**
> Neural voices are created from samples that use a 24 khz sample rate. All voices can upsample or downsample to other sample rates when synthesizing.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Afrikaans (South Africa)	`af-ZA`	Female	`af-ZA-AdriNeural` New	General
Afrikaans (South Africa)	`af-ZA`	Male	`af-ZA-WillemNeural` New	General
Amharic (Ethiopia)	`am-ET`	Female	`am-ET-MekdesNeural` New	General
Amharic (Ethiopia)	`am-ET`	Male	`am-ET-AmehaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Algeria)	`ar-DZ`	Female	`ar-DZ-AminaNeural` New	General
Arabic (Algeria)	`ar-DZ`	Male	`ar-DZ-IsmaelNeural` New	General
Arabic (Bahrain)	`ar-BH`	Female	`ar-BH-LailaNeural` New	General
Arabic (Bahrain)	`ar-BH`	Male	`ar-BH-AliNeural` New	General
Arabic (Egypt)	`ar-EG`	Female	`ar-EG-SalmaNeural`	General
Arabic (Egypt)	`ar-EG`	Male	`ar-EG-ShakirNeural`	General
Arabic (Iraq)	`ar-IQ`	Female	`ar-IQ-RanaNeural` New	General
Arabic (Iraq)	`ar-IQ`	Male	`ar-IQ-BasselNeural` New	General
Arabic (Jordan)	`ar-JO`	Female	`ar-JO-Sana Neural` New	General
Arabic (Jordan)	`ar-JO`	Male	`ar-JO-Taim Neural` New	General
Arabic (Kuwait)	`ar-KW`	Female	`ar-KW-NouraNeural` New	General
Arabic (Kuwait)	`ar-KW`	Male	`ar-KW-FahedNeural` New	General
Arabic (Libya)	`ar-LY`	Female	`ar-LY-ImanNeural` New	General
Arabic (Libya)	`ar-LY`	Male	`ar-LY-OmarNeural` New	General
Arabic (Morocco)	`ar-MA`	Female	`ar-MA-MounaNeural` New	General
Arabic (Morocco)	`ar-MA`	Male	`ar-MA-JamalNeural` New	General
Arabic (Qatar)	`ar-QA`	Female	`ar-QA-AmalNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Qatar)	`ar-QA`	Male	`ar-QA-MoazNeural` New	General
Arabic (Saudi Arabia)	`ar-SA`	Female	`ar-SA-ZariyahNeural`	General
Arabic (Saudi Arabia)	`ar-SA`	Male	`ar-SA-HamedNeural`	General
Arabic (Syria)	`ar-SY`	Female	`ar-SY-AmanyNeural` New	General
Arabic (Syria)	`ar-SY`	Male	`ar-SY-LaithNeural` New	General
Arabic (Tunisia)	`ar-TN`	Female	`ar-TN-ReemNeural` New	General
Arabic (Tunisia)	`ar-TN`	Male	`ar-TN-HediNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Female	`ar-AE-FatimaNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Male	`ar-AE-HamdanNeural` New	General
Arabic (Yemen)	`ar-YE`	Female	`ar-YE-MaryamNeural` New	General
Arabic (Yemen)	`ar-YE`	Male	`ar-YE-SalehNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Female	`bn-BD-NabanitaNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Male	`bn-BD-PradeepNeural` New	General
Bulgarian (Bulgaria)	`bg-BG`	Female	`bg-BG-KalinaNeural`	General
Bulgarian (Bulgaria)	`bg-BG`	Male	`bg-BG-BorislavNeural`	General
Burmese (Myanmar)	`my-MM`	Female	`my-MM-NilarNeural` New	General
Burmese (Myanmar)	`my-MM`	Male	`my-MM-ThihaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Catalan (Spain)	`ca-ES`	Female	`ca-ES-AlbaNeural`	General
Catalan (Spain)	`ca-ES`	Female	`ca-ES-JoanaNeural`	General
Catalan (Spain)	`ca-ES`	Male	`ca-ES-EnricNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuGaaiNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuMaanNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Male	`zh-HK-WanLungNeural`	General
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaohanNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaomoNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoruiNeural`	Senior voice, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxiaoNeural`	General, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxuanNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyouNeural`	Child voice, optimized for story narrating
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunxiNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyangNeural`	Optimized for news reading, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyeNeural`	Optimized for story narrating
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoChenNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoYuNeural`	General
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-YunJheNeural`	General
Croatian (Croatia)	`hr-HR`	Female	`hr-HR-GabrijelaNeural`	General
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-SreckoNeural`	General
Czech (Czech)	`cs-CZ`	Female	`cs-CZ-VlastaNeural`	General
Czech (Czech)	`cs-CZ`	Male	`cs-CZ-AntoninNeural`	General
Danish (Denmark)	`da-DK`	Female	`da-DK-ChristelNeural`	General
Danish (Denmark)	`da-DK`	Male	`da-DK-JeppeNeural`	General
Dutch (Belgium)	`nl-BE`	Female	`nl-BE-DenaNeural`	General
Dutch (Belgium)	`nl-BE`	Male	`nl-BE-ArnaudNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-ColetteNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-FennaNeural`	General
Dutch (Netherlands)	`nl-NL`	Male	`nl-NL-MaartenNeural`	General
English (Australia)	`en-AU`	Female	`en-AU-NatashaNeural`	General
English (Australia)	`en-AU`	Male	`en-AU-WilliamNeural`	General
English (Canada)	`en-CA`	Female	`en-CA-ClaraNeural`	General
English (Canada)	`en-CA`	Male	`en-CA-LiamNeural`	General
English (Hongkong)	`en-HK`	Female	`en-HK-YanNeural`	General
English (Hongkong)	`en-HK`	Male	`en-HK-SamNeural`	General
English (India)	`en-IN`	Female	`en-IN-NeerjaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (India)	`en-IN`	Male	`en-IN-PrabhatNeural`	General
English (Ireland)	`en-IE`	Female	`en-IE-EmilyNeural`	General
English (Ireland)	`en-IE`	Male	`en-IE-ConnorNeural`	General
English (Kenya)	`en-KE`	Female	`en-KE-AsiliaNeural` New	General
English (Kenya)	`en-KE`	Male	`en-KE-ChilembaNeural` New	General
English (New Zealand)	`en-NZ`	Female	`en-NZ-MollyNeural`	General
English (New Zealand)	`en-NZ`	Male	`en-NZ-MitchellNeural`	General
English (Nigeria)	`en-NG`	Female	`en-NG-EzinneNeural` New	General
English (Nigeria)	`en-NG`	Male	`en-NG-AbeoNeural` New	General
English (Philippines)	`en-PH`	Female	`en-PH-RosaNeural`	General
English (Philippines)	`en-PH`	Male	`en-PH-JamesNeural`	General
English (Singapore)	`en-SG`	Female	`en-SG-LunaNeural`	General
English (Singapore)	`en-SG`	Male	`en-SG-WayneNeural`	General
English (South Africa)	`en-ZA`	Female	`en-ZA-LeahNeural`	General
English (South Africa)	`en-ZA`	Male	`en-ZA-LukeNeural`	General
English (Tanzania)	`en-TZ`	Female	`en-TZ-ImaniNeural` New	General
English (Tanzania)	`en-TZ`	Male	`en-TZ-ElimuNeural` New	General
English (United Kingdom)	`en-GB`	Female	`en-GB-LibbyNeural`	General
English (United Kingdom)	`en-GB`	Female	`en-GB-SoniaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United Kingdom)	`en-GB`	Female	`en-GB-MiaNeural` Retired on 30 October 2021, see below	General
English (United Kingdom)	`en-GB`	Male	`en-GB-RyanNeural`	General
English (United States)	`en-US`	Female	`en-US-AmberNeural`	General
English (United States)	`en-US`	Female	`en-US-AriaNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-AshleyNeural`	General
English (United States)	`en-US`	Female	`en-US-CoraNeural`	General
English (United States)	`en-US`	Female	`en-US-ElizabethNeural`	General
English (United States)	`en-US`	Female	`en-US-JennyNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-MichelleNeural`	General
English (United States)	`en-US`	Female	`en-US-MonicaNeural`	General
English (United States)	`en-US`	Female	`en-US-SaraNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Kid	`en-US-AnaNeural`	General
English (United States)	`en-US`	Male	`en-US-BrandonNeural`	General
English (United States)	`en-US`	Male	`en-US-ChristopherNeural`	General
English (United States)	`en-US`	Male	`en-US-EricNeural`	General
English (United States)	`en-US`	Male	`en-US-GuyNeural`	General, multiple voice styles available using SSML

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Male	`en-US-JacobNeural`	General
Estonian (Estonia)	`et-EE`	Female	`et-EE-AnuNeural`	General
Estonian (Estonia)	`et-EE`	Male	`et-EE-KertNeural`	General
Filipino (Philippines)	`fil-PH`	Female	`fil-PH-BlessicaNeural` New	General
Filipino (Philippines)	`fil-PH`	Male	`fil-PH-AngeloNeural` New	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-NooraNeural`	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-SelmaNeural`	General
Finnish (Finland)	`fi-FI`	Male	`fi-FI-HarriNeural`	General
French (Belgium)	`fr-BE`	Female	`fr-BE-CharlineNeural`	General
French (Belgium)	`fr-BE`	Male	`fr-BE-GerardNeural`	General
French (Canada)	`fr-CA`	Female	`fr-CA-SylvieNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-AntoineNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-JeanNeural`	General
French (France)	`fr-FR`	Female	`fr-FR-DeniseNeural`	General
French (France)	`fr-FR`	Male	`fr-FR-HenriNeural`	General
French (Switzerland)	`fr-CH`	Female	`fr-CH-ArianeNeural`	General
French (Switzerland)	`fr-CH`	Male	`fr-CH-FabriceNeural`	General
Galician (Spain)	`gl-ES`	Female	`gl-ES-SabelaNeural` New	General
Galician (Spain)	`gl-ES`	Male	`gl-ES-RoiNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
German (Austria)	`de-AT`	Female	`de-AT-IngridNeural`	General
German (Austria)	`de-AT`	Male	`de-AT-JonasNeural`	General
German (Germany)	`de-DE`	Female	`de-DE-KatjaNeural`	General
German (Germany)	`de-DE`	Male	`de-DE-ConradNeural`	General
German (Switzerland)	`de-CH`	Female	`de-CH-LeniNeural`	General
German (Switzerland)	`de-CH`	Male	`de-CH-JanNeural`	General
Greek (Greece)	`el-GR`	Female	`el-GR-AthinaNeural`	General
Greek (Greece)	`el-GR`	Male	`el-GR-NestorasNeural`	General
Gujarati (India)	`gu-IN`	Female	`gu-IN-DhwaniNeural`	General
Gujarati (India)	`gu-IN`	Male	`gu-IN-NiranjanNeural`	General
Hebrew (Israel)	`he-IL`	Female	`he-IL-HilaNeural`	General
Hebrew (Israel)	`he-IL`	Male	`he-IL-AvriNeural`	General
Hindi (India)	`hi-IN`	Female	`hi-IN-SwaraNeural`	General
Hindi (India)	`hi-IN`	Male	`hi-IN-MadhurNeural`	General
Hungarian (Hungary)	`hu-HU`	Female	`hu-HU-NoemiNeural`	General
Hungarian (Hungary)	`hu-HU`	Male	`hu-HU-TamasNeural`	General
Indonesian (Indonesia)	`id-ID`	Female	`id-ID-GadisNeural`	General
Indonesian (Indonesia)	`id-ID`	Male	`id-ID-ArdiNeural`	General
Irish (Ireland)	`ga-IE`	Female	`ga-IE-OrlaNeural`	General
Irish (Ireland)	`ga-IE`	Male	`ga-IE-ColmNeural`	General
Italian (Italy)	`it-IT`	Female	`it-IT-ElsaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Italian (Italy)	`it-IT`	Female	`it-IT-IsabellaNeural`	General
Italian (Italy)	`it-IT`	Male	`it-IT-DiegoNeural`	General
Japanese (Japan)	`ja-JP`	Female	`ja-JP-NanamiNeural`	General
Japanese (Japan)	`ja-JP`	Male	`ja-JP-KeitaNeural`	General
Javanese (Indonesia)	`jv-ID`	Female	`jv-ID-SitiNeural` New	General
Javanese (Indonesia)	`jv-ID`	Male	`jv-ID-DimasNeural` New	General
Khmer (Cambodia)	`km-KH`	Female	`km-KH-SreymomNeural` New	General
Khmer (Cambodia)	`km-KH`	Male	`km-KH-PisethNeural` New	General
Korean (Korea)	`ko-KR`	Female	`ko-KR-SunHiNeural`	General
Korean (Korea)	`ko-KR`	Male	`ko-KR-InJoonNeural`	General
Latvian (Latvia)	`lv-LV`	Female	`lv-LV-EveritaNeural`	General
Latvian (Latvia)	`lv-LV`	Male	`lv-LV-NilsNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Female	`lt-LT-OnaNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Male	`lt-LT-LeonasNeural`	General
Malay (Malaysia)	`ms-MY`	Female	`ms-MY-YasminNeural`	General
Malay (Malaysia)	`ms-MY`	Male	`ms-MY-OsmanNeural`	General
Maltese (Malta)	`mt-MT`	Female	`mt-MT-GraceNeural`	General
Maltese (Malta)	`mt-MT`	Male	`mt-MT-JosephNeural`	General
Marathi (India)	`mr-IN`	Female	`mr-IN-AarohiNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Marathi (India)	`mr-IN`	Male	`mr-IN-ManoharNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-IselinNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-PernilleNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Male	`nb-NO-FinnNeural`	General
Persian (Iran)	`fa-IR`	Female	`fa-IR-DilaraNeural` New	General
Persian (Iran)	`fa-IR`	Male	`fa-IR-FaridNeural` New	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-AgnieszkaNeural`	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-ZofiaNeural`	General
Polish (Poland)	`pl-PL`	Male	`pl-PL-MarekNeural`	General
Portuguese (Brazil)	`pt-BR`	Female	`pt-BR-FranciscaNeural`	General, multiple voice styles available using SSML
Portuguese (Brazil)	`pt-BR`	Male	`pt-BR-AntonioNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-FernandaNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-RaquelNeural`	General
Portuguese (Portugal)	`pt-PT`	Male	`pt-PT-DuarteNeural`	General
Romanian (Romania)	`ro-RO`	Female	`ro-RO-AlinaNeural`	General
Romanian (Romania)	`ro-RO`	Male	`ro-RO-EmilNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-DariyaNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-SvetlanaNeural`	General
Russian (Russia)	`ru-RU`	Male	`ru-RU-DmitryNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Slovak (Slovakia)	`sk-SK`	Female	`sk-SK-ViktoriaNeural`	General
Slovak (Slovakia)	`sk-SK`	Male	`sk-SK-LukasNeural`	General
Slovenian (Slovenia)	`sl-SI`	Female	`sl-SI-PetraNeural`	General
Slovenian (Slovenia)	`sl-SI`	Male	`sl-SI-RokNeural`	General
Somali (Somalia)	`so-SO`	Female	`so-SO-UbaxNeural` New	General
Somali (Somalia)	`so-SO`	Male	`so-SO-MuuseNeural` New	General
Spanish (Argentina)	`es-AR`	Female	`es-AR-ElenaNeural`	General
Spanish (Argentina)	`es-AR`	Male	`es-AR-TomasNeural`	General
Spanish (Bolivia)	`es-BO`	Female	`es-BO-SofiaNeural` New	General
Spanish (Bolivia)	`es-BO`	Male	`es-BO-MarceloNeural` New	General
Spanish (Chile)	`es-CL`	Female	`es-CL-CatalinaNeural` New	General
Spanish (Chile)	`es-CL`	Male	`es-CL-LorenzoNeural` New	General
Spanish (Colombia)	`es-CO`	Female	`es-CO-SalomeNeural`	General
Spanish (Colombia)	`es-CO`	Male	`es-CO-GonzaloNeural`	General
Spanish (Costa Rica)	`es-CR`	Female	`es-CR-MariaNeural` New	General
Spanish (Costa Rica)	`es-CR`	Male	`es-CR-JuanNeural` New	General
Spanish (Cuba)	`es-CU`	Female	`es-CU-BelkysNeural` New	General
Spanish (Cuba)	`es-CU`	Male	`es-CU-ManuelNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Dominican Republic)	`es-DO`	Female	`es-DO-RamonaNeural` New	General
Spanish (Dominican Republic)	`es-DO`	Male	`es-DO-EmilioNeural` New	General
Spanish (Ecuador)	`es-EC`	Female	`es-EC-AndreaNeural` New	General
Spanish (Ecuador)	`es-EC`	Male	`es-EC-LuisNeural` New	General
Spanish (El Salvador)	`es-SV`	Female	`es-SV-LorenaNeural` New	General
Spanish (El Salvador)	`es-SV`	Male	`es-SV-RodrigoNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Female	`es-GQ-TeresaNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Male	`es-GQ-JavierNeural` New	General
Spanish (Guatemala)	`es-GT`	Female	`es-GT-MartaNeural` New	General
Spanish (Guatemala)	`es-GT`	Male	`es-GT-AndresNeural` New	General
Spanish (Honduras)	`es-HN`	Female	`es-HN-KarlaNeural` New	General
Spanish (Honduras)	`es-HN`	Male	`es-HN-CarlosNeural` New	General
Spanish (Mexico)	`es-MX`	Female	`es-MX-DaliaNeural`	General
Spanish (Mexico)	`es-MX`	Male	`es-MX-JorgeNeural`	General
Spanish (Nicaragua)	`es-NI`	Female	`es-NI-YolandaNeural` New	General
Spanish (Nicaragua)	`es-NI`	Male	`es-NI-FedericoNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Panama)	`es-PA`	Female	`es-PA-MargaritaNeural` New	General
Spanish (Panama)	`es-PA`	Male	`es-PA-RobertoNeural` New	General
Spanish (Paraguay)	`es-PY`	Female	`es-PY-TaniaNeural` New	General
Spanish (Paraguay)	`es-PY`	Male	`es-PY-MarioNeural` New	General
Spanish (Peru)	`es-PE`	Female	`es-PE-CamilaNeural` New	General
Spanish (Peru)	`es-PE`	Male	`es-PE-AlexNeural` New	General
Spanish (Puerto Rico)	`es-PR`	Female	`es-PR-Karina Neural` New	General
Spanish (Puerto Rico)	`es-PR`	Male	`es-PR-Victor Neural` New	General
Spanish (Spain)	`es-ES`	Female	`es-ES-ElviraNeural`	General
Spanish (Spain)	`es-ES`	Male	`es-ES-AlvaroNeural`	General
Spanish (Uruguay)	`es-UY`	Female	`es-UY-ValentinaNeural` New	General
Spanish (Uruguay)	`es-UY`	Male	`es-UY-MateoNeural` New	General
Spanish (US)	`es-US`	Female	`es-US-PalomaNeural`	General
Spanish (US)	`es-US`	Male	`es-US-AlonsoNeural`	General
Spanish (Venezuela)	`es-VE`	Female	`es-VE-PaolaNeural` New	General
Spanish (Venezuela)	`es-VE`	Male	`es-VE-SebastianNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Sundanese (Indonesia)	`su-ID`	Female	`su-ID-TutiNeural` New	General
Sundanese (Indonesia)	`su-ID`	Male	`su-ID-JajangNeural` New	General
Swahili (Kenya)	`sw-KE`	Female	`sw-KE-ZuriNeural`	General
Swahili (Kenya)	`sw-KE`	Male	`sw-KE-RafikiNeural`	General
Swahili (Tanzania)	`sw-TZ`	Female	`sw-TZ-RehemaNeural` New	General
Swahili (Tanzania)	`sw-TZ`	Male	`sw-TZ-DaudiNeural` New	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-HilleviNeural`	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-SofieNeural`	General
Swedish (Sweden)	`sv-SE`	Male	`sv-SE-MattiasNeural`	General
Tamil (India)	`ta-IN`	Female	`ta-IN-PallaviNeural`	General
Tamil (India)	`ta-IN`	Male	`ta-IN-ValluvarNeural`	General
Tamil (Singapore)	`ta-SG`	Female	`ta-SG-VenbaNeural` New	General
Tamil (Singapore)	`ta-SG`	Male	`ta-SG-AnbuNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Female	`ta-LK-SaranyaNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Male	`ta-LK-KumarNeural` New	General
Telugu (India)	`te-IN`	Female	`te-IN-ShrutiNeural`	General
Telugu (India)	`te-IN`	Male	`te-IN-MohanNeural`	General
Thai (Thailand)	`th-TH`	Female	`th-TH-AcharaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Thai (Thailand)	th-TH	Female	th-TH-PremwadeeNeural	General
Thai (Thailand)	th-TH	Male	th-TH-NiwatNeural	General
Turkish (Turkey)	tr-TR	Female	tr-TR-EmelNeural	General
Turkish (Turkey)	tr-TR	Male	tr-TR-AhmetNeural	General
Ukrainian (Ukraine)	uk-UA	Female	uk-UA-PolinaNeural	General
Ukrainian (Ukraine)	uk-UA	Male	uk-UA-OstapNeural	General
Urdu (India)	ur-IN	Female	ur-IN-GulNeural New	General
Urdu (India)	ur-IN	Male	ur-IN-SalmanNeural New	General
Urdu (Pakistan)	ur-PK	Female	ur-PK-UzmaNeural	General
Urdu (Pakistan)	ur-PK	Male	ur-PK-AsadNeural	General
Uzbek (Uzbekistan)	uz-UZ	Female	uz-UZ-MadinaNeural New	General
Uzbek (Uzbekistan)	uz-UZ	Male	uz-UZ-SardorNeural New	General
Vietnamese (Vietnam)	vi-VN	Female	vi-VN-HoaiMyNeural	General
Vietnamese (Vietnam)	vi-VN	Male	vi-VN-NamMinhNeural	General
Welsh (United Kingdom)	cy-GB	Female	cy-GB-NiaNeural	General
Welsh (United Kingdom)	cy-GB	Male	cy-GB-AledNeural	General
Zulu (South Africa)	zu-ZA	Female	zu-ZA-ThandoNeural New	General
Zulu (South Africa)	zu-ZA	Male	zu-ZA-ThembaNeural New	General

> **IMPORTANT**
>
> The English (United Kingdom) voice `en-GB-MiaNeural` retired on **30 October 2021**. All service requests to `en-GB-MiaNeural` now will be re-directed to `en-GB-SoniaNeural` automatically since **30 October 2021**. If you are using container Neural TTS, please download and deploy the latest version, starting from **30 October 2021**, all requests with previous versions will be rejected.

Neural voices in preview

Below neural voices are in public preview.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Female	`en-US-JennyMultilingualNeural` New	General, multi-lingual capabilities available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaochenNeural` New	Optimized for spontaneous conversation
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyanNeural` New	Optimized for customer service
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoshuangNeural` New	Child voice, optimized for child story and chat; multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoqiuNeural` New	Optimized for narrating

> **IMPORTANT**
>
> Voices in public preview are only available in 3 service regions: East US, West Europe and Southeast Asia.

> **TIP**
>
> `en-US-JennyNeuralMultilingual` supports multiple languages. Check the voices list API for supported languages list.

For more information about regional availability, see regions.

To learn how you can configure and adjust neural voices, such as Speaking Styles, see Speech Synthesis Markup Language.

> **IMPORTANT**
>
> The `en-US-JessaNeural` voice has changed to `en-US-AriaNeural`. If you were using "Jessa" before, convert over to "Aria".

> **TIP**
> You can continue to use the full service name mapping like "Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)" in your speech synthesis requests.

Standard voices

More than 75 standard voices are available in over 45 languages and locales, which allow you to convert text into synthesized speech. For more information about regional availability, see regions.

> **IMPORTANT**
> We are retiring the standard voices on **31st August 2024** and they will no longer be supported after that date.We announced this in emails sent to all existing Speech subscriptions created before **31st August 2021**. During the retiring period (**31st August 2021 - 31st August 2024**), existing standard voice users can continue to use standard voices, but all new users/new speech resources must choose neural voices.

> **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 (Egypt)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-Zhiwei`
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-Matej`
Czech (Czech)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
French (France)	`fr-FR`	Female	`fr-FR-Julie`
French (France)	`fr-FR`	Male	`fr-FR-Paul`
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-HeloisaRUS`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-HeliaRUS`
Romanian (Romania)	`ro-RO`	Male	`ro-RO-Andrei`
Russian (Russia)	`ru-RU`	Female	`ru-RU-EkaterinaRUS`
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".

TIP

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.

Customization

Custom Voice is available in the neural tier (a.k.a, Custom Neural Voice). Based on the Neural TTS technology and the multi-lingual multi-speaker universal model, Custom Neural Voice lets you create synthetic voices that

are rich in speaking styles, or adaptable cross languages. Check below for the languages supported.

> **IMPORTANT**
> The standard tier including the statistical parametric and the concatenative training methods of custom voice is being deprecated and will be retired on 2/29/2024. If you are using non-neural/standard Custom Voice, migrate to Custom Neural Voice immediately to enjoy the better quality and deploy the voices responsibly.

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Arabic (Egypt)	`ar-EG`	Yes	No
Bulgarian (Bulgaria)	`bg-BG`	Yes	No
Chinese (Mandarin, Simplified)	`zh-CN`	Yes	Yes
Chinese (Mandarin, Simplified), English bilingual	`zh-CN` bilingual	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Yes	No
Czech (Czech)	`cs-CZ`	Yes	No
Dutch (Netherlands)	`nl-NL`	Yes	No
English (Australia)	`en-AU`	Yes	Yes
English (Canada)	`en-CA`	Yes	No
English (India)	`en-IN`	Yes	No
English (Ireland)	`en-IE`	Yes	No
English (United Kingdom)	`en-GB`	Yes	Yes
English (United States)	`en-US`	Yes	Yes
French (Canada)	`fr-CA`	Yes	Yes
French (France)	`fr-FR`	Yes	Yes
German (Austria)	`de-AT`	Yes	No
German (Germany)	`de-DE`	Yes	Yes
Hungarian (Hungary)	`hu-HU`	Yes	No
Italian (Italy)	`it-IT`	Yes	Yes

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Japanese (Japan)	`ja-JP`	Yes	Yes
Korean (Korea)	`ko-KR`	Yes	Yes
Norwegian (Bokmål, Norway)	`nb-NO`	Yes	No
Portuguese (Brazil)	`pt-BR`	Yes	Yes
Portuguese (Portugal)	`pt-PT`	Yes	No
Russian (Russia)	`ru-RU`	Yes	Yes
Slovak (Slovakia)	`sk-SK`	Yes	No
Spanish (Mexico)	`es-MX`	Yes	Yes
Spanish (Spain)	`es-ES`	Yes	Yes
Turkish (Turkey)	`tr-TR`	Yes	No
Vietnamese (Vietnam)	`vi-VN`	Yes	No

Select the right locale that matches the training data you have to train a custom voice model. For example, if the recording data you have is spoken in English with a British accent, select `en-GB`.

> **NOTE**
> We do not support bi-lingual model training in Custom Voice, except for the Chinese-English bi-lingual. Select "Chinese-English bilingual" if you want to train a Chinese voice that can speak English as well. Chinese-English bilingual model training using the standard method is available in North Europe and North Central US only. Custom Neural Voice training is available in UK South and East US.

Speech translation

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

Text languages

TEXT LANGUAGE	LANGUAGE CODE
Afrikaans	`af`
Albanian	`sq`
Amharic	`am`

TEXT LANGUAGE	LANGUAGE CODE
Arabic	`ar`
Armenian	`hy`
Assamese	`as`
Azerbaijani	`az`
Bangla	`bn`
Bosnian (Latin)	`bs`
Bulgarian	`bg`
Cantonese (Traditional)	`yue`
Catalan	`ca`
Chinese (Literary)	`lzh`
Chinese Simplified	`zh-Hans`
Chinese Traditional	`zh-Hant`
Croatian	`hr`
Czech	`cs`
Danish	`da`
Dari	`prs`
Dutch	`nl`
English	`en`
Estonian	`et`
Fijian	`fj`
Filipino	`fil`
Finnish	`fi`
French	`fr`
French (Canada)	`fr-ca`

TEXT LANGUAGE	LANGUAGE CODE
German	`de`
Greek	`el`
Gujarati	`gu`
Haitian Creole	`ht`
Hebrew	`he`
Hindi	`hi`
Hmong Daw	`mww`
Hungarian	`hu`
Icelandic	`is`
Indonesian	`id`
Inuktitut	`iu`
Irish	`ga`
Italian	`it`
Japanese	`ja`
Kannada	`kn`
Kazakh	`kk`
Khmer	`km`
Klingon	`tlh-Latn`
Klingon (plqaD)	`tlh-Piqd`
Korean	`ko`
Kurdish (Central)	`ku`
Kurdish (Northern)	`kmr`
Lao	`lo`
Latvian	`lv`

TEXT LANGUAGE	LANGUAGE CODE
Lithuanian	`lt`
Malagasy	`mg`
Malay	`ms`
Malayalam	`ml`
Maltese	`mt`
Maori	`mi`
Marathi	`mr`
Myanmar	`my`
Nepali	`ne`
Norwegian	`nb`
Odia	`or`
Pashto	`ps`
Persian	`fa`
Polish	`pl`
Portuguese (Brazil)	`pt`
Portuguese (Portugal)	`pt-pt`
Punjabi	`pa`
Queretaro Otomi	`otq`
Romanian	`ro`
Russian	`ru`
Samoan	`sm`
Serbian (Cyrillic)	`sr-Cyrl`
Serbian (Latin)	`sr-Latn`
Slovak	`sk`

TEXT LANGUAGE	LANGUAGE CODE
Slovenian	`sl`
Spanish	`es`
Swahili	`sw`
Swedish	`sv`
Tahitian	`ty`
Tamil	`ta`
Telugu	`te`
Thai	`th`
Tigrinya	`ti`
Tongan	`to`
Turkish	`tr`
Ukrainian	`uk`
Urdu	`ur`
Vietnamese	`vi`
Welsh	`cy`
Yucatec Maya	`yua`

Speaker Recognition

Speaker recognition is mostly language agnostic. We built a universal model for text-independent speaker recognition by combining various data sources from multiple languages. We have tuned and evaluated the model on the languages and locales that appear in the following table. See the overview for additional information on Speaker Recognition.

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (US)	`en-US`	yes	yes	yes
Chinese (Mandarin, simplified)	`zh-CN`	n/a	yes	yes
English (Australia)	`en-AU`	n/a	yes	yes

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (Canada)	`en-CA`	n/a	yes	yes
English (India)	`en-IN`	n/a	yes	yes
English (UK)	`en-GB`	n/a	yes	yes
French (Canada)	`fr-CA`	n/a	yes	yes
French (France)	`fr-FR`	n/a	yes	yes
German (Germany)	`de-DE`	n/a	yes	yes
Italian	`it-IT`	n/a	yes	yes
Japanese	`ja-JP`	n/a	yes	yes
Portuguese (Brazil)	`pt-BR`	n/a	yes	yes
Spanish (Mexico)	`es-MX`	n/a	yes	yes
Spanish (Spain)	`es-ES`	n/a	yes	yes

Custom Keyword and Keyword Verification

The following table outlines supported languages for Custom Keyword and Keyword Verification.

LANGUAGE	LOCALE (BCP-47)	CUSTOM KEYWORD	KEYWORD VERIFICATION
Chinese (Mandarin, Simplified)	zh-CN	Yes	Yes
English (United States)	en-US	Yes	Yes
Japanese (Japan)	ja-JP	No	Yes
Portuguese (Brazil)	pt-BR	No	Yes

Next steps

- Create a free Azure account
- See how to recognize speech in C#

Speech service supported regions

11/23/2021 • 5 minutes to read • Edit Online

The Speech service allows your application to convert audio to text, perform speech translation, and covert text to speech. The service is available in multiple regions with unique endpoints for the Speech SDK and REST APIs.

The Speech portal, where you can perform custom configurations to your speech experience for all regions, is available at speech.microsoft.com.

Keep in mind the following points when considering regions:

- If your application uses a Speech SDK, you provide the region identifier, such as `westus`, when creating a speech configuration. Make sure the region matches the region of your subscription.
- If your application uses one of the Speech service's REST APIs, the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

> **NOTE**
>
> Speech Services doesn't store/process customer data outside the region the customer deploys the service instance in.

Speech SDK

In the Speech SDK, the region is specified as a parameter (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

Speech-to-text, text-to-speech, and translation

The Speech service is available in these regions for **speech recognition**, **text-to-speech**, and **translation**:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`

GEOGRAPHY	REGION	REGION IDENTIFIER
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

If you plan to train a custom model with audio data, use one of the regions with dedicated hardware for faster training. You can use the REST API to copy the fully trained model to another region later.

Intent recognition

Available regions for **intent recognition** via the Speech SDK are the following:

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 in these regions:

GLOBAL REGION	REGION	REGION IDENTIFIER
North America	West US	`westus`
North America	West US 2	`westus2`
North America	East US	`eastus`
North America	East US 2	`eastus2`
North America	West Central US	`westcentralus`
North America	South Central US	`southcentralus`
Europe	West Europe	`westeurope`
Europe	North Europe	`northeurope`
Asia	East Asia	`eastasia`
Asia	Southeast Asia	`southeastasia`
India	Central India	`centralindia`

Speaker Recognition

Available regions for **Speaker Recognition** are the following:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Central India	`centralindia`
Australia	Australia East	`australiaeast`
Europe	North Europe	`northeurope`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	West Europe	`westeurope`
Europe	UK South	`uksouth`

REST APIs

The Speech service also exposes REST endpoints for speech-to-text, text-to-speech and speaker recognition requests.

Speech-to-text

For speech-to-text reference documentation, see Speech-to-text REST API.

The endpoint for the REST API has this format:

```
https://<REGION_IDENTIFIER>.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

> **NOTE**
>
> The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is:
>
> ```
> https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
> ```
>
> .

Text-to-speech

For text-to-speech reference documentation, see Text-to-speech REST API.

Neural and standard voices

Use this table to determine **availability of neural and standard voices** by region/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`
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/v`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`

REGION	ENDPOINT
South Central US	`https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v`
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`

> **TIP**
>
> Voices in preview are only available in these 3 regions: East US, West Europe and Southeast Asia.

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	`https://australiaeast.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Brazil South	`https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
Canada Central	`https://canadacentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Central US	`https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East Asia	`https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US	`https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US 2	`https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
France Central	`https://francecentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
India Central	`https://centralindia.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
Japan East	`https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
Japan West	`https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

REGION	ENDPOINT
Korea Central	`https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
North Central US	`https://northcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
North Europe	`https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
South Central US	`https://southcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
Southeast Asia	`https://southeastasia.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
UK South	`https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West Europe	`https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US	`https://westus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US 2	`https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

Custom Neural Voice

The following table details regional support for Custom Neural Voice features.

FEATURE	SUPPORTED REGIONS
Voice model hosting	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Realtime Characters	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Long Audio Characters	East US, West Europe, Southeast Asia, UK South, India Central
Custom Neural Training	East US, Southeast Asia, UK South

Long audio API

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.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`

Speaker Recognition

For speaker recognition reference documentation, see Speaker Recognition REST API. Available regions are the same as Speaker Recognition SDK.

Speech service phonetic sets

11/23/2021 • 13 minutes to read • Edit Online

The Speech service defines phonetic alphabets ("phone sets" for short), consisting of seven languages; `en-US`, `fr-FR`, `de-DE`, `es-ES`, `ja-JP`, `zh-CN`, and `zh-TW`. The Speech service phone sets typically map to the International Phonetic Alphabet (IPA) . Speech service phone sets are used in conjunction with the Speech Synthesis Markup Language (SSML), as part of the Text-to-speech service offering. In this article, you'll learn how these phone sets are mapped and when to use which phone set.

- en-US
- fr-FR
- de-DE
- es-ES
- zh-CN
- zh-TW
- ja-JP

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/	Speech service phone set put 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/	Speech service phone set put stress after the vowel of the sub-stressed syllable

English vowels

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
iy	i	e**a**t	f**ee**l	val**ley**
ih	ɪ	**i**f	f**i**ll	
ey	eɪ	**a**te	g**a**te	d**ay**
eh	ɛ	**e**very	p**e**t	m**eh** (rare word finally)
ae	æ	**a**ctive	c**a**t	n**ah** (rare word finally)
aa	ɑ	**o**bstinate	p**o**ppy	r**ah** (rare word finally)

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
ao	ɔ	orange	cause	Utah
uh	ʊ	book		
ow	oʊ	old	clone	go
uw	u	Uber	boost	too
ah	ʌ	uncle	cut	
ay	aɪ	ice	bite	fly
aw	aʊ	out	south	cow
oy	ɔɪ	oil	join	toy
y uw	ju	Yuma	human	few
ax	ə	ago	woman	area

English R-colored vowels

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
ih r	ɪɹ	ears	tiramisu	near
eh r	ɛɹ	airplane	apparently	scare
uh r	ʊɹ			cure
ay r	aɪɹ	Ireland	fireplace	choir
aw r	aʊɹ	hours	powerful	sour
ao r	ɔɹ	orange	moral	soar
aa r	ɑɹ	artist	start	car
er r	ɝ	earth	bird	fur
ax r	ɚ		allergy	supper

English Semivowels

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
w	w	with, suede	always	
y	j	yard, few	onion	

English aspirated oral stops

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
p	p	put	happen	flap
b	b	big	number	crab
t	t	talk	capital	sought
d	d	dig	random	rod
k	k	cut	slacker	Iraq
g	g	go	ago	drag

English Nasal stops

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
m	m	mat, smash	camera	room
n	n	no, snow	tent	chicken
ng	ŋ		link	sing

English fricatives

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
f	f	fork	left	half
v	v	value	event	love
th	θ	thin	empathy	month
dh	ð	then	mother	smooth
s	s	sit	risk	facts
z	z	zap	busy	kids
sh	ʃ	sh e	abbreviation	rush
zh	ʒ	Jacques	pleasure	garage
h	h	help	enhance	a-ha!

English affricates

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
ch	tʃ	**ch**in	fu**t**ure	atta**ch**
jh	dʒ	**j**oy	ori**g**inal	oran**g**e

English approximants

SAPI	IPA	EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
l	l	**l**id, g**l**ad	pa**l**ace	chi**ll**
r	ɹ	**r**ed, b**r**ing	bo**rr**ow	ta**r**

What is speech translation?

11/23/2021 • 2 minutes to read • Edit Online

In this overview, 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. With the Speech SDK, your applications, tools, and devices have access to source transcriptions and translation outputs for provided audio. Interim transcription and translation results are returned as speech is detected, and final results can be converted into synthesized speech.

This documentation contains the following article types:

- **Quickstarts** are getting-started instructions to guide you through making requests to the service.
- **How-to guides** contain instructions for using the service in more specific or customized ways.
- **Concepts** provide in-depth explanations of the service functionality and features.
- **Tutorials** are longer guides that show you how to use the service as a component in broader business solutions.

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

See the quickstart to get started with speech translation. The speech translation service is available via the Speech SDK and the Speech CLI.

Sample code

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

- Speech-to-text and translation samples (SDK)

Migration guides

If your applications, tools, or products are using the Translator Speech API, we've created guides to help you migrate to the Speech service.

- Migrate from the Translator Speech API to the Speech service

Reference docs

- Speech SDK
- Speech Devices SDK
- REST API: Speech-to-text
- REST API: Text-to-speech
- REST API: Batch transcription and customization

Next steps

- Complete the speech translation quickstart
- Get a Speech service subscription key for free
- Get the Speech SDK

Get started with speech translation

11/23/2021 • 44 minutes to read • Edit Online

One of the core features of the Speech service is the ability to recognize human speech and translate it to other languages. In this quickstart you learn how to use the Speech SDK in your apps and products to perform high-quality speech translation. This quickstart covers topics including:

- Translating speech-to-text
- Translating speech to multiple target languages
- Performing direct speech-to-speech translation

Skip to samples on GitHub

If you want to skip straight to sample code, see the C# quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, follow the instructions under the Get the Speech SDK section of the *About the Speech SDK* article.

Import dependencies

To run the examples in this article, include the following `using` statements at the top of the *Program.cs* file.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.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 subscription key and region. The `Program` class contains two `static readonly string` values that are assigned from the host machines environment variables, namely `SPEECH__SUBSCRIPTION__KEY` and `SPEECH__SERVICE__REGION`. Both of these fields are at the class scope, making them accessible within method bodies of the class. For more information on environment variables, see environment variables and application configuration.

```
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;
}
```

Create a speech translation configuration

To call the Speech service using the Speech SDK, you need to create a `SpeechTranslationConfig`. 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.

There are a few ways that you can initialize a `SpeechTranslationConfig`:

- 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 take a look at how a `SpeechTranslationConfig` is created using a key and region. Get these credentials by following steps in Try the Speech service for free.

```
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 source language

One common task of speech translation is specifying the input (or source) language. Let's take a look at how you would change the input language to Italian. In your code, interact with the `SpeechTranslationConfig` instance, assigning to the `SpeechRecognitionLanguage` property.

```
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. You can provide any value in the **Locale** column in the list of supported locales/languages.

Add translation language

Another common task of speech translation is to specify target translation languages, at least one is required but using multiple is supported. The following code snippet sets both French and German as translation language targets.

```
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`, the next step is to initialize a `TranslationRecognizer`. When you initialize a `TranslationRecognizer`, you'll need to pass it your `translationConfig`. The configuration object provides the credentials that the speech service requires to validate your request.

If you're recognizing speech using your device's default microphone, here's what the `TranslationRecognizer` should look like:

```
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'll need to create an `AudioConfig` and provide the

`audioConfig` parameter when initializing your `TranslationRecognizer`.

> **TIP**
>
> Learn how to get the device ID for your audio input device.

First, you'll reference the `AudioConfig` object as follows:

```
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'll still need to provide an `audioConfig`. However, when you create an `AudioConfig`, instead of calling `FromDefaultMicrophoneInput`, you'll call `FromWavFileInput` and pass the `filename` parameter.

```
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.

```
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.Write($"Say something in '{fromLanguage}' and ");
    Console.WriteLine($"we'll translate into '{string.Join("', '", toLanguages)}'.\n");

    var result = await recognizer.RecognizeOnceAsync();
    if (result.Reason == ResultReason.TranslatedSpeech)
    {
        Console.WriteLine($"Recognized: \"{result.Text}\":");
        foreach (var (language, translation) in result.Translations)
        {
            Console.WriteLine($"Translated into '{language}': {translation}");
        }
    }
}
```

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, 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` and provide an event handler for the `Synthesizing` event, get the audio. The following example saves the translated audio as a *.wav* file.

> **IMPORTANT**
>
> The event-based synthesis only works with a single translation, **do not** add multiple target translation languages. Additionally, the `VoiceName` should be the same language as the target translation language, for example; `"de"` could map to `"de-DE-Hedda"`.

```
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

The `Translations` dictionary can be used to synthesize audio from the translation text. Iterate through each translation, and synthesize the translation. When 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, and each translation is then synthesized to an audio file in the corresponding neural language.

```
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.Write($"Say something in '{fromLanguage}' and ");
    Console.WriteLine($"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 may not know which input languages to specify. Using language identification allows you to specify up to ten possible input languages, and automatically translate into your target languages.

The following example uses continuous translation from an audio file, and 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 are defined in the `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**.

```
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 = SpeechConfig.FromEndpoint(endpointUrl, "<paste-your-subscription-key>");

    // Source lang 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_AutoDetectSourceLanguageResult);

                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_AutoDetectSourceLanguageResult);

                    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 update the subscription info?");
                }

                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);
            };

            // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
            Console.WriteLine("Start translation...");
            await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

            Task.WaitAny(new[] { stopTranslation.Task });
            await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
        }
    }
}
```

One of the core features of the Speech service is the ability to recognize human speech and translate it to other languages. In this quickstart you learn how to use the Speech SDK in your apps and products to perform high-quality speech translation. This quickstart covers topics including:

- Translating speech-to-text
- Translating speech to multiple target languages
- Performing direct speech-to-speech translation

Skip to samples on GitHub

If you want to skip straight to sample code, see the C++ quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, follow the instructions under the Get the Speech SDK section of the *About the Speech SDK* article.

Import dependencies

To run the examples in this article, include the following `#include` and `using` statements at the top of the C++ code file.

```
#include <iostream> // cin, cout
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Audio;
using namespace Microsoft::CognitiveServices::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 subscription key and region. The C++ code file contains two string values that are assigned from the host machines environment variables, namely `SPEECH__SUBSCRIPTION__KEY` and `SPEECH__SERVICE__REGION`. Both of these fields are at the class scope, making them accessible within method bodies of the class. For more information on environment variables, see environment variables and application configuration.

```
auto SPEECH__SUBSCRIPTION__KEY = getenv("SPEECH__SUBSCRIPTION__KEY");
auto SPEECH__SERVICE__REGION = getenv("SPEECH__SERVICE__REGION");
```

Create a speech translation configuration

To call the Speech service using the Speech SDK, you need to create a `SpeechTranslationConfig`. 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.

There are a few ways that you can initialize a `SpeechTranslationConfig`:

- 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 take a look at how a `SpeechTranslationConfig` is created using a key and region. Get these credentials by following steps in Try the Speech service for free.

```
auto SPEECH__SUBSCRIPTION__KEY = getenv("SPEECH__SUBSCRIPTION__KEY");
auto SPEECH__SERVICE__REGION = getenv("SPEECH__SERVICE__REGION");

void translateSpeech() {
    auto config =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);
}

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

Change source language

One common task of speech translation is specifying the input (or source) language. Let's take a look at how you would change the input language to Italian. In your code, interact with the `SpeechTranslationConfig` instance, calling the `SetSpeechRecognitionLanguage` method.

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    // Source (input) language
    translationConfig->SetSpeechRecognitionLanguage("it-IT");
}
```

The `SpeechRecognitionLanguage` property expects a language-locale format string. You can provide any value in the **Locale** column in the list of supported locales/languages.

Add 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.

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    translationConfig->SetSpeechRecognitionLanguage("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`, the next step is to initialize a `TranslationRecognizer`. When you initialize a `TranslationRecognizer`, you'll need to pass it your `translationConfig`. The configuration object provides the credentials that the speech service requires to validate your request.

If you're recognizing speech using your device's default microphone, here's what the `TranslationRecognizer` should look like:

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    auto fromLanguage = "en-US";
    auto toLanguages = { "it", "fr", "de" };
    translationConfig->SetSpeechRecognitionLanguage(fromLanguage);
    for (auto language : toLanguages) {
        translationConfig->AddTargetLanguage(language);
    }

    auto recognizer = TranslationRecognizer::FromConfig(translationConfig);
}
```

If you want to specify the audio input device, then you'll need to create an `AudioConfig` and provide the `audioConfig` parameter when initializing your `TranslationRecognizer`.

> **TIP**
>
> Learn how to get the device ID for your audio input device.

First, you'll reference the `AudioConfig` object as follows:

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    auto fromLanguage = "en-US";
    auto toLanguages = { "it", "fr", "de" };
    translationConfig->SetSpeechRecognitionLanguage(fromLanguage);
    for (auto language : toLanguages) {
        translationConfig->AddTargetLanguage(language);
    }

    auto audioConfig = AudioConfig::FromDefaultMicrophoneInput();
    auto recognizer = TranslationRecognizer::FromConfig(translationConfig, audioConfig);
}
```

If you want to provide an audio file instead of using a microphone, you'll still need to provide an `audioConfig`. However, when you create an `AudioConfig`, instead of calling `FromDefaultMicrophoneInput`, you'll call `FromWavFileInput` and pass the `filename` parameter.

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    auto fromLanguage = "en-US";
    auto toLanguages = { "it", "fr", "de" };
    translationConfig->SetSpeechRecognitionLanguage(fromLanguage);
    for (auto language : toLanguages) {
        translationConfig->AddTargetLanguage(language);
    }

    auto audioConfig = AudioConfig::FromWavFileInput("YourAudioFile.wav");
    auto recognizer = TranslationRecognizer::FromConfig(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.

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    string fromLanguage = "en-US";
    string toLanguages[3] = { "it", "fr", "de" };
    translationConfig->SetSpeechRecognitionLanguage(fromLanguage);
    for (auto language : toLanguages) {
        translationConfig->AddTargetLanguage(language);
    }

    auto recognizer = TranslationRecognizer::FromConfig(translationConfig);
    cout << "Say something in '" << fromLanguage << "' and we'll translate...\n";

    auto result = recognizer->RecognizeOnceAsync().get();
    if (result->Reason == ResultReason::TranslatedSpeech)
    {
        cout << "Recognized: \"" << result->Text << "\"" << std::endl;
        for (auto pair : result->Translations)
        {
            auto language = pair.first;
            auto translation = pair.second;
            cout << "Translated into '" << language << "': " << translation << std::endl;
        }
    }
}
```

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, 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 `SetVoiceName` and provide an event handler for the `Synthesizing` event, get the audio. The following example saves the translated audio as a *.wav* file.

> **IMPORTANT**
>
> The event-based synthesis only works with a single translation, **do not** add multiple target translation languages. Additionally, the `SetVoiceName` should be the same language as the target translation language, for example; `"de"` could map to `"de-DE-Hedda"`.

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    auto fromLanguage = "en-US";
    auto toLanguage = "de";
    translationConfig->SetSpeechRecognitionLanguage(fromLanguage);
    translationConfig->AddTargetLanguage(toLanguage);

    // See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
    translationConfig->SetVoiceName("de-DE-Hedda");

    auto recognizer = TranslationRecognizer::FromConfig(translationConfig);
    recognizer->Synthesizing.Connect([](const TranslationSynthesisEventArgs& e)
        {
            auto audio = e.Result->Audio;
            auto size = audio.size();
            cout << "Audio synthesized: " << size << " byte(s)" << (size == 0 ? "(COMPLETE)" : "") <<
std::endl;

            if (size > 0) {
                ofstream file("translation.wav", ios::out | ios::binary);
                auto audioData = audio.data();
                file.write((const char*)audioData, sizeof(audio[0]) * size);
                file.close();
            }
        });

    cout << "Say something in '" << fromLanguage << "' and we'll translate...\n";

    auto result = recognizer->RecognizeOnceAsync().get();
    if (result->Reason == ResultReason::TranslatedSpeech)
    {
        cout << "Recognized: \"" << result->Text << "\"" << std::endl;
        for (auto pair : result->Translations)
        {
            auto language = pair.first;
            auto translation = pair.second;
            cout << "Translated into '" << language << "': " << translation << std::endl;
        }
    }
}
```

Manual synthesis

The `Translations` dictionary can be used to synthesize audio from the translation text. Iterate through each translation, and synthesize the translation. When creating a `SpeechSynthesizer` instance, the `SpeechConfig` object needs to have its `SetSpeechSynthesisVoiceName` property set to the desired voice. The following example translates to five languages, and each translation is then synthesized to an audio file in the corresponding neural language.

```
void translateSpeech() {
    auto translationConfig =
        SpeechTranslationConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    auto fromLanguage = "en-US";
    auto toLanguages = { "de", "en", "it", "pt", "zh-Hans" };
    translationConfig->SetSpeechRecognitionLanguage(fromLanguage);
    for (auto language : toLanguages) {
        translationConfig->AddTargetLanguage(language);
    }

    auto recognizer = TranslationRecognizer::FromConfig(translationConfig);

    cout << "Say something in '" << fromLanguage << "' and we'll translate...\n";

    auto result = recognizer->RecognizeOnceAsync().get();
    if (result->Reason == ResultReason::TranslatedSpeech)
    {
        // See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
        map<string, string> languageToVoiceMap;
        languageToVoiceMap["de"] = "de-DE-KatjaNeural";
        languageToVoiceMap["en"] = "en-US-AriaNeural";
        languageToVoiceMap["it"] = "it-IT-ElsaNeural";
        languageToVoiceMap["pt"] = "pt-BR-FranciscaNeural";
        languageToVoiceMap["zh-Hans"] = "zh-CN-XiaoxiaoNeural";

        cout << "Recognized: \"" << result->Text << "\"" << std::endl;
        for (auto pair : result->Translations)
        {
            auto language = pair.first;
            auto translation = pair.second;
            cout << "Translated into '" << language << "': " << translation << std::endl;

            auto speech_config =
                SpeechConfig::FromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);
            speech_config->SetSpeechSynthesisVoiceName(languageToVoiceMap[language]);

            auto audio_config = AudioConfig::FromWavFileOutput(language + "-translation.wav");
            auto synthesizer = SpeechSynthesizer::FromConfig(speech_config, audio_config);

            synthesizer->SpeakTextAsync(translation).get();
        }
    }
}
```

For more information about speech synthesis, see the basics of speech synthesis.

Multi-lingual translation with language identification

In many scenarios, you may not know which input languages to specify. Using language identification allows you to specify up to ten possible input languages, and automatically translate into your target languages.

The following example uses continuous translation from an audio file, and 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 are defined in the `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**.

```
using namespace std;
```

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

void MultiLingualTranslation()
{
    auto region = "<paste-your-region>";
    // currently the v2 endpoint is required for this design pattern
    auto endpointString = std::format("wss://{}.stt.speech.microsoft.com/speech/universal/v2", region);
    auto config = SpeechConfig::FromEndpoint(endpointString, "<paste-your-subscription-key>");

    config->SetProperty(PropertyId::SpeechServiceConnection_ContinuousLanguageIdPriority, "Latency");
    auto autoDetectSourceLanguageConfig = AutoDetectSourceLanguageConfig::FromLanguages({ "en-US", "zh-CN"
});

    promise<void> recognitionEnd;
    // Source lang is required, but is currently NoOp
    auto fromLanguage = "en-US";
    config->SetSpeechRecognitionLanguage(fromLanguage);
    config->AddTargetLanguage("de");
    config->AddTargetLanguage("fr");

    auto audioInput = AudioConfig::FromWavFileInput("path-to-your-audio-file.wav");
    auto recognizer = TranslationRecognizer::FromConfig(config, autoDetectSourceLanguageConfig, audioInput);

    recognizer->Recognizing.Connect([](const TranslationRecognitionEventArgs& e)
        {
            std::string lidResult = e.Result-
>Properties.GetProperty(PropertyId::SpeechServiceConnection_AutoDetectSourceLanguageResult);

            cout << "Recognizing in Language = "<< lidResult << ":" << e.Result->Text << std::endl;
            for (const auto& it : e.Result->Translations)
            {
                cout << "  Translated into '" << it.first.c_str() << "': " << it.second.c_str() <<
std::endl;
            }
        });

    recognizer->Recognized.Connect([](const TranslationRecognitionEventArgs& e)
        {
            if (e.Result->Reason == ResultReason::TranslatedSpeech)
            {
                std::string lidResult = e.Result-
>Properties.GetProperty(PropertyId::SpeechServiceConnection_AutoDetectSourceLanguageResult);
                cout << "RECOGNIZED in Language = " << lidResult << ": Text=" << e.Result->Text <<
std::endl;
            }
            else if (e.Result->Reason == ResultReason::RecognizedSpeech)
            {
                cout << "RECOGNIZED: Text=" << e.Result->Text << " (text could not be translated)" <<
std::endl;
            }
            else if (e.Result->Reason == ResultReason::NoMatch)
            {
                cout << "NOMATCH: Speech could not be recognized." << std::endl;
            }

            for (const auto& it : e.Result->Translations)
            {
                cout << "  Translated into '" << it.first.c_str() << "': " << it.second.c_str() <<
std::endl;
            }
        });

    recognizer->Canceled.Connect([&recognitionEnd](const TranslationRecognitionCanceledEventArgs& e)
        {
            cout << "CANCELED: Reason=" << (int)e.Reason << std::endl;
            if (e.Reason == CancellationReason::Error)
            {
```

```
                cout << "CANCELED: ErrorCode=" << (int)e.ErrorCode << std::endl;
                cout << "CANCELED: ErrorDetails=" << e.ErrorDetails << std::endl;
                cout << "CANCELED: Did you update the subscription info?" << std::endl;

                recognitionEnd.set_value();
            }
        });

    recognizer->Synthesizing.Connect([](const TranslationSynthesisEventArgs& e)
        {
            auto size = e.Result->Audio.size();
            cout << "Translation synthesis result: size of audio data: " << size
                << (size == 0 ? "(END)" : "");
        });

    recognizer->SessionStopped.Connect([&recognitionEnd](const SessionEventArgs& e)
        {
            cout << "Session stopped.";
            recognitionEnd.set_value();
        });

    // Starts continuos recognition. Use StopContinuousRecognitionAsync() to stop recognition.
    recognizer->StartContinuousRecognitionAsync().get();
    recognitionEnd.get_future().get();
    recognizer->StopContinuousRecognitionAsync().get();
}
```

One of the core features of the Speech service is the ability to recognize human speech and translate it to other languages. In this quickstart you learn how to use the Speech SDK in your apps and products to perform high-quality speech translation. This quickstart covers topics including:

- Translating speech-to-text
- Translating speech to multiple target languages
- Performing direct speech-to-speech translation

Skip to samples on GitHub

If you want to skip straight to sample code, see the Java quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, follow the instructions under the Get the Speech SDK section of the *About the Speech SDK* article.

Import dependencies

To run the examples in this article, include the following `import` statements at the top of the *.*Java* code file.

```
package speech;

import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.*;
import com.microsoft.cognitiveservices.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 subscription key and region. The Java code file contains two `static final String` values that are assigned from the host machines environment variables, namely `SPEECH__SUBSCRIPTION__KEY` and `SPEECH__SERVICE__REGION`. Both of these fields are at the class scope, making them accessible within method bodies of the class. For more information on environment variables, see environment variables and application configuration.

```
public class App {

    static final String SPEECH__SUBSCRIPTION__KEY = System.getenv("SPEECH__SUBSCRIPTION__KEY");
    static final String SPEECH__SERVICE__REGION = System.getenv("SPEECH__SERVICE__REGION");

    public static void main(String[] args) { }
}
```

Create a speech translation configuration

To call the Speech service using the Speech SDK, you need to create a `SpeechTranslationConfig`. 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.

There are a few ways that you can initialize a `SpeechTranslationConfig`:

- 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 take a look at how a `SpeechTranslationConfig` is created using a key and region. Get these credentials by following steps in Try the Speech service for free.

```
public class App {

    static final String SPEECH__SUBSCRIPTION__KEY = System.getenv("SPEECH__SERVICE__KEY");
    static final String SPEECH__SERVICE__REGION = System.getenv("SPEECH__SERVICE__REGION");

    public static void main(String[] args) {
        try {
            translateSpeech();
            System.exit(0);
        } catch (Exception ex) {
            System.out.println(ex);
            System.exit(1);
        }
    }

    static void translateSpeech() {
        SpeechTranslationConfig config = SpeechTranslationConfig.fromSubscription(
            SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);
    }
}
```

Change source language

One common task of speech translation is specifying the input (or source) language. Let's take a look at how you would change the input language to Italian. In your code, interact with the `SpeechTranslationConfig` instance, calling the `setSpeechRecognitionLanguage` method.

```
static void translateSpeech() {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    // Source (input) language
    translationConfig.setSpeechRecognitionLanguage("it-IT");
}
```

The `setSpeechRecognitionLanguage` function expects a language-locale format string. You can provide any value in the **Locale** column in the list of supported locales/languages.

Add 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.

```
static void translateSpeech() {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    translationConfig.setSpeechRecognitionLanguage("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`, the next step is to initialize a `TranslationRecognizer`. When you initialize a `TranslationRecognizer`, you'll need to pass it your `translationConfig`. The configuration object provides the credentials that the speech service requires to validate your request.

If you're recognizing speech using your device's default microphone, here's what the `TranslationRecognizer` should look like:

```
static void translateSpeech() {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    String fromLanguage = "en-US";
    String[] toLanguages = { "it", "fr", "de" };
    translationConfig.setSpeechRecognitionLanguage(fromLanguage);
    for (String language : toLanguages) {
        translationConfig.addTargetLanguage(language);
    }

    try (TranslationRecognizer recognizer = new TranslationRecognizer(translationConfig)) {
    }
}
```

If you want to specify the audio input device, then you'll need to create an `AudioConfig` and provide the `audioConfig` parameter when initializing your `TranslationRecognizer`.

> **TIP**
>
> Learn how to get the device ID for your audio input device.

First, you'll reference the `AudioConfig` object as follows:

```
static void translateSpeech() {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    String fromLanguage = "en-US";
    String[] toLanguages = { "it", "fr", "de" };
    translationConfig.setSpeechRecognitionLanguage(fromLanguage);
    for (String language : toLanguages) {
        translationConfig.addTargetLanguage(language);
    }

    AudioConfig audioConfig = AudioConfig.fromDefaultMicrophoneInput();
    try (TranslationRecognizer recognizer = new TranslationRecognizer(translationConfig, audioConfig)) {

    }
}
```

If you want to provide an audio file instead of using a microphone, you'll still need to provide an `audioConfig`. However, when you create an `AudioConfig`, instead of calling `fromDefaultMicrophoneInput`, you'll call `fromWavFileInput` and pass the `filename` parameter.

```
static void translateSpeech() {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    String fromLanguage = "en-US";
    String[] toLanguages = { "it", "fr", "de" };
    translationConfig.setSpeechRecognitionLanguage(fromLanguage);
    for (String language : toLanguages) {
        translationConfig.addTargetLanguage(language);
    }

    AudioConfig audioConfig = AudioConfig.fromWavFileInput("YourAudioFile.wav");
    try (TranslationRecognizer 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.

```
static void translateSpeech() throws ExecutionException, InterruptedException {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    String fromLanguage = "en-US";
    String[] toLanguages = { "it", "fr", "de" };
    translationConfig.setSpeechRecognitionLanguage(fromLanguage);
    for (String language : toLanguages) {
        translationConfig.addTargetLanguage(language);
    }

    try (TranslationRecognizer recognizer = new TranslationRecognizer(translationConfig)) {
        System.out.printf("Say something in '%s' and we'll translate...", fromLanguage);

        TranslationRecognitionResult result = recognizer.recognizeOnceAsync().get();
        if (result.getReason() == ResultReason.TranslatedSpeech) {
            System.out.printf("Recognized: \"%s\"\n", result.getText());
            for (Map.Entry<String, String> pair : result.getTranslations().entrySet()) {
                System.out.printf("Translated into '%s': %s\n", pair.getKey(), pair.getValue());
            }
        }
    }
}
```

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 `getTranslations` function returns a dictionary with the key as the target translation language and the value is the translated text. Recognized speech can be translated, 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 `setVoiceName` and provide an event handler for the `synthesizing` event, get the audio. The following example saves the translated audio as a *.wav* file.

> **IMPORTANT**
>
> The event-based synthesis only works with a single translation, **do not** add multiple target translation languages. Additionally, the `setVoiceName` should be the same language as the target translation language, for example; `"de"` could map to `"de-DE-Hedda"`.

```
static void translateSpeech() throws ExecutionException, FileNotFoundException, InterruptedException,
IOException {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    String fromLanguage = "en-US";
    String toLanguage = "de";
    translationConfig.setSpeechRecognitionLanguage(fromLanguage);
    translationConfig.addTargetLanguage(toLanguage);

    // See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
    translationConfig.setVoiceName("de-DE-Hedda");

    try (TranslationRecognizer recognizer = new TranslationRecognizer(translationConfig)) {
        recognizer.synthesizing.addEventListener((s, e) -> {
            byte[] audio = e.getResult().getAudio();
            int size = audio.length;
            System.out.println("Audio synthesized: " + size + " byte(s)" + (size == 0 ? "(COMPLETE)" : ""));

            if (size > 0) {
                try (FileOutputStream file = new FileOutputStream("translation.wav")) {
                    file.write(audio);
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
        });

        System.out.printf("Say something in '%s' and we'll translate...", fromLanguage);

        TranslationRecognitionResult result = recognizer.recognizeOnceAsync().get();
        if (result.getReason() == ResultReason.TranslatedSpeech) {
            System.out.printf("Recognized: \"%s\"\n", result.getText());
            for (Map.Entry<String, String> pair : result.getTranslations().entrySet()) {
                String language = pair.getKey();
                String translation = pair.getValue();
                System.out.printf("Translated into '%s': %s\n", language, translation);
            }
        }
    }
}
```

Manual synthesis

The `getTranslations` function returns a dictionary that can be used to synthesize audio from the translation text. Iterate through each translation, and synthesize the translation. When creating a `SpeechSynthesizer` instance, the `SpeechConfig` object needs to have its `setSpeechSynthesisVoiceName` property set to the desired voice. The following example translates to five languages, and each translation is then synthesized to an audio file in the corresponding neural language.

```
static void translateSpeech() throws ExecutionException, InterruptedException {
    SpeechTranslationConfig translationConfig = SpeechTranslationConfig.fromSubscription(
        SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);

    String fromLanguage = "en-US";
    String[] toLanguages = { "de", "en", "it", "pt", "zh-Hans" };
    translationConfig.setSpeechRecognitionLanguage(fromLanguage);
    for (String language : toLanguages) {
        translationConfig.addTargetLanguage(language);
    }

    try (TranslationRecognizer recognizer = new TranslationRecognizer(translationConfig)) {
        System.out.printf("Say something in '%s' and we'll translate...", fromLanguage);

        TranslationRecognitionResult result = recognizer.recognizeOnceAsync().get();
        if (result.getReason() == ResultReason.TranslatedSpeech) {
            // See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
            Map<String, String> languageToVoiceMap = new HashMap<String, String>();
            languageToVoiceMap.put("de", "de-DE-KatjaNeural");
            languageToVoiceMap.put("en", "en-US-AriaNeural");
            languageToVoiceMap.put("it", "it-IT-ElsaNeural");
            languageToVoiceMap.put("pt", "pt-BR-FranciscaNeural");
            languageToVoiceMap.put("zh-Hans", "zh-CN-XiaoxiaoNeural");

            System.out.printf("Recognized: \"%s\"\n", result.getText());
            for (Map.Entry<String, String> pair : result.getTranslations().entrySet()) {
                String language = pair.getKey();
                String translation = pair.getValue();
                System.out.printf("Translated into '%s': %s\n", language, translation);

                SpeechConfig speechConfig =
                    SpeechConfig.fromSubscription(SPEECH__SUBSCRIPTION__KEY, SPEECH__SERVICE__REGION);
                speechConfig.setSpeechSynthesisVoiceName(languageToVoiceMap.get(language));

                AudioConfig audioConfig = AudioConfig.fromWavFileOutput(language + "-translation.wav");
                try (SpeechSynthesizer synthesizer = new SpeechSynthesizer(speechConfig, audioConfig)) {
                    synthesizer.SpeakTextAsync(translation).get();
                }
            }
        }
    }
}
```

For more information about speech synthesis, see the basics of speech synthesis.

One of the core features of the Speech service is the ability to recognize human speech and translate it to other languages. In this quickstart you learn how to use the Speech SDK in your apps and products to perform high-quality speech translation. This quickstart covers topics including:

- Translating speech-to-text
- Translating speech to multiple target languages
- Performing direct speech-to-speech translation

Skip to samples on GitHub

If you want to skip straight to sample code, see the JavaScript quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK for JavaScript . Depending on your platform, use the following instructions:

- Node.js
- Web browser

Additionally, depending on the target environment use one of the following:

- script
- import
- require

Download and extract the Speech SDK for JavaScript *microsoft.cognitiveservices.speech.sdk.bundle.js* file, and place it in a folder accessible to your HTML file.

```
<script src="microsoft.cognitiveservices.speech.sdk.bundle.js"></script>;
```

> **TIP**
> If you're targeting a web browser, and using the `<script>` tag; the `sdk` prefix is not needed. The `sdk` prefix is an alias used to name the `require` module.

Create a translation configuration

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

> **NOTE**
> Regardless of whether you're performing speech recognition, speech synthesis, translation, or intent recognition, you'll always create a configuration. There are a few ways that you can initialize a `SpeechTranslationConfig`:

- 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 take a look at how a `SpeechTranslationConfig` is created using a key and region. Get these credentials by following steps in Try the Speech service for free.

```
const speechTranslationConfig = SpeechTranslationConfig.fromSubscription("YourSubscriptionKey",
"YourServiceRegion");
```

Initialize a translator

After you've created a `SpeechTranslationConfig`, the next step is to initialize a `TranslationRecognizer`. When you initialize a `TranslationRecognizer`, you'll need to pass it your `speechTranslationConfig`. This provides the credentials that the translation service requires to validate your request.

If you're translating speech provided through your device's default microphone, here's what the `TranslationRecognizer` should look like:

```
const translator = new TranslationRecognizer(speechTranslationConfig);
```

If you want to specify the audio input device, then you'll need to create an `AudioConfig` and provide the `audioConfig` parameter when initializing your `TranslationRecognizer`.

> **TIP**
> Learn how to get the device ID for your audio input device. Reference the `AudioConfig` object as follows:

```
const audioConfig = AudioConfig.fromDefaultMicrophoneInput();
const recognizer = new TranslationRecognizer(speechTranslationConfig, audioConfig);
```

If you want to provide an audio file instead of using a microphone, you'll still need to provide an `audioConfig`. However, this can only be done when targeting **Node.js** and when you create an `AudioConfig`, instead of calling `fromDefaultMicrophoneInput`, you'll call `fromWavFileOutput` and pass the `filename` parameter.

```
const audioConfig = AudioConfig.fromWavFileInput("YourAudioFile.wav");
const recognizer = new TranslationRecognizer(speechTranslationConfig, audioConfig);
```

Translate speech

The TranslationRecognizer class for the Speech SDK for JavaScript exposes a few methods that you can use for speech translation.

- At-start translation (async) - Performs translation in a non-blocking (asynchronous) mode. This will translate 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.
- Continuous translation (async) - Asynchronously initiates continuous translation operation. The user registers to events and handles various application states. To stop asynchronous continuous translation, call `stopContinuousRecognitionAsync`.

> **NOTE**
> Learn more about how to choose a speech recognition mode.

Specify a target language

To translate, you must specify both a source language and at least one target language. You can choose a source language using a locale listed in the Speech translation table. Find your options for translated language at the same link. Your options for target languages differ when you want to view text, or want to hear synthesized translated speech. To translate from English to German, modify the translation config object:

```
speechTranslationConfig.speechRecognitionLanguage = "en-US";
speechTranslationConfig.addTargetLanguage("de");
```

At-start recognition

Here's an example of asynchronous at-start translation using `recognizeOnceAsync`:

```
recognizer.recognizeOnceAsync(result => {
    // Interact with result
});
```

You'll need to write some code to handle the result. This sample evaluates the `result.reason` for a translation to German:

```
recognizer.recognizeOnceAsync(
  function (result) {
    let translation = result.translations.get("de");
    window.console.log(translation);
    recognizer.close();
  },
  function (err) {
    window.console.log(err);
    recognizer.close();
});
```

Your code can also handle updates provided while the translation is processing. You can use these updates to provide visual feedback about the translation progress. See this JavaScript Node.js example for sample code that shows updates provided during the translation process. The following code also displays details produced during the translation process.

```
recognizer.recognizing = function (s, e) {
    var str = ("(recognizing) Reason: " + SpeechSDK.ResultReason[e.result.reason] +
            " Text: " +  e.result.text +
            " Translation:");
    str += e.result.translations.get("de");
    console.log(str);
};
recognizer.recognized = function (s, e) {
    var str = "\r\n(recognized)  Reason: " + SpeechSDK.ResultReason[e.result.reason] +
            " Text: " + e.result.text +
            " Translation:";
    str += e.result.translations.get("de");
    str += "\r\n";
    console.log(str);
};
```

Continuous translation

Continuous translation is a bit more involved than at-start recognition. It requires you to subscribe to the `recognizing`, `recognized`, and `canceled` events to get the recognition results. To stop translation, you must call `stopContinuousRecognitionAsync`. Here's an example of how continuous translation is performed on an audio input file.

Let's start by defining the input and initializing a `TranslationRecognizer`:

```
const translator = new TranslationRecognizer(speechTranslationConfig);
```

We'll subscribe to the events sent from the `TranslationRecognizer`.

- `recognizing`: Signal for events containing intermediate translation results.
- `recognized`: Signal for events containing final translation results (indicating a successful translation attempt).
- `sessionStopped`: Signal for events indicating the end of a translation session (operation).
- `canceled`: Signal for events containing canceled translation results (indicating a translation attempt that was canceled as a result or a direct cancellation request or, alternatively, a transport or protocol failure).

```
recognizer.recognizing = (s, e) => {
    console.log(`TRANSLATING: Text=${e.result.text}`);
};
recognizer.recognized = (s, e) => {
    if (e.result.reason == ResultReason.RecognizedSpeech) {
        console.log(`TRANSLATED: Text=${e.result.text}`);
    }
    else if (e.result.reason == ResultReason.NoMatch) {
        console.log("NOMATCH: Speech could not be translated.");
    }
};
recognizer.canceled = (s, e) => {
    console.log(`CANCELED: Reason=${e.reason}`);
    if (e.reason == CancellationReason.Error) {
        console.log(`"CANCELED: ErrorCode=${e.errorCode}`);
        console.log(`"CANCELED: ErrorDetails=${e.errorDetails}`);
        console.log("CANCELED: Did you update the subscription info?");
    }
    recognizer.stopContinuousRecognitionAsync();
};
recognizer.sessionStopped = (s, e) => {
    console.log("\n    Session stopped event.");
    recognizer.stopContinuousRecognitionAsync();
};
```

With everything set up, we can call `startContinuousRecognitionAsync`.

```
// Starts continuous recognition. Uses stopContinuousRecognitionAsync() to stop recognition.
recognizer.startContinuousRecognitionAsync();
// Something later can call, stops recognition.
// recognizer.StopContinuousRecognitionAsync();
```

Choose a source language

A common task for speech translation is specifying the input (or source) language. Let's take a look at how you would change the input language to Italian. In your code, find your `SpeechTranslationConfig`, then add the following line directly below it.

```
speechTranslationConfig.speechRecognitionLanguage = "it-IT";
```

The `speechRecognitionLanguage` property expects a language-locale format string. You can provide any value in the **Locale** column in the list of supported locales/languages.

Choose one or more target languages

The Speech SDK can translate to multiple target languages in parallel. The target languages available are somewhat different from the source language list, and you specify target languages using a language code, rather than a locale. See a list of language codes for text targets in the speech translation table on the language support page. You can also find details about translation into synthesized languages there.

The following code adds German as a target language:

```
translationConfig.addTargetLanguage("de");
```

Since multiple target language translations are possible, your code must specify the target language when examining the result. The following code gets translation results for German.

```
recognizer.recognized = function (s, e) {
    var str = "\r\n(recognized)  Reason: " +
            sdk.ResultReason[e.result.reason] +
            " Text: " + e.result.text + " Translations:";
    var language = "de";
    str += " [" + language + "] " + e.result.translations.get(language);
    str += "\r\n";
    // show str somewhere
};
```

One of the core features of the Speech service is the ability to recognize human speech and translate it to other languages. In this quickstart you learn how to use the Speech SDK in your apps and products to perform high-quality speech translation. This quickstart covers topics including:

- Translating speech-to-text
- Translating speech to multiple target languages
- Performing direct speech-to-speech translation

Skip to samples on GitHub

If you want to skip straight to sample code, see the Python quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, follow the instructions under the Get the Speech SDK section of the *About the Speech SDK* article.

Import dependencies

To run the examples in this article, include the following `import` statements at the top of the python code file.

```
import os
import azure.cognitiveservices.speech as speechsdk
```

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 subscription key and region. The python code file contains two values that are assigned from the host machines environment variables, namely `SPEECH__SUBSCRIPTION__KEY` and `SPEECH__SERVICE__REGION`. Both of these variables are at the global scope, making them accessible within function definition of the code file. For more information on environment variables, see environment variables and application configuration.

```
speech_key, service_region = os.environ['SPEECH__SUBSCRIPTION__KEY'], os.environ['SPEECH__SERVICE__REGION']
```

Create a speech translation configuration

To call the Speech service using the Speech SDK, you need to create a `SpeechTranslationConfig`. 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.

There are a few ways that you can initialize a `SpeechTranslationConfig`:

- 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 take a look at how a `SpeechTranslationConfig` is created using a key and region. Get these credentials by following steps in Try the Speech service for free.

```
from_language, to_language = 'en-US', 'de'

def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)
```

Change source language

One common task of speech translation is specifying the input (or source) language. Let's take a look at how you would change the input language to Italian. In your code, interact with the `SpeechTranslationConfig` instance, assigning to the `speech_recognition_language` property.

```
def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    # Source (input) language
    from_language = "it-IT"
    translation_config.speech_recognition_language = from_language
```

The `speech_recognition_language` property expects a language-locale format string. You can provide any value in the **Locale** column in the list of supported locales/languages.

Add 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.

```
def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = "it-IT"

    # Translate to languages. See, https://aka.ms/speech/sttt-languages
    translation_config.add_target_language("fr")
    translation_config.add_target_language("de")
```

With every call to `add_target_language`, 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`, the next step is to initialize a `TranslationRecognizer`. When you initialize a `TranslationRecognizer`, you'll need to pass it your `translation_config`. The configuration object provides the credentials that the speech service requires to validate your request.

If you're recognizing speech using your device's default microphone, here's what the `TranslationRecognizer` should look like:

```
def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = from_language
    translation_config.add_target_language(to_language)

    recognizer = speechsdk.translation.TranslationRecognizer(
            translation_config=translation_config)
```

If you want to specify the audio input device, then you'll need to create an `AudioConfig` and provide the `audio_config` parameter when initializing your `TranslationRecognizer`.

> **TIP**
>
> Learn how to get the device ID for your audio input device.

First, you'll reference the `AudioConfig` object as follows:

```
def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = from_language
    for lang in to_languages:
        translation_config.add_target_language(lang)

    audio_config = speechsdk.audio.AudioConfig(use_default_microphone=True)
    recognizer = speechsdk.translation.TranslationRecognizer(
            translation_config=translation_config, audio_config=audio_config)
```

If you want to provide an audio file instead of using a microphone, you'll still need to provide an `audioConfig`. However, when you create an `AudioConfig`, instead of calling with `use_default_microphone=True`, you'll call with `filename="path-to-file.wav"` and provide the `filename` parameter.

```
def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = from_language
    for lang in to_languages:
        translation_config.add_target_language(lang)

    audio_config = speechsdk.audio.AudioConfig(filename="path-to-file.wav")
    recognizer = speechsdk.translation.TranslationRecognizer(
            translation_config=translation_config, audio_config=audio_config)
```

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.

```
import os
import azure.cognitiveservices.speech as speechsdk

speech_key, service_region = os.environ['SPEECH__SERVICE__KEY'], os.environ['SPEECH__SERVICE__REGION']
from_language, to_languages = 'en-US', 'de'

def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = from_language
    translation_config.add_target_language(to_language)

    recognizer = speechsdk.translation.TranslationRecognizer(
            translation_config=translation_config)

    print('Say something...')
    result = recognizer.recognize_once()
    print(get_result_text(reason=result.reason, result=result))

def get_result_text(reason, result):
    reason_format = {
        speechsdk.ResultReason.TranslatedSpeech:
            f'RECOGNIZED "{from_language}": {result.text}\n' +
            f'TRANSLATED into "{to_language}"": {result.translations[to_language]}',
        speechsdk.ResultReason.RecognizedSpeech: f'Recognized: "{result.text}"',
        speechsdk.ResultReason.NoMatch: f'No speech could be recognized: {result.no_match_details}',
        speechsdk.ResultReason.Canceled: f'Speech Recognition canceled: {result.cancellation_details}'
    }
    return reason_format.get(reason, 'Unable to recognize speech')

translate_speech_to_text()
```

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, 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 `voice_name` and provide an event handler for the `Synthesizing` event, get the audio. The following example saves the translated audio as a *.wav* file.

> **IMPORTANT**
> The event-based synthesis only works with a single translation, **do not** add multiple target translation languages. Additionally, the `voice_name` should be the same language as the target translation language, for example; `"de"` could map to `"de-DE-Hedda"`.

```
import os
import azure.cognitiveservices.speech as speechsdk

speech_key, service_region = os.environ['SPEECH__SERVICE__KEY'], os.environ['SPEECH__SERVICE__REGION']
from_language, to_language = 'en-US', 'de'

def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = from_language
    translation_config.add_target_language(to_language)

    # See: https://aka.ms/speech/sdkregion#standard-and-neural-voices
    translation_config.voice_name = "de-DE-Hedda"

    recognizer = speechsdk.translation.TranslationRecognizer(
            translation_config=translation_config)

    def synthesis_callback(evt):
        size = len(evt.result.audio)
        print(f'Audio synthesized: {size} byte(s) {"(COMPLETED)" if size == 0 else ""}')

        if size > 0:
            file = open('translation.wav', 'wb+')
            file.write(evt.result.audio)
            file.close()

    recognizer.synthesizing.connect(synthesis_callback)

    print(f'Say something in "{from_language}" and we\'ll translate into "{to_language}".')

    result = recognizer.recognize_once()
    print(get_result_text(reason=result.reason, result=result))

def get_result_text(reason, result):
    reason_format = {
        speechsdk.ResultReason.TranslatedSpeech:
            f'Recognized "{from_language}": {result.text}\n' +
            f'Translated into "{to_language}"": {result.translations[to_language]}',
        speechsdk.ResultReason.RecognizedSpeech: f'Recognized: "{result.text}"',
        speechsdk.ResultReason.NoMatch: f'No speech could be recognized: {result.no_match_details}',
        speechsdk.ResultReason.Canceled: f'Speech Recognition canceled: {result.cancellation_details}'
    }
    return reason_format.get(reason, 'Unable to recognize speech')

translate_speech_to_text()
```

Manual synthesis

The `translations` dictionary can be used to synthesize audio from the translation text. Iterate through each

translation, and synthesize the translation. When creating a `SpeechSynthesizer` instance, the `SpeechConfig` object needs to have its `speech_synthesis_voice_name` property set to the desired voice. The following example translates to five languages, and each translation is then synthesized to an audio file in the corresponding neural language.

```
import os
import azure.cognitiveservices.speech as speechsdk

speech_key, service_region = os.environ['SPEECH__SERVICE__KEY'], os.environ['SPEECH__SERVICE__REGION']
from_language, to_languages = 'en-US', [ 'de', 'en', 'it', 'pt', 'zh-Hans' ]

def translate_speech_to_text():
    translation_config = speechsdk.translation.SpeechTranslationConfig(
            subscription=speech_key, region=service_region)

    translation_config.speech_recognition_language = from_language
    for lang in to_languages:
        translation_config.add_target_language(lang)

    recognizer = speechsdk.translation.TranslationRecognizer(
            translation_config=translation_config)

    print('Say something...')
    result = recognizer.recognize_once()
    synthesize_translations(result=result)

def synthesize_translations(result):
    language_to_voice_map = {
        "de": "de-DE-KatjaNeural",
        "en": "en-US-AriaNeural",
        "it": "it-IT-ElsaNeural",
        "pt": "pt-BR-FranciscaNeural",
        "zh-Hans": "zh-CN-XiaoxiaoNeural"
    }
    print(f'Recognized: "{result.text}"')

    for language in result.translations:
        translation = result.translations[language]
        print(f'Translated into "{language}": {translation}')

        speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)
        speech_config.speech_synthesis_voice_name = language_to_voice_map.get(language)

        audio_config = speechsdk.audio.AudioOutputConfig(filename=f'{language}-translation.wav')
        speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config,
audio_config=audio_config)
        speech_synthesizer.speak_text_async(translation).get()

translate_speech_to_text()
```

For more information about speech synthesis, see the basics of speech synthesis.

One of the core features of the Speech service is the ability to recognize human speech and translate it to other languages. In this quickstart you learn how to use the Speech SDK in your apps and products to perform high-quality speech translation. This quickstart translates speech from the microphone into text in another language.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Download and install

- Windows Install
- Linux Install
- Docker Install (Windows, Linux, macOS)

Follow these steps to install the Speech CLI on Windows:

1. On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
2. Install .NET Core 3.1 SDK.
3. Install the Speech CLI using NuGet by entering this command:

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

Type `spx` to see help for the Speech CLI.

> **NOTE**
>
> As an alternative to NuGet, you can download and extract the Speech CLI for Windows as a zip file.

Font limitations

On Windows, the Speech CLI can only show fonts available to the command prompt on the local computer. Windows Terminal supports all fonts produced interactively by the Speech CLI.

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

Set source and target language

This command calls Speech CLI to translate speech from the microphone from Italian to French.

```
spx translate --microphone --source it-IT --target fr
```

Next steps

- Use codec compressed audio formats
- See the quickstart samples on GitHub.

Tutorial: Build a Flask app with Azure Cognitive Services

11/23/2021 • 24 minutes to read • Edit Online

In this tutorial, you'll build a Flask web app that uses Azure Cognitive Services to translate text, analyze sentiment, and synthesize translated text into speech. Our focus is on the Python code and Flask routes that enable our application, however, we will help you out with the HTML and JavaScript that pulls the app together. If you run into any issues let us know using the feedback button below.

Here's what this tutorial covers:

- Get Azure subscription keys
- Set up your development environment and install dependencies
- Create a Flask app
- Use the Translator to translate text
- Use the Language service to analyze positive/negative sentiment of input text and translations
- Use the Speech Service to convert translated text into synthesized speech
- Run your Flask app locally

> **TIP**
> If you'd like to skip ahead and see all the code at once, the entire sample, along with build instructions are available on GitHub.

What is Flask?

Flask is a microframework for creating web applications. This means Flask provides you with tools, libraries, and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as substantive as a web-based calendar application or a commercial website.

For those of you who want to deep dive after this tutorial here are a few helpful links:

- Flask documentation
- Flask for Dummies - A Beginner's Guide to Flask

Prerequisites

Let's review the software and subscription keys that you'll need for this tutorial.

- Python 3.6 or later
- Git tools
- An IDE or text editor, such as Visual Studio Code or Atom
- Chrome or Firefox
- A **Translator** subscription key (you can likely use the **global** location.)
- A **Language service** subscription key in the **West US** region.
- A **Speech Services** subscription key in the **West US** region.

Create an account and subscribe to resources

As previously mentioned, you're going to need three subscription keys for this tutorial. This means that you need to create a resource within your Azure account for:

- Translator
- Language service
- Speech Services

Use Create a Cognitive Services Account in the Azure portal for step-by-step instructions to create resources.

> **IMPORTANT**
> For this tutorial, please create your resources in the West US region. If using a different region, you'll need to adjust the base URL in each of your Python files.

Set up your dev environment

Before you build your Flask web app, you'll need to create a working directory for your project and install a few Python packages.

Create a working directory

1. Open command line (Windows) or terminal (macOS/Linux). Then, create a working directory and sub directories for your project:

```
mkdir -p flask-cog-services/static/scripts && mkdir flask-cog-services/templates
```

2. Change to your project's working directory:

```
cd flask-cog-services
```

Create and activate your virtual environment with `virtualenv`

Let's create a virtual environment for our Flask app using `virtualenv`. Using a virtual environment ensures that you have a clean environment to work from.

1. In your working directory, run this command to create a virtual environment: **macOS/Linux:**

```
virtualenv venv --python=python3
```

We've explicitly declared that the virtual environment should use Python 3. This ensures that users with multiple Python installations are using the correct version.

Windows CMD / Windows Bash:

```
virtualenv venv
```

To keep things simple, we're naming your virtual environment venv.

2. The commands to activate your virtual environment will vary depending on your platform/shell:

PLATFORM	SHELL	COMMAND
macOS/Linux	bash/zsh	`source venv/bin/activate`

PLATFORM	SHELL	COMMAND
Windows	bash	`source venv/Scripts/activate`
	Command Line	`venv\Scripts\activate.bat`
	PowerShell	`venv\Scripts\Activate.ps1`

After running this command, your command line or terminal session should be prefaced with `venv`.

3. You can deactivate the session at any time by typing this into the command line or terminal: `deactivate`.

> **NOTE**
> Python has extensive documentation for creating and managing virtual environments, see virtualenv.

Install requests

Requests is a popular module that is used to send HTTP 1.1 requests. There's no need to manually add query strings to your URLs, or to form-encode your POST data.

1. To install requests, run:

```
pip install requests
```

> **NOTE**
> If you'd like to learn more about requests, see Requests: HTTP for Humans.

Install and configure Flask

Next we need to install Flask. Flask handles the routing for our web app, and allows us to make server-to-server calls that hide our subscription keys from the end user.

1. To install Flask, run:

```
pip install Flask
```

Let's make sure Flask was installed. Run:

```
flask --version
```

The version should be printed to terminal. Anything else means something went wrong.

2. To run the Flask app, you can either use the flask command or Python's -m switch with Flask. Before you can do that you need to tell your terminal which app to work with by exporting the `FLASK_APP` environment variable:

macOS/Linux:

```
export FLASK_APP=app.py
```

Windows:

```
set FLASK_APP=app.py
```

Create your Flask app

In this section, you're going to create a barebones Flask app that returns an HTML file when users hit the root of your app. Don't spend too much time trying to pick apart the code, we'll come back to update this file later.

What is a Flask route?

Let's take a minute to talk about "routes". Routing is used to bind a URL to a specific function. Flask uses route decorators to register functions to specific URLs. For example, when a user navigates to the root (`/`) of our web app, `index.html` is rendered.

```
@app.route('/')
def index():
    return render_template('index.html')
```

Let's take a look at one more example to hammer this home.

```
@app.route('/about')
def about():
    return render_template('about.html')
```

This code ensures that when a user navigates to `http://your-web-app.com/about` that the `about.html` file is rendered.

While these samples illustrate how to render html pages for a user, routes can also be used to call APIs when a button is pressed, or take any number of actions without having to navigate away from the homepage. You'll see this in action when you create routes for translation, sentiment, and speech synthesis.

Get started

1. Open the project in your IDE, then create a file named `app.py` in the root of your working directory. Next, copy this code into `app.py` and save:

   ```
   from flask import Flask, render_template, url_for, jsonify, request

   app = Flask(__name__)
   app.config['JSON_AS_ASCII'] = False

   @app.route('/')
   def index():
       return render_template('index.html')
   ```

   This code block tells the app to display `index.html` whenever a user navigates to the root of your web app (`/`).

2. Next, let's create the front-end for our web app. Create a file named `index.html` in the `templates` directory. Then copy this code into `templates/index.html`.

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required metadata tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="Translate and analyze text with Azure Cognitive Services.">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
    <title>Translate and analyze text with Azure Cognitive Services</title>
  </head>
  <body>
    <div class="container">
      <h1>Translate, synthesize, and analyze text with Azure</h1>
      <p>This simple web app uses Azure for text translation, text-to-speech conversion, and
sentiment analysis of input text and translations. Learn more about <a
href="https://docs.microsoft.com/azure/cognitive-services/">Azure Cognitive Services</a>.
     </p>
     <!-- HTML provided in the following sections goes here. -->

     <!-- End -->
    </div>

    <!-- Required Javascript for this tutorial -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
    <script type = "text/javascript" src ="static/scripts/main.js"></script>
  </body>
</html>
```

3. Let's test the Flask app. From the terminal, run:

```
flask run
```

4. Open a browser and navigate to the URL provided. You should see your single page app. Press **Ctrl + C** to kill the app.

Translate text

Now that you have an idea of how a simple Flask app works, let's:

- Write some Python to call the Translator and return a response
- Create a Flask route to call your Python code
- Update the HTML with an area for text input and translation, a language selector, and translate button
- Write JavaScript that allows users to interact with your Flask app from the HTML

Call the Translator

The first thing you need to do is write a function to call the Translator. This function will take two arguments: `text_input` and `language_output`. This function is called whenever a user presses the translate button in your app. The text area in the HTML is sent as the `text_input`, and the language selection value in the HTML is sent as `language_output`.

1. Let's start by creating a file called `translate.py` in the root of your working directory.
2. Next, add this code to `translate.py`. This function takes two arguments: `text_input` and `language_output`.

```
import os, requests, uuid, json

# Don't forget to replace with your Cog Services subscription key!
# If you prefer to use environment variables, see Extra Credit for more info.
subscription_key = 'YOUR_TRANSLATOR_TEXT_SUBSCRIPTION_KEY'
location = 'YOUR_TRANSLATOR_RESOURCE_LOCATION'
# Don't forget to replace with your Cog Services location!
# Our Flask route will supply two arguments: text_input and language_output.
# When the translate text button is pressed in our Flask app, the Ajax request
# will grab these values from our web app, and use them in the request.
# See main.js for Ajax calls.
def get_translation(text_input, language_output):
    base_url = 'https://api.cognitive.microsofttranslator.com'
    path = '/translate?api-version=3.0'
    params = '&to=' + language_output
    constructed_url = base_url + path + params

    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key,
        'Ocp-Apim-Subscription-Region': location,
        'Content-type': 'application/json',
        'X-ClientTraceId': str(uuid.uuid4())
    }

    # You can pass more than one object in body.
    body = [{
        'text' : text_input
    }]
    response = requests.post(constructed_url, headers=headers, json=body)
    return response.json()
```

3. Add your Translator subscription key and save.

Add a route to `app.py`

Next, you'll need to create a route in your Flask app that calls `translate.py`. This route will be called each time a user presses the translate button in your app.

For this app, your route is going to accept `POST` requests. This is because the function expects the text to translate and an output language for the translation.

Flask provides helper functions to help you parse and manage each request. In the code provided, `get_json()` returns the data from the `POST` request as JSON. Then using `data['text']` and `data['to']`, the text and output language values are passed to `get_translation()` function available from `translate.py`. The last step is to return the response as JSON, since you'll need to display this data in your web app.

In the following sections, you'll repeat this process as you create routes for sentiment analysis and speech synthesis.

1. Open `app.py` and locate the import statement at the top of `app.py` and add the following line:

```
import translate
```

   Now our Flask app can use the method available via `translate.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/translate-text', methods=['POST'])
def translate_text():
    data = request.get_json()
    text_input = data['text']
    translation_output = data['to']
    response = translate.get_translation(text_input, translation_output)
    return jsonify(response)
```

Update `index.html`

Now that you have a function to translate text, and a route in your Flask app to call it, the next step is to start building the HTML for your app. The HTML below does a few things:

- Provides a text area where users can input text to translate.
- Includes a language selector.
- Includes HTML elements to render the detected language and confidence scores returned during translation.
- Provides a read-only text area where the translation output is displayed.
- Includes placeholders for sentiment analysis and speech synthesis code that you'll add to this file later in the tutorial.

Let's update `index.html`.

1. Open `index.html` and locate these code comments:

```
<!-- HTML provided in the following sections goes here. -->

<!-- End -->
```

2. Replace the code comments with this HTML block:

```
<div class="row">
  <div class="col">
    <form>
      <!-- Enter text to translate. -->
      <div class="form-group">
        <label for="text-to-translate"><strong>Enter the text you'd like to translate:</strong>
</label>
        <textarea class="form-control" id="text-to-translate" rows="5"></textarea>
      </div>
      <!-- Select output language. -->
      <div class="form-group">
        <label for="select-language"><strong>Translate to:</strong></label>
        <select class="form-control" id="select-language">
          <option value="ar">Arabic</option>
          <option value="ca">Catalan</option>
          <option value="zh-Hans">Chinese (Simplified)</option>
          <option value="zh-Hant">Chinese (Traditional)</option>
          <option value="hr">Croatian</option>
          <option value="en">English</option>
          <option value="fr">French</option>
          <option value="de">German</option>
          <option value="el">Greek</option>
          <option value="he">Hebrew</option>
          <option value="hi">Hindi</option>
          <option value="it">Italian</option>
          <option value="ja">Japanese</option>
          <option value="ko">Korean</option>
          <option value="pt">Portuguese</option>
          <option value="ru">Russian</option>
          <option value="es">Spanish</option>
          <option value="th">Thai</option>
```

```
            <option value="tr">Turkish</option>
            <option value="vi">Vietnamese</option>
          </select>
        </div>
        <button type="submit" class="btn btn-primary mb-2" id="translate">Translate text</button></br>
        <div id="detected-language" style="display: none">
          <strong>Detected language:</strong> <span id="detected-language-result"></span><br />
          <strong>Detection confidence:</strong> <span id="confidence"></span><br /><br />
        </div>

        <!-- Start sentiment code-->

        <!-- End sentiment code -->

      </form>
    </div>
    <div class="col">
      <!-- Translated text returned by the Translate API is rendered here. -->
      <form>
        <div class="form-group" id="translator-text-response">
          <label for="translation-result"><strong>Translated text:</strong></label>
          <textarea readonly class="form-control" id="translation-result" rows="5"></textarea>
        </div>

        <!-- Start voice font selection code -->

        <!-- End voice font selection code -->

      </form>

      <!-- Add Speech Synthesis button and audio element -->

      <!-- End Speech Synthesis button -->

    </div>
  </div>
```

The next step is to write some JavaScript. This is the bridge between your HTML and Flask route.

Create `main.js`

The `main.js` file is the bridge between your HTML and Flask route. Your app will use a combination of jQuery, Ajax, and XMLHttpRequest to render content, and make `POST` requests to your Flask routes.

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the contents of the text area and the language selector are assigned to variables, and then passed along in the request to `translate-text`.

The code then iterates through the response, and updates the HTML with the translation, detected language, and confidence score.

1. From your IDE, create a file named `main.js` in the `static/scripts` directory.
2. Copy this code into `static/scripts/main.js`:

```
//Initiate jQuery on load.
$(function() {
  //Translate text with flask route
  $("#translate").on("click", function(e) {
    e.preventDefault();
    var translateVal = document.getElementById("text-to-translate").value;
    var languageVal = document.getElementById("select-language").value;
    var translateRequest = { 'text': translateVal, 'to': languageVal }

    if (translateVal !== "") {
      $.ajax({
        url: '/translate-text',
        method: 'POST',
        headers: {
            'Content-Type':'application/json'
        },
        dataType: 'json',
        data: JSON.stringify(translateRequest),
        success: function(data) {
          for (var i = 0; i < data.length; i++) {
            document.getElementById("translation-result").textContent = data[i].translations[0].text;
            document.getElementById("detected-language-result").textContent =
data[i].detectedLanguage.language;
            if (document.getElementById("detected-language-result").textContent !== ""){
              document.getElementById("detected-language").style.display = "block";
            }
            document.getElementById("confidence").textContent = data[i].detectedLanguage.score;
          }
        }
      });
    };
  });
  // In the following sections, you'll add code for sentiment analysis and
  // speech synthesis here.
})
```

Test translation

Let's test translation in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. If it doesn't work, make sure that you've added your subscription key.

> **TIP**
> If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

Press **CTRL + c** to kill the app, then head to the next section.

Analyze sentiment

The Language service API can be used to perform sentiment analysis, extract key phrases from text, or detect the source language. In this app, we're going to use sentiment analysis to determine if the provided text is positive, neutral, or negative. The API returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment, and scores close to 0 indicate negative sentiment.

In this section, you're going to do a few things:

- Write some Python to call the Langauge service API to perform sentiment analysis and return a response
- Create a Flask route to call your Python code
- Update the HTML with an area for sentiment scores, and a button to perform analysis
- Write JavaScript that allows users to interact with your Flask app from the HTML

Call the Language service API

Let's write a function to call the Language service API. This function will take four arguments: `input_text`, `input_language`, `output_text`, and `output_language`. This function is called whenever a user presses the run sentiment analysis button in your app. Data provided by the user from the text area and language selector, as well as the detected language and translation output are provided with each request. The response object includes sentiment scores for the source and translation. In the following sections, you're going to write some JavaScript to parse the response and use it in your app. For now, let's focus on call the Language service API.

1. Let's create a file called `sentiment.py` in the root of your working directory.
2. Next, add this code to `sentiment.py`.

```
import os, requests, uuid, json

# Don't forget to replace with your Cog Services subscription key!
subscription_key = 'YOUR_TEXT_ANALYTICS_SUBSCRIPTION_KEY'
endpoint = "YOUR_TEXT_ANALYTICS_ENDPOINT"
# Our Flask route will supply four arguments: input_text, input_language,
# output_text, output_language.
# When the run sentiment analysis button is pressed in our Flask app,
# the Ajax request will grab these values from our web app, and use them
# in the request. See main.js for Ajax calls.

def get_sentiment(input_text, input_language):
    path = '/text/analytics/v3.0/sentiment'
    constructed_url = endpoint + path

    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key,
        'Content-type': 'application/json',
        'X-ClientTraceId': str(uuid.uuid4())
    }

    # You can pass more than one object in body.
    body = {
        'documents': [
            {
                'language': input_language,
                'id': '1',
                'text': input_text
            },
        ]
    }
    response = requests.post(constructed_url, headers=headers, json=body)
    return response.json()
```

3. Add your Language service subscription key and save.

Add a route to `app.py`

Let's create a route in your Flask app that calls `sentiment.py`. This route will be called each time a user presses the run sentiment analysis button in your app. Like the route for translation, this route is going to accept `POST` requests since the function expects arguments.

1. Open `app.py` and locate the import statement at the top of `app.py` and update it:

```
import translate, sentiment
```

Now our Flask app can use the method available via `sentiment.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/sentiment-analysis', methods=['POST'])
def sentiment_analysis():
    data = request.get_json()
    input_text = data['inputText']
    input_lang = data['inputLanguage']
    response = sentiment.get_sentiment(input_text, input_lang)
    return jsonify(response)
```

Update `index.html`

Now that you have a function to run sentiment analysis, and a route in your Flask app to call it, the next step is to start writing the HTML for your app. The HTML below does a few things:

- Adds a button to your app to run sentiment analysis
- Adds an element that explains sentiment scoring
- Adds an element to display the sentiment scores

1. Open `index.html` and locate these code comments:

```
<!-- Start sentiment code-->

<!-- End sentiment code -->
```

2. Replace the code comments with this HTML block:

```
<button type="submit" class="btn btn-primary mb-2" id="sentiment-analysis">Run sentiment
analysis</button></br>
<div id="sentiment" style="display: none">
   <p>Sentiment can be labeled as "positive", "negative", "neutral", or "mixed". </p>
   <strong>Sentiment label for input:</strong> <span id="input-sentiment"></span><br />
</div>
```

Update `main.js`

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the contents of the text area and the language selector are assigned to variables, and then passed along in the request to the `sentiment-analysis` route.

The code then iterates through the response, and updates the HTML with the sentiment scores.

1. From your IDE, create a file named `main.js` in the `static` directory.

2. Copy this code into `static/scripts/main.js`:

```
//Run sentiment analysis on input and translation.
$("#sentiment-analysis").on("click", function(e) {
  e.preventDefault();
  var inputText = document.getElementById("text-to-translate").value;
  var inputLanguage = document.getElementById("detected-language-result").innerHTML;
  var outputText = document.getElementById("translation-result").value;
  var outputLanguage = document.getElementById("select-language").value;

  var sentimentRequest = { "inputText": inputText, "inputLanguage": inputLanguage};

  if (inputText !== "") {
    $.ajax({
      url: "/sentiment-analysis",
      method: "POST",
      headers: {
          "Content-Type":"application/json"
      },
      dataType: "json",
      data: JSON.stringify(sentimentRequest),
      success: function(data) {
        for (var i = 0; i < data.documents.length; i++) {
          if (typeof data.documents[i] !== "undefined"){
            if (data.documents[i].id === "1") {
              document.getElementById("input-sentiment").textContent = data.documents[i].sentiment;
            }
          }
        }
        for (var i = 0; i < data.errors.length; i++) {
          if (typeof data.errors[i] !== "undefined"){
            if (data.errors[i].id === "1") {
              document.getElementById("input-sentiment").textContent = data.errors[i].message;
            }
          }
        }
        if (document.getElementById("input-sentiment").textContent !== ''){
          document.getElementById("sentiment").style.display = "block";
        }
      }
    });
  }
});
// In the next section, you'll add code for speech synthesis here.
```

Test sentiment analysis

Let's test sentiment analysis in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. Next, press the run sentiment analysis button. You should see two scores. If it doesn't work, make sure that you've added your subscription key.

> **TIP**
> If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

Press **CTRL + C** to kill the app, then head to the next section.

Convert text-to-speech

The Text-to-speech API enables your app to convert text into natural human-like synthesized speech. The service supports standard, neural, and custom voices. Our sample app uses a handful of the available voices, for a full list, see supported languages.

In this section, you're going to do a few things:

- Write some Python to convert text-to-speech with the Text-to-speech API
- Create a Flask route to call your Python code
- Update the HTML with a button to convert text-to-speech, and an element for audio playback
- Write JavaScript that allows users to interact with your Flask app

Call the Text-to-Speech API

Let's write a function to convert text-to-speech. This function will take two arguments: `input_text` and `voice_font`. This function is called whenever a user presses the convert text-to-speech button in your app. `input_text` is the translation output returned by the call to translate text, `voice_font` is the value from the voice font selector in the HTML.

1. Let's create a file called `synthesize.py` in the root of your working directory.

2. Next, add this code to `synthesize.py`.

```
import os, requests, time
from xml.etree import ElementTree

class TextToSpeech(object):
    def __init__(self, input_text, voice_font):
        subscription_key = 'YOUR_SPEECH_SERVICES_SUBSCRIPTION_KEY'
        self.subscription_key = subscription_key
        self.input_text = input_text
        self.voice_font = voice_font
        self.timestr = time.strftime('%Y%m%d-%H%M')
        self.access_token = None

    # This function performs the token exchange.
    def get_token(self):
        fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
        headers = {
            'Ocp-Apim-Subscription-Key': self.subscription_key
        }
        response = requests.post(fetch_token_url, headers=headers)
        self.access_token = str(response.text)

    # This function calls the TTS endpoint with the access token.
    def save_audio(self):
        base_url = 'https://westus.tts.speech.microsoft.com/'
        path = 'cognitiveservices/v1'
        constructed_url = base_url + path
        headers = {
            'Authorization': 'Bearer ' + self.access_token,
            'Content-Type': 'application/ssml+xml',
            'X-Microsoft-OutputFormat': 'riff-24khz-16bit-mono-pcm',
            'User-Agent': 'YOUR_RESOURCE_NAME',
        }
        # Build the SSML request with ElementTree
        xml_body = ElementTree.Element('speak', version='1.0')
        xml_body.set('{http://www.w3.org/XML/1998/namespace}lang', 'en-us')
        voice = ElementTree.SubElement(xml_body, 'voice')
        voice.set('{http://www.w3.org/XML/1998/namespace}lang', 'en-US')
        voice.set('name', 'Microsoft Server Speech Text to Speech Voice {}'.format(self.voice_font))
        voice.text = self.input_text
        # The body must be encoded as UTF-8 to handle non-ascii characters.
        body = ElementTree.tostring(xml_body, encoding="utf-8")

        #Send the request
        response = requests.post(constructed_url, headers=headers, data=body)

        # Write the response as a wav file for playback. The file is located
        # in the same directory where this sample is run.
        return response.content
```

3. Add your Speech Services subscription key and save.

Add a route to `app.py`

Let's create a route in your Flask app that calls `synthesize.py`. This route will be called each time a user presses the convert text-to-speech button in your app. Like the routes for translation and sentiment analysis, this route is going to accept `POST` requests since the function expects two arguments: the text to synthesize, and the voice font for playback.

1. Open `app.py` and locate the import statement at the top of `app.py` and update it:

```
import translate, sentiment, synthesize
```

Now our Flask app can use the method available via `synthesize.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/text-to-speech', methods=['POST'])
def text_to_speech():
    data = request.get_json()
    text_input = data['text']
    voice_font = data['voice']
    tts = synthesize.TextToSpeech(text_input, voice_font)
    tts.get_token()
    audio_response = tts.save_audio()
    return audio_response
```

Update `index.html`

Now that you have a function to convert text-to-speech, and a route in your Flask app to call it, the next step is to start writing the HTML for your app. The HTML below does a few things:

- Provides a voice selection drop-down
- Adds a button to convert text-to-speech
- Adds an audio element, which is used to play back the synthesized speech

1. Open `index.html` and locate these code comments:

```
<!-- Start voice font selection code -->

<!-- End voice font selection code -->
```

2. Replace the code comments with this HTML block:

```
<div class="form-group">
  <label for="select-voice"><strong>Select voice font:</strong></label>
  <select class="form-control" id="select-voice">
    <option value="(ar-SA, Naayf)">Arabic | Male | Naayf</option>
    <option value="(ca-ES, HerenaRUS)">Catalan | Female | HerenaRUS</option>
    <option value="(zh-CN, HuihuiRUS)">Chinese (Mainland) | Female | HuihuiRUS</option>
    <option value="(zh-CN, Kangkang, Apollo)">Chinese (Mainland) | Male | Kangkang, Apollo</option>
    <option value="(zh-HK, Tracy, Apollo)">Chinese (Hong Kong)| Female | Tracy, Apollo</option>
    <option value="(zh-HK, Danny, Apollo)">Chinese (Hong Kong) | Male | Danny, Apollo</option>
    <option value="(zh-TW, Yating, Apollo)">Chinese (Taiwan)| Female | Yating, Apollo</option>
    <option value="(zh-TW, Zhiwei, Apollo)">Chinese (Taiwan) | Male | Zhiwei, Apollo</option>
    <option value="(hr-HR, Matej)">Croatian | Male | Matej</option>
    <option value="(en-US, AriaRUS)">English (US) | Female | AriaRUS</option>
    <option value="(en-US, Guy24kRUS)">English (US) | Male | Guy24kRUS</option>
    <option value="(en-IE, Sean)">English (IE) | Male | Sean</option>
    <option value="(fr-FR, Julie, Apollo)">French | Female | Julie, Apollo</option>
    <option value="(fr-FR, HortenseRUS)">French | Female | Julie, HortenseRUS</option>
    <option value="(fr-FR, Paul, Apollo)">French | Male | Paul, Apollo</option>
    <option value="(de-DE, Hedda)">German | Female | Hedda</option>
    <option value="(de-DE, HeddaRUS)">German | Female | HeddaRUS</option>
    <option value="(de-DE, Stefan, Apollo)">German | Male | Apollo</option>
    <option value="(el-GR, Stefanos)">Greek | Male | Stefanos</option>
    <option value="(he-IL, Asaf)">Hebrew (Isreal) | Male | Asaf</option>
    <option value="(hi-IN, Kalpana, Apollo)">Hindi | Female | Kalpana, Apollo</option>
    <option value="(hi-IN, Hemant)">Hindi | Male | Hemant</option>
    <option value="(it-IT, LuciaRUS)">Italian | Female | LuciaRUS</option>
    <option value="(it-IT, Cosimo, Apollo)">Italian | Male | Cosimo, Apollo</option>
    <option value="(ja-JP, Ichiro, Apollo)">Japanese | Male | Ichiro</option>
    <option value="(ja-JP, HarukaRUS)">Japanese | Female | HarukaRUS</option>
    <option value="(ko-KR, HeamiRUS)">Korean | Female | Heami</option>
    <option value="(pt-BR, HeloisaRUS)">Portuguese (Brazil) | Female | HeloisaRUS</option>
    <option value="(pt-BR, Daniel, Apollo)">Portuguese (Brazil) | Male | Daniel, Apollo</option>
    <option value="(pt-PT, HeliaRUS)">Portuguese (Portugal) | Female | HeliaRUS</option>
    <option value="(ru-RU, Irina, Apollo)">Russian | Female | Irina, Apollo</option>
    <option value="(ru-RU, Pavel, Apollo)">Russian | Male | Pavel, Apollo</option>
    <option value="(ru-RU, EkaterinaRUS)">Russian | Female | EkaterinaRUS</option>
    <option value="(es-ES, Laura, Apollo)">Spanish | Female | Laura, Apollo</option>
    <option value="(es-ES, HelenaRUS)">Spanish | Female | HelenaRUS</option>
    <option value="(es-ES, Pablo, Apollo)">Spanish | Male | Pablo, Apollo</option>
    <option value="(th-TH, Pattara)">Thai | Male | Pattara</option>
    <option value="(tr-TR, SedaRUS)">Turkish | Female | SedaRUS</option>
    <option value="(vi-VN, An)">Vietnamese | Male | An</option>
  </select>
</div>
```

3. Next, locate these code comments:

```
<!-- Add Speech Synthesis button and audio element -->

<!-- End Speech Synthesis button -->
```

4. Replace the code comments with this HTML block:

```
<button type="submit" class="btn btn-primary mb-2" id="text-to-speech">Convert text-to-speech</button>
<div id="audio-playback">
  <audio id="audio" controls>
    <source id="audio-source" type="audio/mpeg" />
  </audio>
</div>
```

5. Make sure to save your work.

Update `main.js`

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the translation and the voice font are assigned to variables, and then passed along in the request to the `text-to-speech` route.

The code then iterates through the response, and updates the HTML with the sentiment scores.

1. From your IDE, create a file named `main.js` in the `static` directory.
2. Copy this code into `static/scripts/main.js`:

```
// Convert text-to-speech
$("#text-to-speech").on("click", function(e) {
  e.preventDefault();
  var ttsInput = document.getElementById("translation-result").value;
  var ttsVoice = document.getElementById("select-voice").value;
  var ttsRequest = { 'text': ttsInput, 'voice': ttsVoice }

  var xhr = new XMLHttpRequest();
  xhr.open("post", "/text-to-speech", true);
  xhr.setRequestHeader("Content-Type", "application/json");
  xhr.responseType = "blob";
  xhr.onload = function(evt){
    if (xhr.status === 200) {
      audioBlob = new Blob([xhr.response], {type: "audio/mpeg"});
      audioURL = URL.createObjectURL(audioBlob);
      if (audioURL.length > 5){
        var audio = document.getElementById("audio");
        var source = document.getElementById("audio-source");
        source.src = audioURL;
        audio.load();
        audio.play();
      }else{
        console.log("An error occurred getting and playing the audio.")
      }
    }
  }
  xhr.send(JSON.stringify(ttsRequest));
});
// Code for automatic language selection goes here.
```

3. You're almost done. The last thing you're going to do is add some code to `main.js` to automatically select a voice font based on the language selected for translation. Add this code block to `main.js`:

```
// Automatic voice font selection based on translation output.
$('select[id="select-language"]').change(function(e) {
  if ($(this).val() == "ar"){
    document.getElementById("select-voice").value = "(ar-SA, Naayf)";
  }
  if ($(this).val() == "ca"){
    document.getElementById("select-voice").value = "(ca-ES, HerenaRUS)";
  }
  if ($(this).val() == "zh-Hans"){
    document.getElementById("select-voice").value = "(zh-HK, Tracy, Apollo)";
  }
  if ($(this).val() == "zh-Hant"){
    document.getElementById("select-voice").value = "(zh-HK, Tracy, Apollo)";
  }
  if ($(this).val() == "hr"){
    document.getElementById("select-voice").value = "(hr-HR, Matej)";
  }
  if ($(this).val() == "en"){
    document.getElementById("select-voice").value = "(en-US, Jessa24kRUS)";
  }
  if ($(this).val() == "fr"){
    document.getElementById("select-voice").value = "(fr-FR, HortenseRUS)";
  }
  if ($(this).val() == "de"){
    document.getElementById("select-voice").value = "(de-DE, HeddaRUS)";
  }
  if ($(this).val() == "el"){
    document.getElementById("select-voice").value = "(el-GR, Stefanos)";
  }
  if ($(this).val() == "he"){
    document.getElementById("select-voice").value = "(he-IL, Asaf)";
  }
  if ($(this).val() == "hi"){
    document.getElementById("select-voice").value = "(hi-IN, Kalpana, Apollo)";
  }
  if ($(this).val() == "it"){
    document.getElementById("select-voice").value = "(it-IT, LuciaRUS)";
  }
  if ($(this).val() == "ja"){
    document.getElementById("select-voice").value = "(ja-JP, HarukaRUS)";
  }
  if ($(this).val() == "ko"){
    document.getElementById("select-voice").value = "(ko-KR, HeamiRUS)";
  }
  if ($(this).val() == "pt"){
    document.getElementById("select-voice").value = "(pt-BR, HeloisaRUS)";
  }
  if ($(this).val() == "ru"){
    document.getElementById("select-voice").value = "(ru-RU, EkaterinaRUS)";
  }
  if ($(this).val() == "es"){
    document.getElementById("select-voice").value = "(es-ES, HelenaRUS)";
  }
  if ($(this).val() == "th"){
    document.getElementById("select-voice").value = "(th-TH, Pattara)";
  }
  if ($(this).val() == "tr"){
    document.getElementById("select-voice").value = "(tr-TR, SedaRUS)";
  }
  if ($(this).val() == "vi"){
    document.getElementById("select-voice").value = "(vi-VN, An)";
  }
});
```

Test your app

Let's test speech synthesis in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. Next, select a voice, then press the convert text-to-speech button. the translation should be played back as synthesized speech. If it doesn't work, make sure that you've added your subscription key.

> **TIP**
> If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

That's it, you have a working app that performs translations, analyzes sentiment, and synthesized speech. Press **CTRL + c** to kill the app. Be sure to check out the other Azure Cognitive Services.

Get the source code

The source code for this project is available on GitHub.

Next steps

- Translator reference
- Language service API reference
- Text-to-speech API reference

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

Language and voice support for the Speech service

11/23/2021 • 22 minutes to read • Edit Online

Language support varies by Speech service functionality. The following tables summarize language support for Speech-to-text, Text-to-speech, Speech translation and Speaker Recognition service offerings.

Speech-to-text

Both the Microsoft Speech SDK and the REST API support the following languages (locales).

To improve accuracy, customization is offered for a subset of the languages through uploading **Audio + Human-labeled Transcripts** or **Related Text: Sentences**. Support for customization of the acoustic model with **Audio + Human-labeled Transcripts** is limited to the specific base models listed below. Other base models and languages will only use the text of the transcripts to train custom models just like with **Related Text: Sentences**. To learn more about customization, see Get started with Custom Speech.

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Algeria)	`ar-DZ`	Text		
Arabic (Bahrain), modern standard	`ar-BH`	Text		
Arabic (Egypt)	`ar-EG`	Text	Yes	
Arabic (Iraq)	`ar-IQ`	Text		
Arabic (Israel)	`ar-IL`	Text		
Arabic (Jordan)	`ar-JO`	Text		
Arabic (Kuwait)	`ar-KW`	Text		
Arabic (Lebanon)	`ar-LB`	Text		
Arabic (Libya)	`ar-LY`	Text		
Arabic (Morocco)	`ar-MA`	Text		
Arabic (Oman)	`ar-OM`	Text		
Arabic (Qatar)	`ar-QA`	Text		
Arabic (Saudi Arabia)	`ar-SA`	Text		
Arabic (Palestinian Authority)	`ar-PS`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Arabic (Syria)	`ar-SY`	Text		
Arabic (Tunisia)	`ar-TN`	Text		
Arabic (United Arab Emirates)	`ar-AE`	Text		
Arabic (Yemen)	`ar-YE`	Text		
Bulgarian (Bulgaria)	`bg-BG`	Text		
Catalan (Spain)	`ca-ES`	Text Pronunciation	Yes	
Chinese (Cantonese, Traditional)	`zh-HK`	Audio (20201015) Text	Yes	
Chinese (Mandarin, Simplified)	`zh-CN`	Audio (20200910) Text	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Audio (20190701, 20201015) Text	Yes	
Croatian (Croatia)	`hr-HR`	Text Pronunciation		
Czech (Czech)	`cs-CZ`	Text Pronunciation		
Danish (Denmark)	`da-DK`	Text Pronunciation	Yes	
Dutch (Netherlands)	`nl-NL`	Audio (20201015) Text Pronunciation	Yes	
English (Australia)	`en-AU`	Audio (20201019) Text Pronunciation	Yes	
English (Canada)	`en-CA`	Audio (20201019) Text Pronunciation	Yes	
English (Ghana)	`en-GH`	Text Pronunciation		
English (Hong Kong)	`en-HK`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
English (India)	`en-IN`	Audio (20200923) Text Pronunciation		
English (Ireland)	`en-IE`	Text Pronunciation		
English (Kenya)	`en-KE`	Text Pronunciation		
English (New Zealand)	`en-NZ`	Audio (20201019) Text Pronunciation		
English (Nigeria)	`en-NG`	Text Pronunciation		
English (Philippines)	`en-PH`	Text Pronunciation		
English (Singapore)	`en-SG`	Text Pronunciation		
English (South Africa)	`en-ZA`	Text Pronunciation		
English (Tanzania)	`en-TZ`	Text Pronunciation		
English (United Kingdom)	`en-GB`	Audio (20201019) Text Pronunciation	Yes	Yes
English (United States)	`en-US`	Audio (20201019, 20210223) Text Pronunciation	Yes	Yes
Estonian(Estonia)	`et-EE`	Text Pronunciation		
Filipino (Philippines)	`fil-PH`	Text Pronunciation		
Finnish (Finland)	`fi-FI`	Text Pronunciation	Yes	
French (Canada)	`fr-CA`	Audio (20201015) Text Pronunciation	Yes	
French (France)	`fr-FR`	Audio (20201015) Text Pronunciation	Yes	

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
French (Switzerland)	`fr-CH`	Text Pronunciation		
German (Austria)	`de-AT`	Text Pronunciation		
German (Switzerland)	`de-CH`	Text Pronunciation		
German (Germany)	`de-DE`	Audio (20190701, 20200619, 20201127) Text Pronunciation	Yes	
Greek (Greece)	`el-GR`	Text	Yes	
Gujarati (Indian)	`gu-IN`	Text		
Hebrew (Israel)	`he-IL`	Text		
Hindi (India)	`hi-IN`	Audio (20200701) Text	Yes	
Hungarian (Hungary)	`hu-HU`	Text Pronunciation		
Indonesian (Indonesia)	`id-ID`	Text Pronunciation		
Irish (Ireland)	`ga-IE`	Text Pronunciation		
Italian (Italy)	`it-IT`	Audio (20201016) Text Pronunciation	Yes	
Japanese (Japan)	`ja-JP`	Text	Yes	
Kannada (India)	`kn-IN`	Text		
Korean (Korea)	`ko-KR`	Audio (20201015) Text	Yes	
Latvian (Latvia)	`lv-LV`	Text Pronunciation		
Lithuanian (Lithuania)	`lt-LT`	Text Pronunciation		
Malay (Malaysia)	`ms-MY`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Maltese (Malta)	`mt-MT`	Text		
Marathi (India)	`mr-IN`	Text		
Norwegian (Bokmål, Norway)	`nb-NO`	Text	Yes	
Persian (Iran)	`fa-IR`	Text		
Polish (Poland)	`pl-PL`	Text Pronunciation	Yes	
Portuguese (Brazil)	`pt-BR`	Audio (20190620, 20201015) Text Pronunciation	Yes	
Portuguese (Portugal)	`pt-PT`	Text Pronunciation	Yes	
Romanian (Romania)	`ro-RO`	Text Pronunciation	Yes	
Russian (Russia)	`ru-RU`	Audio (20200907) Text	Yes	
Slovak (Slovakia)	`sk-SK`	Text Pronunciation		
Slovenian (Slovenia)	`sl-SI`	Text Pronunciation		
Spanish (Argentina)	`es-AR`	Text Pronunciation		
Spanish (Bolivia)	`es-BO`	Text Pronunciation		
Spanish (Chile)	`es-CL`	Text Pronunciation		
Spanish (Colombia)	`es-CO`	Text Pronunciation		
Spanish (Costa Rica)	`es-CR`	Text Pronunciation		
Spanish (Cuba)	`es-CU`	Text Pronunciation		
Spanish (Dominican Republic)	`es-DO`	Text Pronunciation		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Spanish (Ecuador)	`es-EC`	Text Pronunciation		
Spanish (El Salvador)	`es-SV`	Text Pronunciation		
Spanish (Equatorial Guinea)	`es-GQ`	Text		
Spanish (Guatemala)	`es-GT`	Text Pronunciation		
Spanish (Honduras)	`es-HN`	Text Pronunciation		
Spanish (Mexico)	`es-MX`	Audio (20200907) Text Pronunciation	Yes	
Spanish (Nicaragua)	`es-NI`	Text Pronunciation		
Spanish (Panama)	`es-PA`	Text Pronunciation		
Spanish (Paraguay)	`es-PY`	Text Pronunciation		
Spanish (Peru)	`es-PE`	Text Pronunciation		
Spanish (Puerto Rico)	`es-PR`	Text Pronunciation		
Spanish (Spain)	`es-ES`	Audio (20201015) Text Pronunciation	Yes	
Spanish (Uruguay)	`es-UY`	Text Pronunciation		
Spanish (USA)	`es-US`	Text Pronunciation		
Spanish (Venezuela)	`es-VE`	Text Pronunciation		
Swahili (Kenya)	`sw-KE`	Text		
Swedish (Sweden)	`sv-SE`	Text Pronunciation	Yes	
Tamil (India)	`ta-IN`	Text		

LANGUAGE	LOCALE (BCP-47)	CUSTOMIZATIONS	LANGUAGE IDENTIFICATION	PRONUNCIATION ASSESSMENT
Telugu (India)	`te-IN`	Text		
Thai (Thailand)	`th-TH`	Text	Yes	
Turkish (Turkey)	`tr-TR`	Text		
Vietnamese (Vietnam)	`vi-VN`	Text		

Text-to-speech

Both the Microsoft Speech SDK and REST APIs support these voices, each of which supports a specific language and dialect, identified by locale. You can also get a full list of languages and voices supported for each specific region/endpoint through the voices list API.

> **IMPORTANT**
> Pricing varies for standard, custom and neural voices. Please visit the Pricing page for additional information.

Neural voices

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

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

> **NOTE**
> Neural voices are created from samples that use a 24 khz sample rate. All voices can upsample or downsample to other sample rates when synthesizing.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Afrikaans (South Africa)	`af-ZA`	Female	`af-ZA-AdriNeural` New	General
Afrikaans (South Africa)	`af-ZA`	Male	`af-ZA-WillemNeural` New	General
Amharic (Ethiopia)	`am-ET`	Female	`am-ET-MekdesNeural` New	General
Amharic (Ethiopia)	`am-ET`	Male	`am-ET-AmehaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Algeria)	`ar-DZ`	Female	`ar-DZ-AminaNeural` New	General
Arabic (Algeria)	`ar-DZ`	Male	`ar-DZ-IsmaelNeural` New	General
Arabic (Bahrain)	`ar-BH`	Female	`ar-BH-LailaNeural` New	General
Arabic (Bahrain)	`ar-BH`	Male	`ar-BH-AliNeural` New	General
Arabic (Egypt)	`ar-EG`	Female	`ar-EG-SalmaNeural`	General
Arabic (Egypt)	`ar-EG`	Male	`ar-EG-ShakirNeural`	General
Arabic (Iraq)	`ar-IQ`	Female	`ar-IQ-RanaNeural` New	General
Arabic (Iraq)	`ar-IQ`	Male	`ar-IQ-BasselNeural` New	General
Arabic (Jordan)	`ar-JO`	Female	`ar-JO-Sana Neural` New	General
Arabic (Jordan)	`ar-JO`	Male	`ar-JO-Taim Neural` New	General
Arabic (Kuwait)	`ar-KW`	Female	`ar-KW-NouraNeural` New	General
Arabic (Kuwait)	`ar-KW`	Male	`ar-KW-FahedNeural` New	General
Arabic (Libya)	`ar-LY`	Female	`ar-LY-ImanNeural` New	General
Arabic (Libya)	`ar-LY`	Male	`ar-LY-OmarNeural` New	General
Arabic (Morocco)	`ar-MA`	Female	`ar-MA-MounaNeural` New	General
Arabic (Morocco)	`ar-MA`	Male	`ar-MA-JamalNeural` New	General
Arabic (Qatar)	`ar-QA`	Female	`ar-QA-AmalNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Arabic (Qatar)	`ar-QA`	Male	`ar-QA-MoazNeural` New	General
Arabic (Saudi Arabia)	`ar-SA`	Female	`ar-SA-ZariyahNeural`	General
Arabic (Saudi Arabia)	`ar-SA`	Male	`ar-SA-HamedNeural`	General
Arabic (Syria)	`ar-SY`	Female	`ar-SY-AmanyNeural` New	General
Arabic (Syria)	`ar-SY`	Male	`ar-SY-LaithNeural` New	General
Arabic (Tunisia)	`ar-TN`	Female	`ar-TN-ReemNeural` New	General
Arabic (Tunisia)	`ar-TN`	Male	`ar-TN-HediNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Female	`ar-AE-FatimaNeural` New	General
Arabic (United Arab Emirates)	`ar-AE`	Male	`ar-AE-HamdanNeural` New	General
Arabic (Yemen)	`ar-YE`	Female	`ar-YE-MaryamNeural` New	General
Arabic (Yemen)	`ar-YE`	Male	`ar-YE-SalehNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Female	`bn-BD-NabanitaNeural` New	General
Bangla (Bangladesh)	`bn-BD`	Male	`bn-BD-PradeepNeural` New	General
Bulgarian (Bulgaria)	`bg-BG`	Female	`bg-BG-KalinaNeural`	General
Bulgarian (Bulgaria)	`bg-BG`	Male	`bg-BG-BorislavNeural`	General
Burmese (Myanmar)	`my-MM`	Female	`my-MM-NilarNeural` New	General
Burmese (Myanmar)	`my-MM`	Male	`my-MM-ThihaNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Catalan (Spain)	`ca-ES`	Female	`ca-ES-AlbaNeural`	General
Catalan (Spain)	`ca-ES`	Female	`ca-ES-JoanaNeural`	General
Catalan (Spain)	`ca-ES`	Male	`ca-ES-EnricNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuGaaiNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Female	`zh-HK-HiuMaanNeural`	General
Chinese (Cantonese, Traditional)	`zh-HK`	Male	`zh-HK-WanLungNeural`	General
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaohanNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaomoNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoruiNeural`	Senior voice, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxiaoNeural`	General, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoxuanNeural`	General, multiple role-play and styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyouNeural`	Child voice, optimized for story narrating
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunxiNeural`	General, multiple styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyangNeural`	Optimized for news reading, multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Male	`zh-CN-YunyeNeural`	Optimized for story narrating
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoChenNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Chinese (Taiwanese Mandarin)	`zh-TW`	Female	`zh-TW-HsiaoYuNeural`	General
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-YunJheNeural`	General
Croatian (Croatia)	`hr-HR`	Female	`hr-HR-GabrijelaNeural`	General
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-SreckoNeural`	General
Czech (Czech)	`cs-CZ`	Female	`cs-CZ-VlastaNeural`	General
Czech (Czech)	`cs-CZ`	Male	`cs-CZ-AntoninNeural`	General
Danish (Denmark)	`da-DK`	Female	`da-DK-ChristelNeural`	General
Danish (Denmark)	`da-DK`	Male	`da-DK-JeppeNeural`	General
Dutch (Belgium)	`nl-BE`	Female	`nl-BE-DenaNeural`	General
Dutch (Belgium)	`nl-BE`	Male	`nl-BE-ArnaudNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-ColetteNeural`	General
Dutch (Netherlands)	`nl-NL`	Female	`nl-NL-FennaNeural`	General
Dutch (Netherlands)	`nl-NL`	Male	`nl-NL-MaartenNeural`	General
English (Australia)	`en-AU`	Female	`en-AU-NatashaNeural`	General
English (Australia)	`en-AU`	Male	`en-AU-WilliamNeural`	General
English (Canada)	`en-CA`	Female	`en-CA-ClaraNeural`	General
English (Canada)	`en-CA`	Male	`en-CA-LiamNeural`	General
English (Hongkong)	`en-HK`	Female	`en-HK-YanNeural`	General
English (Hongkong)	`en-HK`	Male	`en-HK-SamNeural`	General
English (India)	`en-IN`	Female	`en-IN-NeerjaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (India)	`en-IN`	Male	`en-IN-PrabhatNeural`	General
English (Ireland)	`en-IE`	Female	`en-IE-EmilyNeural`	General
English (Ireland)	`en-IE`	Male	`en-IE-ConnorNeural`	General
English (Kenya)	`en-KE`	Female	`en-KE-AsiliaNeural` New	General
English (Kenya)	`en-KE`	Male	`en-KE-ChilembaNeural` New	General
English (New Zealand)	`en-NZ`	Female	`en-NZ-MollyNeural`	General
English (New Zealand)	`en-NZ`	Male	`en-NZ-MitchellNeural`	General
English (Nigeria)	`en-NG`	Female	`en-NG-EzinneNeural` New	General
English (Nigeria)	`en-NG`	Male	`en-NG-AbeoNeural` New	General
English (Philippines)	`en-PH`	Female	`en-PH-RosaNeural`	General
English (Philippines)	`en-PH`	Male	`en-PH-JamesNeural`	General
English (Singapore)	`en-SG`	Female	`en-SG-LunaNeural`	General
English (Singapore)	`en-SG`	Male	`en-SG-WayneNeural`	General
English (South Africa)	`en-ZA`	Female	`en-ZA-LeahNeural`	General
English (South Africa)	`en-ZA`	Male	`en-ZA-LukeNeural`	General
English (Tanzania)	`en-TZ`	Female	`en-TZ-ImaniNeural` New	General
English (Tanzania)	`en-TZ`	Male	`en-TZ-ElimuNeural` New	General
English (United Kingdom)	`en-GB`	Female	`en-GB-LibbyNeural`	General
English (United Kingdom)	`en-GB`	Female	`en-GB-SoniaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United Kingdom)	`en-GB`	Female	`en-GB-MiaNeural` Retired on 30 October 2021, see below	General
English (United Kingdom)	`en-GB`	Male	`en-GB-RyanNeural`	General
English (United States)	`en-US`	Female	`en-US-AmberNeural`	General
English (United States)	`en-US`	Female	`en-US-AriaNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-AshleyNeural`	General
English (United States)	`en-US`	Female	`en-US-CoraNeural`	General
English (United States)	`en-US`	Female	`en-US-ElizabethNeural`	General
English (United States)	`en-US`	Female	`en-US-JennyNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Female	`en-US-MichelleNeural`	General
English (United States)	`en-US`	Female	`en-US-MonicaNeural`	General
English (United States)	`en-US`	Female	`en-US-SaraNeural`	General, multiple voice styles available using SSML
English (United States)	`en-US`	Kid	`en-US-AnaNeural`	General
English (United States)	`en-US`	Male	`en-US-BrandonNeural`	General
English (United States)	`en-US`	Male	`en-US-ChristopherNeural`	General
English (United States)	`en-US`	Male	`en-US-EricNeural`	General
English (United States)	`en-US`	Male	`en-US-GuyNeural`	General, multiple voice styles available using SSML

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Male	`en-US-JacobNeural`	General
Estonian (Estonia)	`et-EE`	Female	`et-EE-AnuNeural`	General
Estonian (Estonia)	`et-EE`	Male	`et-EE-KertNeural`	General
Filipino (Philippines)	`fil-PH`	Female	`fil-PH-BlessicaNeural` New	General
Filipino (Philippines)	`fil-PH`	Male	`fil-PH-AngeloNeural` New	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-NooraNeural`	General
Finnish (Finland)	`fi-FI`	Female	`fi-FI-SelmaNeural`	General
Finnish (Finland)	`fi-FI`	Male	`fi-FI-HarriNeural`	General
French (Belgium)	`fr-BE`	Female	`fr-BE-CharlineNeural`	General
French (Belgium)	`fr-BE`	Male	`fr-BE-GerardNeural`	General
French (Canada)	`fr-CA`	Female	`fr-CA-SylvieNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-AntoineNeural`	General
French (Canada)	`fr-CA`	Male	`fr-CA-JeanNeural`	General
French (France)	`fr-FR`	Female	`fr-FR-DeniseNeural`	General
French (France)	`fr-FR`	Male	`fr-FR-HenriNeural`	General
French (Switzerland)	`fr-CH`	Female	`fr-CH-ArianeNeural`	General
French (Switzerland)	`fr-CH`	Male	`fr-CH-FabriceNeural`	General
Galician (Spain)	`gl-ES`	Female	`gl-ES-SabelaNeural` New	General
Galician (Spain)	`gl-ES`	Male	`gl-ES-RoiNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
German (Austria)	`de-AT`	Female	`de-AT-IngridNeural`	General
German (Austria)	`de-AT`	Male	`de-AT-JonasNeural`	General
German (Germany)	`de-DE`	Female	`de-DE-KatjaNeural`	General
German (Germany)	`de-DE`	Male	`de-DE-ConradNeural`	General
German (Switzerland)	`de-CH`	Female	`de-CH-LeniNeural`	General
German (Switzerland)	`de-CH`	Male	`de-CH-JanNeural`	General
Greek (Greece)	`el-GR`	Female	`el-GR-AthinaNeural`	General
Greek (Greece)	`el-GR`	Male	`el-GR-NestorasNeural`	General
Gujarati (India)	`gu-IN`	Female	`gu-IN-DhwaniNeural`	General
Gujarati (India)	`gu-IN`	Male	`gu-IN-NiranjanNeural`	General
Hebrew (Israel)	`he-IL`	Female	`he-IL-HilaNeural`	General
Hebrew (Israel)	`he-IL`	Male	`he-IL-AvriNeural`	General
Hindi (India)	`hi-IN`	Female	`hi-IN-SwaraNeural`	General
Hindi (India)	`hi-IN`	Male	`hi-IN-MadhurNeural`	General
Hungarian (Hungary)	`hu-HU`	Female	`hu-HU-NoemiNeural`	General
Hungarian (Hungary)	`hu-HU`	Male	`hu-HU-TamasNeural`	General
Indonesian (Indonesia)	`id-ID`	Female	`id-ID-GadisNeural`	General
Indonesian (Indonesia)	`id-ID`	Male	`id-ID-ArdiNeural`	General
Irish (Ireland)	`ga-IE`	Female	`ga-IE-OrlaNeural`	General
Irish (Ireland)	`ga-IE`	Male	`ga-IE-ColmNeural`	General
Italian (Italy)	`it-IT`	Female	`it-IT-ElsaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Italian (Italy)	`it-IT`	Female	`it-IT-IsabellaNeural`	General
Italian (Italy)	`it-IT`	Male	`it-IT-DiegoNeural`	General
Japanese (Japan)	`ja-JP`	Female	`ja-JP-NanamiNeural`	General
Japanese (Japan)	`ja-JP`	Male	`ja-JP-KeitaNeural`	General
Javanese (Indonesia)	`jv-ID`	Female	`jv-ID-SitiNeural` New	General
Javanese (Indonesia)	`jv-ID`	Male	`jv-ID-DimasNeural` New	General
Khmer (Cambodia)	`km-KH`	Female	`km-KH-SreymomNeural` New	General
Khmer (Cambodia)	`km-KH`	Male	`km-KH-PisethNeural` New	General
Korean (Korea)	`ko-KR`	Female	`ko-KR-SunHiNeural`	General
Korean (Korea)	`ko-KR`	Male	`ko-KR-InJoonNeural`	General
Latvian (Latvia)	`lv-LV`	Female	`lv-LV-EveritaNeural`	General
Latvian (Latvia)	`lv-LV`	Male	`lv-LV-NilsNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Female	`lt-LT-OnaNeural`	General
Lithuanian (Lithuania)	`lt-LT`	Male	`lt-LT-LeonasNeural`	General
Malay (Malaysia)	`ms-MY`	Female	`ms-MY-YasminNeural`	General
Malay (Malaysia)	`ms-MY`	Male	`ms-MY-OsmanNeural`	General
Maltese (Malta)	`mt-MT`	Female	`mt-MT-GraceNeural`	General
Maltese (Malta)	`mt-MT`	Male	`mt-MT-JosephNeural`	General
Marathi (India)	`mr-IN`	Female	`mr-IN-AarohiNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Marathi (India)	`mr-IN`	Male	`mr-IN-ManoharNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-IselinNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Female	`nb-NO-PernilleNeural`	General
Norwegian (Bokmål, Norway)	`nb-NO`	Male	`nb-NO-FinnNeural`	General
Persian (Iran)	`fa-IR`	Female	`fa-IR-DilaraNeural` New	General
Persian (Iran)	`fa-IR`	Male	`fa-IR-FaridNeural` New	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-AgnieszkaNeural`	General
Polish (Poland)	`pl-PL`	Female	`pl-PL-ZofiaNeural`	General
Polish (Poland)	`pl-PL`	Male	`pl-PL-MarekNeural`	General
Portuguese (Brazil)	`pt-BR`	Female	`pt-BR-FranciscaNeural`	General, multiple voice styles available using SSML
Portuguese (Brazil)	`pt-BR`	Male	`pt-BR-AntonioNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-FernandaNeural`	General
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-RaquelNeural`	General
Portuguese (Portugal)	`pt-PT`	Male	`pt-PT-DuarteNeural`	General
Romanian (Romania)	`ro-RO`	Female	`ro-RO-AlinaNeural`	General
Romanian (Romania)	`ro-RO`	Male	`ro-RO-EmilNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-DariyaNeural`	General
Russian (Russia)	`ru-RU`	Female	`ru-RU-SvetlanaNeural`	General
Russian (Russia)	`ru-RU`	Male	`ru-RU-DmitryNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Slovak (Slovakia)	`sk-SK`	Female	`sk-SK-ViktoriaNeural`	General
Slovak (Slovakia)	`sk-SK`	Male	`sk-SK-LukasNeural`	General
Slovenian (Slovenia)	`sl-SI`	Female	`sl-SI-PetraNeural`	General
Slovenian (Slovenia)	`sl-SI`	Male	`sl-SI-RokNeural`	General
Somali (Somalia)	`so-SO`	Female	`so-SO-UbaxNeural` New	General
Somali (Somalia)	`so-SO`	Male	`so-SO-MuuseNeural` New	General
Spanish (Argentina)	`es-AR`	Female	`es-AR-ElenaNeural`	General
Spanish (Argentina)	`es-AR`	Male	`es-AR-TomasNeural`	General
Spanish (Bolivia)	`es-BO`	Female	`es-BO-SofiaNeural` New	General
Spanish (Bolivia)	`es-BO`	Male	`es-BO-MarceloNeural` New	General
Spanish (Chile)	`es-CL`	Female	`es-CL-CatalinaNeural` New	General
Spanish (Chile)	`es-CL`	Male	`es-CL-LorenzoNeural` New	General
Spanish (Colombia)	`es-CO`	Female	`es-CO-SalomeNeural`	General
Spanish (Colombia)	`es-CO`	Male	`es-CO-GonzaloNeural`	General
Spanish (Costa Rica)	`es-CR`	Female	`es-CR-MariaNeural` New	General
Spanish (Costa Rica)	`es-CR`	Male	`es-CR-JuanNeural` New	General
Spanish (Cuba)	`es-CU`	Female	`es-CU-BelkysNeural` New	General
Spanish (Cuba)	`es-CU`	Male	`es-CU-ManuelNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Dominican Republic)	`es-DO`	Female	`es-DO-RamonaNeural` New	General
Spanish (Dominican Republic)	`es-DO`	Male	`es-DO-EmilioNeural` New	General
Spanish (Ecuador)	`es-EC`	Female	`es-EC-AndreaNeural` New	General
Spanish (Ecuador)	`es-EC`	Male	`es-EC-LuisNeural` New	General
Spanish (El Salvador)	`es-SV`	Female	`es-SV-LorenaNeural` New	General
Spanish (El Salvador)	`es-SV`	Male	`es-SV-RodrigoNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Female	`es-GQ-TeresaNeural` New	General
Spanish (Equatorial Guinea)	`es-GQ`	Male	`es-GQ-JavierNeural` New	General
Spanish (Guatemala)	`es-GT`	Female	`es-GT-MartaNeural` New	General
Spanish (Guatemala)	`es-GT`	Male	`es-GT-AndresNeural` New	General
Spanish (Honduras)	`es-HN`	Female	`es-HN-KarlaNeural` New	General
Spanish (Honduras)	`es-HN`	Male	`es-HN-CarlosNeural` New	General
Spanish (Mexico)	`es-MX`	Female	`es-MX-DaliaNeural`	General
Spanish (Mexico)	`es-MX`	Male	`es-MX-JorgeNeural`	General
Spanish (Nicaragua)	`es-NI`	Female	`es-NI-YolandaNeural` New	General
Spanish (Nicaragua)	`es-NI`	Male	`es-NI-FedericoNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Spanish (Panama)	`es-PA`	Female	`es-PA-MargaritaNeural` New	General
Spanish (Panama)	`es-PA`	Male	`es-PA-RobertoNeural` New	General
Spanish (Paraguay)	`es-PY`	Female	`es-PY-TaniaNeural` New	General
Spanish (Paraguay)	`es-PY`	Male	`es-PY-MarioNeural` New	General
Spanish (Peru)	`es-PE`	Female	`es-PE-CamilaNeural` New	General
Spanish (Peru)	`es-PE`	Male	`es-PE-AlexNeural` New	General
Spanish (Puerto Rico)	`es-PR`	Female	`es-PR-Karina Neural` New	General
Spanish (Puerto Rico)	`es-PR`	Male	`es-PR-Victor Neural` New	General
Spanish (Spain)	`es-ES`	Female	`es-ES-ElviraNeural`	General
Spanish (Spain)	`es-ES`	Male	`es-ES-AlvaroNeural`	General
Spanish (Uruguay)	`es-UY`	Female	`es-UY-ValentinaNeural` New	General
Spanish (Uruguay)	`es-UY`	Male	`es-UY-MateoNeural` New	General
Spanish (US)	`es-US`	Female	`es-US-PalomaNeural`	General
Spanish (US)	`es-US`	Male	`es-US-AlonsoNeural`	General
Spanish (Venezuela)	`es-VE`	Female	`es-VE-PaolaNeural` New	General
Spanish (Venezuela)	`es-VE`	Male	`es-VE-SebastianNeural` New	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Sundanese (Indonesia)	`su-ID`	Female	`su-ID-TutiNeural` New	General
Sundanese (Indonesia)	`su-ID`	Male	`su-ID-JajangNeural` New	General
Swahili (Kenya)	`sw-KE`	Female	`sw-KE-ZuriNeural`	General
Swahili (Kenya)	`sw-KE`	Male	`sw-KE-RafikiNeural`	General
Swahili (Tanzania)	`sw-TZ`	Female	`sw-TZ-RehemaNeural` New	General
Swahili (Tanzania)	`sw-TZ`	Male	`sw-TZ-DaudiNeural` New	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-HilleviNeural`	General
Swedish (Sweden)	`sv-SE`	Female	`sv-SE-SofieNeural`	General
Swedish (Sweden)	`sv-SE`	Male	`sv-SE-MattiasNeural`	General
Tamil (India)	`ta-IN`	Female	`ta-IN-PallaviNeural`	General
Tamil (India)	`ta-IN`	Male	`ta-IN-ValluvarNeural`	General
Tamil (Singapore)	`ta-SG`	Female	`ta-SG-VenbaNeural` New	General
Tamil (Singapore)	`ta-SG`	Male	`ta-SG-AnbuNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Female	`ta-LK-SaranyaNeural` New	General
Tamil (Sri Lanka)	`ta-LK`	Male	`ta-LK-KumarNeural` New	General
Telugu (India)	`te-IN`	Female	`te-IN-ShrutiNeural`	General
Telugu (India)	`te-IN`	Male	`te-IN-MohanNeural`	General
Thai (Thailand)	`th-TH`	Female	`th-TH-AcharaNeural`	General

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
Thai (Thailand)	th-TH	Female	th-TH-PremwadeeNeural	General
Thai (Thailand)	th-TH	Male	th-TH-NiwatNeural	General
Turkish (Turkey)	tr-TR	Female	tr-TR-EmelNeural	General
Turkish (Turkey)	tr-TR	Male	tr-TR-AhmetNeural	General
Ukrainian (Ukraine)	uk-UA	Female	uk-UA-PolinaNeural	General
Ukrainian (Ukraine)	uk-UA	Male	uk-UA-OstapNeural	General
Urdu (India)	ur-IN	Female	ur-IN-GulNeural New	General
Urdu (India)	ur-IN	Male	ur-IN-SalmanNeural New	General
Urdu (Pakistan)	ur-PK	Female	ur-PK-UzmaNeural	General
Urdu (Pakistan)	ur-PK	Male	ur-PK-AsadNeural	General
Uzbek (Uzbekistan)	uz-UZ	Female	uz-UZ-MadinaNeural New	General
Uzbek (Uzbekistan)	uz-UZ	Male	uz-UZ-SardorNeural New	General
Vietnamese (Vietnam)	vi-VN	Female	vi-VN-HoaiMyNeural	General
Vietnamese (Vietnam)	vi-VN	Male	vi-VN-NamMinhNeural	General
Welsh (United Kingdom)	cy-GB	Female	cy-GB-NiaNeural	General
Welsh (United Kingdom)	cy-GB	Male	cy-GB-AledNeural	General
Zulu (South Africa)	zu-ZA	Female	zu-ZA-ThandoNeural New	General
Zulu (South Africa)	zu-ZA	Male	zu-ZA-ThembaNeural New	General

> **IMPORTANT**
>
> The English (United Kingdom) voice `en-GB-MiaNeural` retired on **30 October 2021**. All service requests to `en-GB-MiaNeural` now will be re-directed to `en-GB-SoniaNeural` automatically since **30 October 2021**. If you are using container Neural TTS, please download and deploy the latest version, starting from **30 October 2021**, all requests with previous versions will be rejected.

Neural voices in preview

Below neural voices are in public preview.

LANGUAGE	LOCALE	GENDER	VOICE NAME	STYLE SUPPORT
English (United States)	`en-US`	Female	`en-US-JennyMultilingualNeural` New	General, multi-lingual capabilities available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaochenNeural` New	Optimized for spontaneous conversation
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoyanNeural` New	Optimized for customer service
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoshuangNeural` New	Child voice, optimized for child story and chat; multiple voice styles available using SSML
Chinese (Mandarin, Simplified)	`zh-CN`	Female	`zh-CN-XiaoqiuNeural` New	Optimized for narrating

> **IMPORTANT**
>
> Voices in public preview are only available in 3 service regions: East US, West Europe and Southeast Asia.

> **TIP**
>
> `en-US-JennyNeuralMultilingual` supports multiple languages. Check the voices list API for supported languages list.

For more information about regional availability, see regions.

To learn how you can configure and adjust neural voices, such as Speaking Styles, see Speech Synthesis Markup Language.

> **IMPORTANT**
>
> The `en-US-JessaNeural` voice has changed to `en-US-AriaNeural`. If you were using "Jessa" before, convert over to "Aria".

> **TIP**
> You can continue to use the full service name mapping like "Microsoft Server Speech Text to Speech Voice (en-US, ChristopherNeural)" in your speech synthesis requests.

Standard voices

More than 75 standard voices are available in over 45 languages and locales, which allow you to convert text into synthesized speech. For more information about regional availability, see regions.

> **IMPORTANT**
> We are retiring the standard voices on **31st August 2024** and they will no longer be supported after that date.We announced this in emails sent to all existing Speech subscriptions created before **31st August 2021**. During the retiring period (**31st August 2021 - 31st August 2024**), existing standard voice users can continue to use standard voices, but all new users/new speech resources must choose neural voices.

> **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 (Egypt)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Chinese (Taiwanese Mandarin)	`zh-TW`	Male	`zh-TW-Zhiwei`
Croatian (Croatia)	`hr-HR`	Male	`hr-HR-Matej`
Czech (Czech)	`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`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
French (France)	`fr-FR`	Female	`fr-FR-Julie`
French (France)	`fr-FR`	Male	`fr-FR-Paul`
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-HeloisaRUS`

LANGUAGE	LOCALE (BCP-47)	GENDER	VOICE NAME
Portuguese (Portugal)	`pt-PT`	Female	`pt-PT-HeliaRUS`
Romanian (Romania)	`ro-RO`	Male	`ro-RO-Andrei`
Russian (Russia)	`ru-RU`	Female	`ru-RU-EkaterinaRUS`
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".

TIP

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.

Customization

Custom Voice is available in the neural tier (a.k.a, Custom Neural Voice). Based on the Neural TTS technology and the multi-lingual multi-speaker universal model, Custom Neural Voice lets you create synthetic voices that

are rich in speaking styles, or adaptable cross languages. Check below for the languages supported.

> **IMPORTANT**
> The standard tier including the statistical parametric and the concatenative training methods of custom voice is being deprecated and will be retired on 2/29/2024. If you are using non-neural/standard Custom Voice, migrate to Custom Neural Voice immediately to enjoy the better quality and deploy the voices responsibly.

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Arabic (Egypt)	`ar-EG`	Yes	No
Bulgarian (Bulgaria)	`bg-BG`	Yes	No
Chinese (Mandarin, Simplified)	`zh-CN`	Yes	Yes
Chinese (Mandarin, Simplified), English bilingual	`zh-CN` bilingual	Yes	Yes
Chinese (Taiwanese Mandarin)	`zh-TW`	Yes	No
Czech (Czech)	`cs-CZ`	Yes	No
Dutch (Netherlands)	`nl-NL`	Yes	No
English (Australia)	`en-AU`	Yes	Yes
English (Canada)	`en-CA`	Yes	No
English (India)	`en-IN`	Yes	No
English (Ireland)	`en-IE`	Yes	No
English (United Kingdom)	`en-GB`	Yes	Yes
English (United States)	`en-US`	Yes	Yes
French (Canada)	`fr-CA`	Yes	Yes
French (France)	`fr-FR`	Yes	Yes
German (Austria)	`de-AT`	Yes	No
German (Germany)	`de-DE`	Yes	Yes
Hungarian (Hungary)	`hu-HU`	Yes	No
Italian (Italy)	`it-IT`	Yes	Yes

LANGUAGE	LOCALE	NEURAL	CROSS-LINGUAL
Japanese (Japan)	`ja-JP`	Yes	Yes
Korean (Korea)	`ko-KR`	Yes	Yes
Norwegian (Bokmål, Norway)	`nb-NO`	Yes	No
Portuguese (Brazil)	`pt-BR`	Yes	Yes
Portuguese (Portugal)	`pt-PT`	Yes	No
Russian (Russia)	`ru-RU`	Yes	Yes
Slovak (Slovakia)	`sk-SK`	Yes	No
Spanish (Mexico)	`es-MX`	Yes	Yes
Spanish (Spain)	`es-ES`	Yes	Yes
Turkish (Turkey)	`tr-TR`	Yes	No
Vietnamese (Vietnam)	`vi-VN`	Yes	No

Select the right locale that matches the training data you have to train a custom voice model. For example, if the recording data you have is spoken in English with a British accent, select `en-GB`.

> **NOTE**
> We do not support bi-lingual model training in Custom Voice, except for the Chinese-English bi-lingual. Select "Chinese-English bilingual" if you want to train a Chinese voice that can speak English as well. Chinese-English bilingual model training using the standard method is available in North Europe and North Central US only. Custom Neural Voice training is available in UK South and East US.

Speech translation

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

Text languages

TEXT LANGUAGE	LANGUAGE CODE
Afrikaans	`af`
Albanian	`sq`
Amharic	`am`

TEXT LANGUAGE	LANGUAGE CODE
Arabic	`ar`
Armenian	`hy`
Assamese	`as`
Azerbaijani	`az`
Bangla	`bn`
Bosnian (Latin)	`bs`
Bulgarian	`bg`
Cantonese (Traditional)	`yue`
Catalan	`ca`
Chinese (Literary)	`lzh`
Chinese Simplified	`zh-Hans`
Chinese Traditional	`zh-Hant`
Croatian	`hr`
Czech	`cs`
Danish	`da`
Dari	`prs`
Dutch	`nl`
English	`en`
Estonian	`et`
Fijian	`fj`
Filipino	`fil`
Finnish	`fi`
French	`fr`
French (Canada)	`fr-ca`

TEXT LANGUAGE	LANGUAGE CODE
German	`de`
Greek	`el`
Gujarati	`gu`
Haitian Creole	`ht`
Hebrew	`he`
Hindi	`hi`
Hmong Daw	`mww`
Hungarian	`hu`
Icelandic	`is`
Indonesian	`id`
Inuktitut	`iu`
Irish	`ga`
Italian	`it`
Japanese	`ja`
Kannada	`kn`
Kazakh	`kk`
Khmer	`km`
Klingon	`tlh-Latn`
Klingon (plqaD)	`tlh-Piqd`
Korean	`ko`
Kurdish (Central)	`ku`
Kurdish (Northern)	`kmr`
Lao	`lo`
Latvian	`lv`

TEXT LANGUAGE	LANGUAGE CODE
Lithuanian	`lt`
Malagasy	`mg`
Malay	`ms`
Malayalam	`ml`
Maltese	`mt`
Maori	`mi`
Marathi	`mr`
Myanmar	`my`
Nepali	`ne`
Norwegian	`nb`
Odia	`or`
Pashto	`ps`
Persian	`fa`
Polish	`pl`
Portuguese (Brazil)	`pt`
Portuguese (Portugal)	`pt-pt`
Punjabi	`pa`
Queretaro Otomi	`otq`
Romanian	`ro`
Russian	`ru`
Samoan	`sm`
Serbian (Cyrillic)	`sr-Cyrl`
Serbian (Latin)	`sr-Latn`
Slovak	`sk`

TEXT LANGUAGE	LANGUAGE CODE
Slovenian	`sl`
Spanish	`es`
Swahili	`sw`
Swedish	`sv`
Tahitian	`ty`
Tamil	`ta`
Telugu	`te`
Thai	`th`
Tigrinya	`ti`
Tongan	`to`
Turkish	`tr`
Ukrainian	`uk`
Urdu	`ur`
Vietnamese	`vi`
Welsh	`cy`
Yucatec Maya	`yua`

Speaker Recognition

Speaker recognition is mostly language agnostic. We built a universal model for text-independent speaker recognition by combining various data sources from multiple languages. We have tuned and evaluated the model on the languages and locales that appear in the following table. See the overview for additional information on Speaker Recognition.

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (US)	`en-US`	yes	yes	yes
Chinese (Mandarin, simplified)	`zh-CN`	n/a	yes	yes
English (Australia)	`en-AU`	n/a	yes	yes

LANGUAGE	LOCALE (BCP-47)	TEXT-DEPENDENT VERIFICATION	TEXT-INDEPENDENT VERIFICATION	TEXT-INDEPENDENT IDENTIFICATION
English (Canada)	`en-CA`	n/a	yes	yes
English (India)	`en-IN`	n/a	yes	yes
English (UK)	`en-GB`	n/a	yes	yes
French (Canada)	`fr-CA`	n/a	yes	yes
French (France)	`fr-FR`	n/a	yes	yes
German (Germany)	`de-DE`	n/a	yes	yes
Italian	`it-IT`	n/a	yes	yes
Japanese	`ja-JP`	n/a	yes	yes
Portuguese (Brazil)	`pt-BR`	n/a	yes	yes
Spanish (Mexico)	`es-MX`	n/a	yes	yes
Spanish (Spain)	`es-ES`	n/a	yes	yes

Custom Keyword and Keyword Verification

The following table outlines supported languages for Custom Keyword and Keyword Verification.

LANGUAGE	LOCALE (BCP-47)	CUSTOM KEYWORD	KEYWORD VERIFICATION
Chinese (Mandarin, Simplified)	zh-CN	Yes	Yes
English (United States)	en-US	Yes	Yes
Japanese (Japan)	ja-JP	No	Yes
Portuguese (Brazil)	pt-BR	No	Yes

Next steps

- Create a free Azure account
- See how to recognize speech in C#

Speech service supported regions

11/23/2021 • 5 minutes to read • Edit Online

The Speech service allows your application to convert audio to text, perform speech translation, and covert text to speech. The service is available in multiple regions with unique endpoints for the Speech SDK and REST APIs.

The Speech portal, where you can perform custom configurations to your speech experience for all regions, is available at speech.microsoft.com.

Keep in mind the following points when considering regions:

- If your application uses a Speech SDK, you provide the region identifier, such as `westus`, when creating a speech configuration. Make sure the region matches the region of your subscription.
- If your application uses one of the Speech service's REST APIs, the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

> **NOTE**
>
> Speech Services doesn't store/process customer data outside the region the customer deploys the service instance in.

Speech SDK

In the Speech SDK, the region is specified as a parameter (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

Speech-to-text, text-to-speech, and translation

The Speech service is available in these regions for **speech recognition**, **text-to-speech**, and **translation**:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`

GEOGRAPHY	REGION	REGION IDENTIFIER
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

If you plan to train a custom model with audio data, use one of the regions with dedicated hardware for faster training. You can use the REST API to copy the fully trained model to another region later.

Intent recognition

Available regions for **intent recognition** via the Speech SDK are the following:

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 in these regions:

GLOBAL REGION	REGION	REGION IDENTIFIER
North America	West US	`westus`
North America	West US 2	`westus2`
North America	East US	`eastus`
North America	East US 2	`eastus2`
North America	West Central US	`westcentralus`
North America	South Central US	`southcentralus`
Europe	West Europe	`westeurope`
Europe	North Europe	`northeurope`
Asia	East Asia	`eastasia`
Asia	Southeast Asia	`southeastasia`
India	Central India	`centralindia`

Speaker Recognition

Available regions for **Speaker Recognition** are the following:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Central India	`centralindia`
Australia	Australia East	`australiaeast`
Europe	North Europe	`northeurope`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	West Europe	`westeurope`
Europe	UK South	`uksouth`

REST APIs

The Speech service also exposes REST endpoints for speech-to-text, text-to-speech and speaker recognition requests.

Speech-to-text

For speech-to-text reference documentation, see Speech-to-text REST API.

The endpoint for the REST API has this format:

```
https://<REGION_IDENTIFIER>.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

> **NOTE**
>
> The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is:
>
> ```
> https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
> ```
>
> .

Text-to-speech

For text-to-speech reference documentation, see Text-to-speech REST API.

Neural and standard voices

Use this table to determine **availability of neural and standard voices** by region/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`
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/v`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`

REGION	ENDPOINT
South Central US	`https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v`
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`

> **TIP**
>
> Voices in preview are only available in these 3 regions: East US, West Europe and Southeast Asia.

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	`https://australiaeast.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Brazil South	`https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
Canada Central	`https://canadacentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Central US	`https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East Asia	`https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US	`https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US 2	`https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
France Central	`https://francecentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
India Central	`https://centralindia.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
Japan East	`https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
Japan West	`https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

REGION	ENDPOINT
Korea Central	`https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
North Central US	`https://northcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
North Europe	`https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
South Central US	`https://southcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
Southeast Asia	`https://southeastasia.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
UK South	`https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West Europe	`https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US	`https://westus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US 2	`https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

Custom Neural Voice

The following table details regional support for Custom Neural Voice features.

FEATURE	SUPPORTED REGIONS
Voice model hosting	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Realtime Characters	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Long Audio Characters	East US, West Europe, Southeast Asia, UK South, India Central
Custom Neural Training	East US, Southeast Asia, UK South

Long audio API

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.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`

Speaker Recognition

For speaker recognition reference documentation, see Speaker Recognition REST API. Available regions are the same as Speaker Recognition SDK.

What is intent recognition?

11/23/2021 • 2 minutes to read • Edit Online

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

- Complete the intent recognition quickstart
- Get a Speech service subscription key for free
- Get the Speech SDK

Get started with intent recognition

11/23/2021 • 38 minutes to read • Edit Online

In this quickstart, you'll use the Speech SDK and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

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

You can view or download all Speech SDK C# Samples on GitHub.

Prerequisites

Before you get started:

- Install the Speech SDK for your development environment and create an empty sample project.

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.

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.

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()`.

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

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeIntentAsync()
        {
                            Console.WriteLine($"CANCELED: Did you update the subscription info?");
                        }
                        break;
                    }
                }
            }

            static async Task Main()
            {
                await RecognizeIntentAsync();
```

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.

```
var config = SpeechConfig.FromSubscription(
```

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

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

Initialize an IntentRecognizer

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

```
// Creates an intent recognizer using microphone as audio input.
                Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
```

Add a LanguageUnderstandingModel and intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer, and add the intents that you want recognized. We're going to use intents from the prebuilt domain for home automation. Insert this code in the using statement from the previous section. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

> **TIP**
>
> If you need help finding this value, see Create a LUIS app for intent recognition.

```
// Creates a Language Understanding model using the app id, and adds specific intents from your model
var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
```

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.

> **NOTE**
>
> Speech SDK only supports LUIS v2.0 endpoints. You must manually modify the v3.0 endpoint URL found in the example query field to use a v2.0 URL pattern. LUIS v2.0 endpoints always follow one of these two patterns:
>
> - `https://{AzureResourceName}.cognitiveservices.azure.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`
> - `https://{Region}.api.cognitive.microsoft.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`

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.

```
// single utterance is determined by listening for silence at the end or until a maximum of 15
```

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:

```
// shot recognition like command or query.
// For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
var result = await recognizer.RecognizeOnceAsync();

// 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.LanguageUnderstandingServiceResponse_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}");
```

Check your code

At this point, your code should look like this:

> **NOTE**
>
> We've added some comments to this version.

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

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeIntentAsync()
        {
            // Creates an instance of a speech config with specified subscription key
            // and service region. Note that in contrast to other services supported by
            // the Cognitive Services Speech SDK, the Language Understanding service
```

```
            // the Cognitive Services Speech SDK, the Language Understanding service
            // requires a specific subscription key from https://www.luis.ai/.
            // The Language Understanding service calls the required key 'endpoint key'.
            // Once you've obtained it, replace with below with your own Language Understanding subscription
key
            // and service region (e.g., "westus").
            // The default language is "en-us".
            var config = SpeechConfig.FromSubscription(
                "YourLanguageUnderstandingSubscriptionKey",
                "YourLanguageUnderstandingServiceRegion");

            // Creates an intent recognizer using microphone as audio input.
            using (var recognizer = new IntentRecognizer(config))
            {
                // Creates a Language Understanding model using the app id, and adds specific intents from
your model
                var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
                recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
                recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
                recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

                // To add all of the possible intents from a LUIS model to the recognizer, uncomment the
line below:
                // recognizer.AddAllIntents(model);

                // 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();

                // 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.LanguageUnderstandingServiceResponse_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;
                }
            }
```

```
        }
    }

    static async Task Main()
    {
        await RecognizeIntentAsync();
```

Build and run your app

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

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

Next steps

Explore C# samples on GitHub See the advanced LUIS sample on GitHub

In this quickstart, you'll use the Speech SDK and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

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

You can view or download all Speech SDK C++ Samples on GitHub.

Prerequisites

Before you get started:

- Install the Speech SDK for your development environment and create an empty sample project.

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.

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.

Open your project in Visual Studio

Next, open your project in Visual Studio.

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

Start with some boilerplate code

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

```
#include "stdafx.h"
// <code>
#include <iostream>
#include <speechapi_cxx.h>

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

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

int wmain()
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and location for your LUIS prediction resource.

> **IMPORTANT**
>
> Your starter key and authoring keys will not work. You must use your prediction key and location that you created earlier. For more information, see Create a LUIS app for intent recognition.

Insert this code in the `recognizeIntent()` method. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location. Use **Region identifier** from region.

> **TIP**
> If you need help finding these values, see Create a LUIS app for intent recognition.

```
auto config = SpeechConfig::FromSubscription(
```

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

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

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. Insert this code in the `recognizeIntent()` method, right below your Speech configuration.

```

```

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer, and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

> **TIP**
> If you need help finding this value, see Create a LUIS app for intent recognition.

```

// Creates a Language Understanding model using the app id, and adds specific intents from your model
auto model = LanguageUnderstandingModel::FromAppId("YourLanguageUnderstandingAppId");
```

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.

> **NOTE**
> Speech SDK only supports LUIS v2.0 endpoints. You must manually modify the v3.0 endpoint URL found in the example query field to use a v2.0 URL pattern. LUIS v2.0 endpoints always follow one of these two patterns:
>
> - `https://{AzureResourceName}.cognitiveservices.azure.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`
> - `https://{Region}.api.cognitive.microsoft.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech. For simplicity we'll wait on the future returned to complete.

Insert this code below your model:

```
// Starts intent recognition, and returns after a single utterance is recognized. The end of a
```

Display the recognition results (or errors)

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

Insert this code below `auto result = recognizer->RecognizeOnceAsync().get();`:

```
// Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
// shot recognition like command or query.
// For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
auto result = recognizer->RecognizeOnceAsync().get();

// Checks result.
if (result->Reason == ResultReason::RecognizedIntent)
{
    cout << "RECOGNIZED: Text=" << result->Text << std::endl;
    cout << "  Intent Id: " << result->IntentId << std::endl;
    cout << "  Intent Service JSON: " << result-
>Properties.GetProperty(PropertyId::LanguageUnderstandingServiceResponse_JsonResult) << std::endl;
}
else if (result->Reason == ResultReason::RecognizedSpeech)
{
    cout << "RECOGNIZED: Text=" << result->Text << " (intent could not be recognized)" << std::endl;
}
else if (result->Reason == ResultReason::NoMatch)
{
    cout << "NOMATCH: Speech could not be recognized." << std::endl;
}
else if (result->Reason == ResultReason::Canceled)
{
    auto cancellation = CancellationDetails::FromResult(result);
    cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

    if (cancellation->Reason == CancellationReason::Error)
```

Check your code

At this point, your code should look like this:

> **NOTE**
>
> We've added some comments to this version.

```
#include "stdafx.h"
// <code>
#include <iostream>
#include <speechapi_cxx.h>

using namespace std;
```

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

void recognizeIntent()
{
    // Creates an instance of a speech config with specified subscription key
    // and service region. Note that in contrast to other services supported by
    // the Cognitive Services Speech SDK, the Language Understanding service
    // requires a specific subscription key from https://www.luis.ai/.
    // The Language Understanding service calls the required key 'endpoint key'.
    // Once you've obtained it, replace with below with your own Language Understanding subscription key
    // and service region (e.g., "westus").
    // The default recognition language is "en-us".
    auto config = SpeechConfig::FromSubscription(
        "YourLanguageUnderstandingSubscriptionKey",
        "YourLanguageUnderstandingServiceRegion");

    // Creates an intent recognizer using microphone as audio input.
    auto recognizer = IntentRecognizer::FromConfig(config);

    // Creates a Language Understanding model using the app id, and adds specific intents from your model
    auto model = LanguageUnderstandingModel::FromAppId("YourLanguageUnderstandingAppId");
    recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
    recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
    recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

    // To add all of the possible intents from a LUIS model to the recognizer, uncomment the line below:
    // recognizer->AddAllIntents(model);

    cout << "Say something...\n";

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

    // Checks result.
    if (result->Reason == ResultReason::RecognizedIntent)
    {
        cout << "RECOGNIZED: Text=" << result->Text << std::endl;
        cout << "  Intent Id: " << result->IntentId << std::endl;
        cout << "  Intent Service JSON: " << result-
>Properties.GetProperty(PropertyId::LanguageUnderstandingServiceResponse_JsonResult) << std::endl;
    }
    else if (result->Reason == ResultReason::RecognizedSpeech)
    {
        cout << "RECOGNIZED: Text=" << result->Text << " (intent could not be recognized)" << std::endl;
    }
    else if (result->Reason == ResultReason::NoMatch)
    {
        cout << "NOMATCH: Speech could not be recognized." << std::endl;
    }
    else if (result->Reason == ResultReason::Canceled)
    {
        auto cancellation = CancellationDetails::FromResult(result);
        cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

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

Build and run your app

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

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

Next steps

Explore C++ samples on GitHub See the advanced LUIS sample on GitHub

In this quickstart, you'll use the Speech SDK and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

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

You can view or download all Speech SDK Java Samples on GitHub.

Prerequisites

Before you get started:

- Install the Speech SDK for your development environment and create an empty sample project.

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.

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.

Open your project

1. Open your preferred IDE.
2. Load your project and open `Main.java`.

Start with some boilerplate code

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

```
package speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.intent.*;

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

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {


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

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

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and location for your LUIS prediction resource.

Insert this code in the try / catch block in `main()`. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location. Use **Region identifier** from region

> **TIP**
> If you need help finding these values, see Create a LUIS app for intent recognition.

```
// Recognizer is created with the default microphone audio input and default language "en-us".
```

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

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

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. Insert this code right below your Speech configuration.

```

```

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer, and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

> **TIP**
> If you need help finding this value, see Create a LUIS app for intent recognition.

```
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.

> **NOTE**
> Speech SDK only supports LUIS v2.0 endpoints. You must manually modify the v3.0 endpoint URL found in the example query field to use a v2.0 URL pattern. LUIS v2.0 endpoints always follow one of these two patterns:
>
> - `https://{AzureResourceName}.cognitiveservices.azure.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`
> - `https://{Region}.api.cognitive.microsoft.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`

Recognize an intent

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

to stop recognizing speech.

Insert this code below your model:

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

Display the recognition results (or errors)

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

Insert this code below your call to `recognizeOnceAsync()`.

```
IntentRecognitionResult result = recognizer.recognizeOnceAsync().get();

// Checks result.
if (result.getReason() == ResultReason.RecognizedIntent) {
    System.out.println("RECOGNIZED: Text=" + result.getText());
    System.out.println("    Intent Id: " + result.getIntentId());
    System.out.println("    Intent Service JSON: " +
result.getProperties().getProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult));
}
else if (result.getReason() == ResultReason.RecognizedSpeech) {
    System.out.println("RECOGNIZED: Text=" + result.getText());
    System.out.println("    Intent not recognized.");
}
else if (result.getReason() == ResultReason.NoMatch) {
    System.out.println("NOMATCH: Speech could not be recognized.");
}
else if (result.getReason() == ResultReason.Canceled) {
    CancellationDetails cancellation = CancellationDetails.fromResult(result);
    System.out.println("CANCELED: Reason=" + cancellation.getReason());

    if (cancellation.getReason() == CancellationReason.Error) {
        System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
        System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
```

Release Resources

It's important to release the speech resources when you're done using them. Insert this code at the end of the try / catch block:

```
    }
}
```

Check your code

At this point, your code should look like this:

> **NOTE**
>
> We've added some comments to this version.

```
package speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.intent.*;
```

```
import com.microsoft.cognitiveservices.speech.intent.*;

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

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {

        // Replace below with with specified subscription key (called 'endpoint key' by the Language
Understanding service)
        String languageUnderstandingSubscriptionKey = "YourLanguageUnderstandingSubscriptionKey";
        // Replace below with your own service region (e.g., "westus").
        String languageUnderstandingServiceRegion = "YourLanguageUnderstandingServiceRegion";

        // Creates an instance of intent recognizer with a given speech configuration.
        // Recognizer is created with the default microphone audio input and default language "en-us".
        try (SpeechConfig config = SpeechConfig.fromSubscription(languageUnderstandingSubscriptionKey,
languageUnderstandingServiceRegion);
             IntentRecognizer recognizer = new IntentRecognizer(config)) {

            // Creates a language understanding model using the app id, and adds specific intents from your
model
            LanguageUnderstandingModel model =
LanguageUnderstandingModel.fromAppId("YourLanguageUnderstandingAppId");
            recognizer.addIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
            recognizer.addIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
            recognizer.addIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

            // To add all of the possible intents from a LUIS model to the recognizer, uncomment the line
below:
            // recognizer.addAllIntents(model);

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

            // Starts recognition. It returns when the first utterance has been recognized.
            IntentRecognitionResult result = recognizer.recognizeOnceAsync().get();

            // Checks result.
            if (result.getReason() == ResultReason.RecognizedIntent) {
                System.out.println("RECOGNIZED: Text=" + result.getText());
                System.out.println("    Intent Id: " + result.getIntentId());
                System.out.println("    Intent Service JSON: " +
result.getProperties().getProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult));
            }
            else if (result.getReason() == ResultReason.RecognizedSpeech) {
                System.out.println("RECOGNIZED: Text=" + result.getText());
                System.out.println("    Intent not recognized.");
            }
            else if (result.getReason() == ResultReason.NoMatch) {
                System.out.println("NOMATCH: Speech could not be recognized.");
            }
            else if (result.getReason() == ResultReason.Canceled) {
                CancellationDetails cancellation = CancellationDetails.fromResult(result);
                System.out.println("CANCELED: Reason=" + cancellation.getReason());

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

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

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

Build and run your app

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

Next steps

Explore Java samples on GitHub See the advanced LUIS sample on GitHub

In this quickstart, you'll use the Speech SDK and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

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

You can view or download all Speech SDK Python Samples on GitHub.

Prerequisites

Before you get started:

- Install the Speech SDK for your development environment and create an empty sample project.

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.

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.

Open your project

1. Open your preferred IDE.
2. Create a new project and create file called `quickstart.py`, then open it.

Start with some boilerplate code

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

```
import azure.cognitiveservices.speech as speechsdk

print("Say something...")
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and location for your LUIS prediction resource.

Insert this code in `quickstart.py`. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location. Use **Region identifier** from region

> **TIP**
> If you need help finding these values, see Create a LUIS app for intent recognition.

```
intent_config = speechsdk.SpeechConfig(
```

This sample constructs the `SpeechConfig` object using LUIS key and region. For a full list of available methods, see SpeechConfig Class.

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

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. Insert this code right below your Speech configuration.

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

> **TIP**
>
> If you need help finding this value, see Create a LUIS app for intent recognition.

```
# set up the intents that are to be recognized. These can be a mix of simple phrases and
# intents specified through a LanguageUnderstanding Model.
model = speechsdk.intent.LanguageUnderstandingModel(app_id="YourLanguageUnderstandingAppId")
intents = [
    (model, "HomeAutomation.TurnOn"),
    (model, "HomeAutomation.TurnOff"),
    ("This is a test.", "test"),
    ("Switch to channel 34.", "34"),
    ("what's the weather like", "weather"),
```

This example uses the `add_intents()` function to add a list of explicitly-defined intents. If you want to add all intents from a model, use `add_all_intents(model)` and pass the model.

> **NOTE**
>
> Speech SDK only supports LUIS v2.0 endpoints. You must manually modify the v3.0 endpoint URL found in the example query field to use a v2.0 URL pattern. LUIS v2.0 endpoints always follow one of these two patterns:
>
> - `https://{AzureResourceName}.cognitiveservices.azure.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`
> - `https://{Region}.api.cognitive.microsoft.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`

Recognize an intent

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

Insert this code below your model.

```
# single utterance is determined by listening for silence at the end or until a maximum of 15
```

Display the recognition results (or errors)

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

Below your call to `recognize_once()`, add this code.

```
# shot recognition like command or query.
# For long-running multi-utterance recognition, use start_continuous_recognition() instead.
intent_result = intent_recognizer.recognize_once()

# Check the results
if intent_result.reason == speechsdk.ResultReason.RecognizedIntent:
    print("Recognized: \"{}\" with intent id `{}`".format(intent_result.text, intent_result.intent_id))
elif intent_result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("Recognized: {}".format(intent_result.text))
elif intent_result.reason == speechsdk.ResultReason.NoMatch:
```

Check your code

At this point, your code should look like this.

> **NOTE**
>
> We've added some comments to this version.

```
import azure.cognitiveservices.speech as speechsdk

print("Say something...")

"""performs one-shot intent recognition from input from the default microphone"""

# Set up the config for the intent recognizer (remember that this uses the Language Understanding key, not
the Speech Services key)!
intent_config = speechsdk.SpeechConfig(
    subscription="YourLanguageUnderstandingSubscriptionKey",
    region="YourLanguageUnderstandingServiceRegion")

# Set up the intent recognizer
intent_recognizer = speechsdk.intent.IntentRecognizer(speech_config=intent_config)

# set up the intents that are to be recognized. These can be a mix of simple phrases and
# intents specified through a LanguageUnderstanding Model.
model = speechsdk.intent.LanguageUnderstandingModel(app_id="YourLanguageUnderstandingAppId")
intents = [
    (model, "HomeAutomation.TurnOn"),
    (model, "HomeAutomation.TurnOff"),
    ("This is a test.", "test"),
    ("Switch to channel 34.", "34"),
    ("what's the weather like", "weather"),
]
intent_recognizer.add_intents(intents)

# To add all of the possible intents from a LUIS model to the recognizer, uncomment the line below:
# intent_recognizer.add_all_intents(model);

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

# Check the results
if intent_result.reason == speechsdk.ResultReason.RecognizedIntent:
    print("Recognized: \"{}\" with intent id `{}`".format(intent_result.text, intent_result.intent_id))
elif intent_result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("Recognized: {}".format(intent_result.text))
elif intent_result.reason == speechsdk.ResultReason.NoMatch:
```

Build and run your app

Run the sample from the console or in your IDE:

```
python quickstart.py
```

The next 15 seconds of speech input from your microphone will be recognized and logged in the console window.

Next steps

Explore Python samples on GitHub See the advanced LUIS sample on GitHub

In this quickstart, you'll use the Speech SDK and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

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

You can view or download all Speech SDK JavaScript Samples on GitHub.

Prerequisites

Before you get started:

- Install the Speech SDK for your development environment and create an empty sample project.

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.

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.

Start with some boilerplate code

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

```
<!DOCTYPE html>
<html>
<head>
<title>Microsoft Cognitive Services Speech SDK JavaScript Quickstart</title>
<meta charset="utf-8" />
</head>
<body style="font-family:'Helvetica Neue',Helvetica,Arial,sans-serif; font-size:13px;">
</body>
</html>
```

Add UI Elements

Now we'll add some basic UI for input boxes, reference the Speech SDK's JavaScript, and grab an authorization token if available.

```
<body style="font-family:'Helvetica Neue',Helvetica,Arial,sans-serif; font-size:13px;">
  <div id="content" style="display:none">
    <table width="100%">
      <tr>
        <td></td>
        <td><h1 style="font-weight:500;">Microsoft Cognitive Services Speech SDK JavaScript Quickstart</h1>
</td>
      </tr>
      <tr>
        <td align="right"><a href="https://docs.microsoft.com/azure/cognitive-services/speech-service/get-
started" target="_blank">Subscription</a>:</td>
        <td><input id="subscriptionKey" type="text" size="40" value="subscription"></td>
      </tr>
      <tr>
        <td align="right">Region</td>
        <td><input id="serviceRegion" type="text" size="40" value="YourServiceRegion"></td>
      </tr>
      <tr>
        <td align="right">Application ID:</td>
        <td><input id="appId" type="text" size="60" value="YOUR_LANGUAGE_UNDERSTANDING_APP_ID"></td>
      </tr>
      <tr>
        <td></td>
        <td><button id="startIntentRecognizeAsyncButton">Start Intent Recognition</button></td>
      </tr>
      <tr>
        <td align="right" valign="top">Input Text</td>
        <td><textarea id="phraseDiv" style="display: inline-block;width:500px;height:200px"></textarea></td>
      </tr>
      <tr>
        <td align="right" valign="top">Result</td>
        <td><textarea id="statusDiv" style="display: inline-block;width:500px;height:100px"></textarea></td>
      </tr>
    </table>
  </div>

  <script src="microsoft.cognitiveservices.speech.sdk.bundle.js"></script>

  <script>
  // Note: Replace the URL with a valid endpoint to retrieve
  //       authorization tokens for your subscription.
  var authorizationEndpoint = "token.php";

  function RequestAuthorizationToken() {
    if (authorizationEndpoint) {
      var a = new XMLHttpRequest();
      a.open("GET", authorizationEndpoint);
      a.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      a.send("");
      a.onload = function() {
```

```
      var token = JSON.parse(atob(this.responseText.split(".")[1]));
      serviceRegion.value = token.region;
      authorizationToken = this.responseText;
      subscriptionKey.disabled = true;
      subscriptionKey.value = "using authorization token (hit F5 to refresh)";
      console.log("Got an authorization token: " + token);
        }
      }
    }
    </script>

    <script>
      // status fields and start button in UI
      var phraseDiv;
      var statusDiv;
      var startIntentRecognizeAsyncButton;

      // subscription key, region, and appId for LUIS services.
      var subscriptionKey, serviceRegion, appId;
      var authorizationToken;
      var SpeechSDK;
      var recognizer;

      document.addEventListener("DOMContentLoaded", function () {
        startIntentRecognizeAsyncButton = document.getElementById("startIntentRecognizeAsyncButton");
        subscriptionKey = document.getElementById("subscriptionKey");
        serviceRegion = document.getElementById("serviceRegion");
        appId = document.getElementById("appId");
        phraseDiv = document.getElementById("phraseDiv");
        statusDiv = document.getElementById("statusDiv");

        startIntentRecognizeAsyncButton.addEventListener("click", function () {
          startIntentRecognizeAsyncButton.disabled = true;
          phraseDiv.innerHTML = "";
          statusDiv.innerHTML = "";
        });

        if (!!window.SpeechSDK) {
          SpeechSDK = window.SpeechSDK;
          startIntentRecognizeAsyncButton.disabled = false;

          document.getElementById('content').style.display = 'block';
          document.getElementById('warning').style.display = 'none';

          // in case we have a function for getting an authorization token, call it.
          if (typeof RequestAuthorizationToken === "function") {
            RequestAuthorizationToken();
          }
        }
      });
    </script>
```

Create a Speech configuration

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

> **NOTE**
>
> The Speech SDK will default to recognizing using en-us for the language, see Specify source language for speech to text for information on choosing the source language.

```
        // if we got an authorization token, use the token. Otherwise use the provided subscription key
        var speechConfig;
        if (authorizationToken) {
          speechConfig = SpeechSDK.SpeechConfig.fromAuthorizationToken(authorizationToken,
serviceRegion.value);
        } else {
          if (subscriptionKey.value === "" || subscriptionKey.value === "subscription") {
            alert("Please enter your Microsoft Cognitive Services Speech subscription key!");
            return;
          }
          startIntentRecognizeAsyncButton.disabled = false;
          speechConfig = SpeechSDK.SpeechConfig.fromSubscription(subscriptionKey.value,
serviceRegion.value);
        }

        speechConfig.speechRecognitionLanguage = "en-US";
```

Create an Audio configuration

Now, you need to create an `AudioConfig` object that points to your input device. Insert this code in the `startIntentRecognizeAsyncButton.addEventListener()` method, right below your Speech configuration.

```
        var audioConfig = SpeechSDK.AudioConfig.fromDefaultMicrophoneInput();
```

Initialize a IntentRecognizer

Now, let's create the `IntentRecognizer` object using the `SpeechConfig` and `AudioConfig` objects created earlier. Insert this code in the `startIntentRecognizeAsyncButton.addEventListener()` method.

```
        recognizer = new SpeechSDK.IntentRecognizer(speechConfig, audioConfig);
```

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

```
        if (appId.value !== "" && appId.value !== "YOUR_LANGUAGE_UNDERSTANDING_APP_ID") {
          var lm = SpeechSDK.LanguageUnderstandingModel.fromAppId(appId.value);

          recognizer.addAllIntents(lm);
        }
```

NOTE

Speech SDK only supports LUIS v2.0 endpoints. You must manually modify the v3.0 endpoint URL found in the example query field to use a v2.0 URL pattern. LUIS v2.0 endpoints always follow one of these two patterns:

- `https://{AzureResourceName}.cognitiveservices.azure.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`
- `https://{Region}.api.cognitive.microsoft.com/luis/v2.0/apps/{app-id}?subscription-key={subkey}&verbose=true&q=`

Recognize an intent

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

Insert this code below the model addition:

```
        recognizer.recognizeOnceAsync(
          function (result) {
            window.console.log(result);

            phraseDiv.innerHTML = result.text + "\r\n";

            statusDiv.innerHTML += "(continuation) Reason: " + SpeechSDK.ResultReason[result.reason];
            switch (result.reason) {
              case SpeechSDK.ResultReason.RecognizedSpeech:
                statusDiv.innerHTML += " Text: " + result.text;
                break;
              case SpeechSDK.ResultReason.RecognizedIntent:
                statusDiv.innerHTML += " Text: " + result.text + " IntentId: " + result.intentId;

                // The actual JSON returned from Language Understanding is a bit more complex to get to, but
it is available for things like
                // the entity name and type if part of the intent.
                statusDiv.innerHTML += " Intent JSON: " +
result.properties.getProperty(SpeechSDK.PropertyId.LanguageUnderstandingServiceResponse_JsonResult);
                phraseDiv.innerHTML +=
result.properties.getProperty(SpeechSDK.PropertyId.LanguageUnderstandingServiceResponse_JsonResult) +
"\r\n";
                break;
              case SpeechSDK.ResultReason.NoMatch:
                var noMatchDetail = SpeechSDK.NoMatchDetails.fromResult(result);
                statusDiv.innerHTML += " NoMatchReason: " + SpeechSDK.NoMatchReason[noMatchDetail.reason];
                break;
              case SpeechSDK.ResultReason.Canceled:
                var cancelDetails = SpeechSDK.CancellationDetails.fromResult(result);
                statusDiv.innerHTML += " CancellationReason: " +
SpeechSDK.CancellationReason[cancelDetails.reason];

              if (cancelDetails.reason === SpeechSDK.CancellationReason.Error) {
                statusDiv.innerHTML += ": " + cancelDetails.errorDetails;
              }
            break;
            }
            statusDiv.innerHTML += "\r\n";
            startIntentRecognizeAsyncButton.disabled = false;
          },
          function (err) {
            window.console.log(err);

            phraseDiv.innerHTML += "ERROR: " + err;
            startIntentRecognizeAsyncButton.disabled = false;
          });
```

Check your code

```
<!DOCTYPE html>
<html>
<head>
  <title>Microsoft Cognitive Services Speech SDK JavaScript Quickstart</title>
  <meta charset="utf-8" />
</head>
<body style="font-family:'Helvetica Neue',Helvetica,Arial,sans-serif; font-size:13px;">
```

```
  <div id="warning">
    <h1 style="font-weight:500;">Speech Recognition Speech SDK not found
(microsoft.cognitiveservices.speech.sdk.bundle.js missing).</h1>
  </div>

  <div id="content" style="display:none">
    <table width="100%">
      <tr>
        <td></td>
        <td><h1 style="font-weight:500;">Microsoft Cognitive Services Speech SDK JavaScript Quickstart</h1>
</td>
      </tr>
      <tr>
        <td align="right"><a href="https://docs.microsoft.com/azure/cognitive-services/speech-
service/quickstarts/intent-recognition?pivots=programming-language-csharp#create-a-luis-app-for-intent-
recognition" target="_blank">LUIS Primary Key</a>:</td>
        <td><input id="subscriptionKey" type="text" size="40" value="subscription"></td>
      </tr>
      <tr>
        <td align="right">LUIS Location</td>
        <td><input id="serviceRegion" type="text" size="40" value="YourServiceRegion"></td>
      </tr>
      <tr>
        <td align="right">LUIS App ID:</td>
        <td><input id="appId" type="text" size="60" value="YOUR_LANGUAGE_UNDERSTANDING_APP_ID"></td>
      </tr>
      <tr>
        <td></td>
        <td><button id="startIntentRecognizeAsyncButton">Start Intent Recognition</button></td>
      </tr>
      <tr>
        <td align="right" valign="top">Input Text</td>
        <td><textarea id="phraseDiv" style="display: inline-block;width:500px;height:200px"></textarea></td>
      </tr>
      <tr>
        <td align="right" valign="top">Result</td>
        <td><textarea id="statusDiv" style="display: inline-block;width:500px;height:100px"></textarea></td>
      </tr>
    </table>
  </div>

  <!-- Speech SDK reference sdk. -->
  <script src="https://aka.ms/csspeech/jsbrowserpackageraw"></script>

  <!-- Speech SDK USAGE -->
  <script>
    // status fields and start button in UI
    var phraseDiv;
    var statusDiv;
    var startIntentRecognizeAsyncButton;

    // subscription key and region for speech services.
    var subscriptionKey, serviceRegion, appId;
    var SpeechSDK;
    var recognizer;

    document.addEventListener("DOMContentLoaded", function () {

      startIntentRecognizeAsyncButton = document.getElementById("startIntentRecognizeAsyncButton");
      subscriptionKey = document.getElementById("subscriptionKey");
      serviceRegion = document.getElementById("serviceRegion");
      appId = document.getElementById("appId");
      phraseDiv = document.getElementById("phraseDiv");
      statusDiv = document.getElementById("statusDiv");

      startIntentRecognizeAsyncButton.addEventListener("click", function () {
        startIntentRecognizeAsyncButton.disabled = true;
        phraseDiv.innerHTML = "";
        statusDiv.innerHTML = "";
```

```
        statusDiv.innerHTML = "";

        let audioConfig = SpeechSDK.AudioConfig.fromDefaultMicrophoneInput();
        if (subscriptionKey.value === "" || subscriptionKey.value === "subscription") {
          alert("Please enter your Microsoft Cognitive Services Speech subscription key!");
          startIntentRecognizeAsyncButton.disabled = false;
          return;
        }
        var speechConfig = SpeechSDK.SpeechConfig.fromSubscription(subscriptionKey.value,
serviceRegion.value);

        speechConfig.speechRecognitionLanguage = "en-US";
        recognizer = new SpeechSDK.IntentRecognizer(speechConfig, audioConfig);

        // Set up a Language Understanding Model from Language Understanding Intelligent Service (LUIS).
        // See https://www.luis.ai/home for more information on LUIS.
        if (appId.value !== "" && appId.value !== "YOUR_LANGUAGE_UNDERSTANDING_APP_ID") {
          var lm = SpeechSDK.LanguageUnderstandingModel.fromAppId(appId.value);

          recognizer.addAllIntents(lm);
        }

        recognizer.recognizeOnceAsync(
          function (result) {
            window.console.log(result);
            phraseDiv.innerHTML = result.text + "\r\n";

            statusDiv.innerHTML += "(continuation) Reason: " + SpeechSDK.ResultReason[result.reason];
            switch (result.reason) {

              case SpeechSDK.ResultReason.RecognizedSpeech:
                statusDiv.innerHTML += " Text: " + result.text;
                break;

              case SpeechSDK.ResultReason.RecognizedIntent:
                statusDiv.innerHTML += " Text: " + result.text + " IntentId: " + result.intentId;

                // The actual JSON returned from Language Understanding is a bit more complex to get to, but
it is available for things like
                // the entity name and type if part of the intent.
                statusDiv.innerHTML += " Intent JSON: " +
result.properties.getProperty(SpeechSDK.PropertyId.LanguageUnderstandingServiceResponse_JsonResult);
                phraseDiv.innerHTML +=
result.properties.getProperty(SpeechSDK.PropertyId.LanguageUnderstandingServiceResponse_JsonResult) +
"\r\n";
                break;

              case SpeechSDK.ResultReason.NoMatch:
                var noMatchDetail = SpeechSDK.NoMatchDetails.fromResult(result);
                statusDiv.innerHTML += " NoMatchReason: " + SpeechSDK.NoMatchReason[noMatchDetail.reason];
                break;

              case SpeechSDK.ResultReason.Canceled:
                var cancelDetails = SpeechSDK.CancellationDetails.fromResult(result);
                statusDiv.innerHTML += " CancellationReason: " +
SpeechSDK.CancellationReason[cancelDetails.reason];

                if (cancelDetails.reason === SpeechSDK.CancellationReason.Error) {
                  statusDiv.innerHTML += ": " + cancelDetails.errorDetails;
                }
                break;
            }
            statusDiv.innerHTML += "\r\n";
            startIntentRecognizeAsyncButton.disabled = false;
          },
          function (err) {
            window.console.log(err);

            phraseDiv.innerHTML += "ERROR: " + err;
            startIntentRecognizeAsyncButton.disabled = false;
```

```
            startIntentRecognizeAsyncButton.disabled = false;
        });
      });

      if (!!window.SpeechSDK) {
        SpeechSDK = window.SpeechSDK;
        startIntentRecognizeAsyncButton.disabled = false;

        document.getElementById('content').style.display = 'block';
        document.getElementById('warning').style.display = 'none';
      }
    });

  </script>
</body>
</html>
```

Create the token source (optional)

In case you want to host the web page on a web server, you can optionally provide a token source for your demo application. That way, your subscription key will never leave your server while allowing users to use speech capabilities without entering any authorization code themselves.

Create a new file named `token.php`. In this example we assume your web server supports the PHP scripting language with curl enabled. Enter the following code:

```
<?php
header('Access-Control-Allow-Origin: ' . $_SERVER['SERVER_NAME']);

// Replace with your own subscription key and service region (e.g., "westus").
$subscriptionKey = 'YourSubscriptionKey';
$region = 'YourServiceRegion';

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, 'https://' . $region . '.api.cognitive.microsoft.com/sts/v1.0/issueToken');
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, '{}');
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: application/json', 'Ocp-Apim-Subscription-Key: ' .
$subscriptionKey));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
echo curl_exec($ch);
?>
```

> **NOTE**
>
> Authorization tokens only have a limited lifetime. This simplified example does not show how to refresh authorization tokens automatically. As a user, you can manually reload the page or hit F5 to refresh.

Build and run the sample locally

To launch the app, double-click on the index.html file or open index.html with your favorite web browser. It will present a simple GUI allowing you to enter your LUIS key, LUIS region, and LUIS Application ID. Once those fields have been entered, you can click the appropriate button to trigger a recognition using the microphone.

> **NOTE**
>
> This method doesn't work on the Safari browser. On Safari, the sample web page needs to be hosted on a web server; Safari doesn't allow websites loaded from a local file to use the microphone.

Build and run the sample via a web server

To launch your app, open your favorite web browser and point it to the public URL that you host the folder on, enter your LUIS region as well as your LUIS Application ID, and trigger a recognition using the microphone. If configured, it will acquire a token from your token source and begin recognizing spoken commands.

Next steps

Explore JavaScript samples on GitHub See the advanced LUIS sample on GitHub

How to recognize intents from speech using the Speech SDK for C#

11/23/2021 • 12 minutes to read • Edit Online

The Cognitive Services Speech SDK integrates with the Language Understanding service (LUIS) to provide **intent recognition**. An intent is something the user wants to do: book a flight, check the weather, or make a call. The user can use whatever terms feel natural. Using machine learning, LUIS maps user requests to the intents you've defined.

> **NOTE**
>
> A LUIS application defines the intents and entities you want to recognize. It's separate from the C# application that uses the Speech service. In this article, "app" means the LUIS app, while "application" means the C# code.

In this guide, you use the Speech SDK to develop a C# console application that derives intents from user utterances through your device's microphone. You'll learn how to:

- Create a Visual Studio project referencing the Speech SDK NuGet package
- Create a speech configuration and get an intent recognizer
- Get the model for your LUIS app and add the intents you need
- Specify the language for speech recognition
- Recognize speech from a file
- Use asynchronous, event-driven continuous recognition

Prerequisites

Be sure you have the following items before you begin this guide:

- A LUIS account. You can get one for free through the LUIS portal.
- Visual Studio 2019 (any edition).

LUIS and speech

LUIS integrates with the Speech service to recognize intents from speech. You don't need a Speech service subscription, just LUIS.

LUIS uses three kinds of keys:

KEY TYPE	PURPOSE
Authoring	Lets you create and modify LUIS apps programmatically
Starter	Lets you test your LUIS application using text only
Endpoint	Authorizes access to a particular LUIS app

For this guide, you need the endpoint key type. This guide uses the example Home Automation LUIS app, which you can create by following the Use prebuilt Home automation app quickstart. If you've created a LUIS app of your own, you can use it instead.

When you create a LUIS app, LUIS automatically generates a starter key so you can test the app using text queries. This key doesn't enable the Speech service integration and won't work with this guide. Create a LUIS resource in the Azure dashboard and assign it to the LUIS app. You can use the free subscription tier for this guide.

After you create the LUIS resource in the Azure dashboard, log into the LUIS portal, choose your application on the **My Apps** page, then switch to the app's **Manage** page. Finally, select **Keys and Endpoints** in the sidebar.

On the **Keys and Endpoint settings** page:

1. Scroll down to the **Resources and Keys** section and select **Assign resource**.
2. In the **Assign a key to your app** dialog box, make the following changes:
   - Under **Tenant**, choose **Microsoft**.
   - Under **Subscription Name**, choose the Azure subscription that contains the LUIS resource you want to use.
   - Under **Key**, choose the LUIS resource that you want to use with the app.

   In a moment, the new subscription appears in the table at the bottom of the page.
3. Select the icon next to a key to copy it to the clipboard. (You may use either key.)

Create a speech project in Visual Studio

To create a Visual Studio project for Windows development, you need to create the project, set up Visual Studio for .NET desktop development, install the Speech SDK, and choose the target architecture.

Create the project and add the workload

To start, create the project in Visual Studio, and make sure that Visual Studio is set up for .NET desktop development:

1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.

3. In the **Create a new project** window, choose **Console App (.NET Framework)**, and then select **Next**.

4. In the **Configure your new project** window, enter *helloworld* in **Project name**, choose or create the directory path in **Location**, and then select **Create**.

5. From the Visual Studio menu bar, select **Tools** > **Get Tools and Features**, which opens Visual Studio Installer and displays the **Modifying** dialog box.

6. Check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, select the check box next to it, and then select **Modify** to start the installation. It may take a few minutes to download and install.

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

7. Close Visual Studio Installer.

Install the Speech SDK

The next step is to install the Speech SDK NuGet package, so you can reference it in the code.

1. In the Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show the NuGet Package Manager.

2. In the upper-right corner, find the **Package Source** drop-down box, and make sure that **nuget.org** is

selected.

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

4. In the search box, type *Microsoft.CognitiveServices.Speech* and select **Enter**.

5. From the search results, select the **Microsoft.CognitiveServices.Speech** package, and then select **Install** to install the latest stable version.

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

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

Choose the target architecture

Now, to build and run the console application, create a platform configuration matching your computer's architecture.

1. From the menu bar, select **Build** > **Configuration Manager**. The **Configuration Manager** dialog box appears.

2. In the **Active solution platform** drop-down box, select **New**. The **New Solution Platform** dialog box appears.

3. In the **Type or select the new platform** drop-down box:

   - If you're running 64-bit Windows, select **x64**.
   - If you're running 32-bit Windows, select **x86**.

4. Select **OK** and then **Close**.

Add the code

Next, you add code to the project.

1. From **Solution Explorer**, open the file **Program.cs**.

2. Replace the block of `using` statements at the beginning of the file with the following declarations:

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

3. Replace the provided `Main()` method, with the following asynchronous equivalent:

```
public static async Task Main()
{
    await RecognizeIntentAsync();
    Console.WriteLine("Please press Enter to continue.");
    Console.ReadLine();
}
```

4. Create an empty asynchronous method `RecognizeIntentAsync()`, as shown here:

```
static async Task RecognizeIntentAsync()
{
}
```

5. In the body of this new method, add this code:

```
// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
// The default language is "en-us".
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using microphone as audio input.
using (var recognizer = new IntentRecognizer(config))
{
    // Creates a Language Understanding model using the app id, and adds specific intents from your
model
    var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

    // Starts recognizing.
    Console.WriteLine("Say something...");

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

    // Checks result.
    if (result.Reason == ResultReason.RecognizedIntent)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"    Intent Id: {result.IntentId}.");
        Console.WriteLine($"    Language Understanding JSON:
{result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)}.");
    }
    else if (result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"    Intent not recognized.");
    }
    else if (result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
    else if (result.Reason == ResultReason.Canceled)
    {
        var cancellation = CancellationDetails.FromResult(result);
        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

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

6. Replace the placeholders in this method with your LUIS subscription key, region, and app ID as follows.

PLACEHOLDER	REPLACE WITH
`YourLanguageUnderstandingSubscriptionKey`	Your LUIS endpoint key. Again, you must get this item from your Azure dashboard, not a "starter key." You can find it on your app's **Keys and Endpoints** page (under **Manage**) in the LUIS portal.
`YourLanguageUnderstandingServiceRegion`	The short identifier for the region your LUIS subscription is in, such as `westus` for West US. See Regions.
`YourLanguageUnderstandingAppId`	The LUIS app ID. You can find it on your app's **Settings** page in the LUIS portal.

With these changes made, you can build (**Control+Shift+B**) and run (**F5**) the application. When you're prompted, try saying "Turn off the lights" into your PC's microphone. The application displays the result in the console window.

The following sections include a discussion of the code.

Create an intent recognizer

First, you need to create a speech configuration from your LUIS endpoint key and region. You can use speech configurations to create recognizers for the various capabilities of the Speech SDK. The speech configuration has multiple ways to specify the subscription you want to use; here, we use `FromSubscription`, which takes the subscription key and region.

> **NOTE**
> Use the key and region of your LUIS subscription, not a Speech service subscription.

Next, create an intent recognizer using `new IntentRecognizer(config)`. Since the configuration already knows which subscription to use, you don't need to specify the subscription key and endpoint again when creating the recognizer.

Import a LUIS model and add intents

Now import the model from the LUIS app using `LanguageUnderstandingModel.FromAppId()` and add the LUIS intents that you wish to recognize via the recognizer's `AddIntent()` method. These two steps improve the accuracy of speech recognition by indicating words that the user is likely to use in their requests. You don't have to add all the app's intents if you don't need to recognize them all in your application.

To add intents, you must provide three arguments: the LUIS model (which has been created and is named `model`), the intent name, and an intent ID. The difference between the ID and the name is as follows.

`ADDINTENT()` ARGUMENT	PURPOSE
`intentName`	The name of the intent as defined in the LUIS app. This value must match the LUIS intent name exactly.
`intentID`	An ID assigned to a recognized intent by the Speech SDK. This value can be whatever you like; it doesn't need to correspond to the intent name as defined in the LUIS app. If multiple intents are handled by the same code, for instance, you could use the same ID for them.

The Home Automation LUIS app has two intents: one for turning on a device, and another for turning off a device. The lines below add these intents to the recognizer; replace the three `AddIntent` lines in the `RecognizeIntentAsync()` method with this code.

```
recognizer.AddIntent(model, "HomeAutomation.TurnOff", "off");
recognizer.AddIntent(model, "HomeAutomation.TurnOn", "on");
```

Instead of adding individual intents, you can also use the `AddAllIntents` method to add all the intents in a model to the recognizer.

Start recognition

With the recognizer created and the intents added, recognition can begin. The Speech SDK supports both at-start and continuous recognition.

RECOGNITION MODE	METHODS TO CALL	RESULT
At-start	`RecognizeOnceAsync()`	Returns the recognized intent, if any, after one utterance.
Continuous	`StartContinuousRecognitionAsync()` `StopContinuousRecognitionAsync()`	Recognizes multiple utterances; emits events (for example, `IntermediateResultReceived`) when results are available.

The application uses at-start mode and so calls `RecognizeOnceAsync()` to begin recognition. The result is an `IntentRecognitionResult` object containing information about the intent recognized. You extract the LUIS JSON response by using the following expression:

```
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)
```

The application doesn't parse the JSON result. It only displays the JSON text in the console window.

```
Say something...
RECOGNIZED: Text=Hey turn off the light.
    Intent Id: off.
    Language Understanding JSON: {
  "query": "Hey turn off the light",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOff",
    "score": 0.997960269
  },
  "entities": [
    {
      "entity": "light",
      "type": "HomeAutomation.DeviceType",
      "startIndex": 17,
      "endIndex": 21,
      "resolution": {
        "values": [
          "light"
        ]
      }
    }
  ]
}.
Please press Enter to continue.
```

Specify recognition language

By default, LUIS recognizes intents in US English (`en-us`). By assigning a locale code to the `SpeechRecognitionLanguage` property of the speech configuration, you can recognize intents in other languages. For example, add `config.SpeechRecognitionLanguage = "de-de";` in our application before creating the recognizer to recognize intents in German. For more information, see LUIS language support.

Continuous recognition from a file

The following code illustrates two additional capabilities of intent recognition using the Speech SDK. The first, previously mentioned, is continuous recognition, where the recognizer emits events when results are available. These events can then be processed by event handlers that you provide. With continuous recognition, you call the recognizer's `StartContinuousRecognitionAsync()` method to start recognition instead of `RecognizeOnceAsync()`.

The other capability is reading the audio containing the speech to be processed from a WAV file. Implementation involves creating an audio configuration that can be used when creating the intent recognizer. The file must be single-channel (mono) with a sampling rate of 16 kHz.

To try out these features, delete or comment out the body of the `RecognizeIntentAsync()` method, and add the following code in its place.

```
// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using file as audio input.
// Replace with your own audio file name.
using (var audioInput = AudioConfig.FromWavFileInput("whatstheweatherlike.wav"))
{
    using (var recognizer = new IntentRecognizer(config, audioInput))
    {
        // The TaskCompletionSource to stop recognition.
        var stopRecognition = new TaskCompletionSource<int>();

        // Creates a Language Understanding model using the app id, and adds specific intents from your
model
        var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

        // Subscribes to events.
        recognizer.Recognizing += (s, e) => {
            Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
        };

        recognizer.Recognized += (s, e) => {
            if (e.Result.Reason == ResultReason.RecognizedIntent)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"    Intent Id: {e.Result.IntentId}.");
                Console.WriteLine($"    Language Understanding JSON:
{e.Result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)}.");
            }
            else if (e.Result.Reason == ResultReason.RecognizedSpeech)
            {
```

```
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"    Intent not recognized.");
            }
            else if (e.Result.Reason == ResultReason.NoMatch)
            {
                Console.WriteLine($"NOMATCH: Speech could not be recognized.");
            }
        };

        recognizer.Canceled += (s, e) => {
            Console.WriteLine($"CANCELED: Reason={e.Reason}");

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

            stopRecognition.TrySetResult(0);
        };

        recognizer.SessionStarted += (s, e) => {
            Console.WriteLine("\n    Session started event.");
        };

        recognizer.SessionStopped += (s, e) => {
            Console.WriteLine("\n    Session stopped event.");
            Console.WriteLine("\nStop recognition.");
            stopRecognition.TrySetResult(0);
        };


        // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
        Console.WriteLine("Say something...");
        await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

        // Waits for completion.
        // Use Task.WaitAny to keep the task rooted.
        Task.WaitAny(new[] { stopRecognition.Task });

        // Stops recognition.
        await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
    }
}
```

Revise the code to include your LUIS endpoint key, region, and app ID and to add the Home Automation intents, as before. Change `whatstheweatherlike.wav` to the name of your recorded audio file. Then build, copy the audio file to the build directory, and run the application.

For example, if you say "Turn off the lights", pause, and then say "Turn on the lights" in your recorded audio file, console output similar to the following may appear:

```
Say something...

    Session started event.
RECOGNIZING: Text=turn
RECOGNIZING: Text=turn off
RECOGNIZING: Text=turn off the
RECOGNIZING: Text=turn off the light
RECOGNIZING: Text=turn off the lights
RECOGNIZED: Text=Turn off the lights.
    Intent Id: off.
    Language Understanding JSON: {
  "query": "turn off the lights",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOff",
    "score": 0.997679055
  },
  "entities": [
    {
      "entity": "lights",
      "type": "HomeAutomation.DeviceType",
      "startIndex": 13,
      "endIndex": 18,
      "resolution": {
        "values": [
          "light"
        ]
      }
    }
  ]
}.
RECOGNIZING: Text=turn
RECOGNIZING: Text=turn on
RECOGNIZING: Text=turn on the
RECOGNIZING: Text=turn on the light
RECOGNIZING: Text=turn on the lights
RECOGNIZED: Text=Turn on the lights.
    Intent Id: on.
    Language Understanding JSON: {
  "query": "turn on the lights",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOn",
    "score": 0.987454832
  },
  "entities": [
    {
      "entity": "lights",
      "type": "HomeAutomation.DeviceType",
      "startIndex": 12,
      "endIndex": 17,
      "resolution": {
        "values": [
          "light"
        ]
      }
    }
  ]
}.
NOMATCH: Speech could not be recognized.
NOMATCH: Speech could not be recognized.
CANCELED: Reason=EndOfStream

    Session stopped event.

Stop recognition.
Please press Enter to continue.
```

Sample source code

The Speech SDK team actively maintains a large set of examples in an open-source repository. For the sample source code repository, visit the Microsoft Cognitive Services Speech SDK on GitHub . There are samples for C#, C++, Java, Python, Objective-C, Swift, JavaScript, UWP, Unity, and Xamarin.

GitHub

Look for the code from this article in the **samples/csharp/sharedcontent/console** folder.

Next steps

Quickstart: Recognize speech from a microphone

How to use simple language pattern matching with the C++ Speech SDK

11/23/2021 • 8 minutes to read • Edit Online

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 file
- Use asynchronous, event-driven continuous recognition

When should you use this?

Use this sample code if:

- You are only interested in matching very strictly what the user said. These patterns match more aggressively than LUIS.
- You do not have access to a LUIS app, but still want intents. This can be helpful since it is embedded within the SDK.
- You cannot or do not want to create a LUIS app but you still want some voice commanding capability.

If you do not have access to a LUIS app, but still want intents, this can be helpful since it is embedded within the SDK.

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. For example:

```
Take me to the {floorName}
```

This defines an Entity with the ID "floorName" which is case sensitive.

All other special characters and punctuation will be ignored.

Intents will be added using calls to the IntentRecognizer->AddIntent() API.

Create a speech project in Visual Studio

To create a Visual Studio project for C++ desktop development, you need to set up Visual Studio development options, create the project, select the target architecture, and install the Speech SDK.

Set up Visual Studio development options

To start, make sure you're set up correctly in Visual Studio for C++ desktop development:

1. Open Visual Studio 2019 to display the **Start** window.

2. Select **Continue without code** to go to the Visual Studio IDE.

3. From the Visual Studio menu bar, select **Tools** > **Get Tools and Features** to open Visual Studio Installer and view the **Modifying** dialog box.

4. In the **Workloads** tab, under **Windows**, find the **Desktop development with C++** workload. If the check box next to that workload isn't already selected, select it.

5. In the **Individual components** tab, find the **NuGet package manager** check box. If the check box isn't already selected, select it.

6. Select the button in the corner labeled either **Close** or **Modify**. (The button name varies depending on whether you selected any features for installation.) If you select **Modify**, installation begins, which may take a while.

7. Close Visual Studio Installer.

Create the project and select the target architecture

Next, create your project:

1. In the Visual Studio menu bar, choose **File** > **New** > **Project** to display the **Create a new project** window.

2. Find and select **Console App**. Make sure that you select the C++ version of this project type (as opposed to C# or Visual Basic).

3. Select **Next** to display the **Configure your new project** screen.

4. In **Project name**, enter `helloworld`.

5. In **Location**, navigate to and select or create the folder to save your project in.

Now select your target platform architecture. In the Visual Studio toolbar, find the **Solution Platforms** drop-down box. (If you don't see it, choose **View** > **Toolbars** > **Standard** to display the toolbar containing **Solution Platforms**.) If you're running 64-bit Windows, choose **x64** in the drop-down box. 64-bit Windows can also run 32-bit applications, so you can choose **x86** if you prefer.

Install the Speech SDK

Finally, install the Speech SDK NuGet package, and reference the Speech SDK in your project:

1. In **Solution Explorer**, right-click your solution, and choose **Manage NuGet Packages for Solution** to go to the **NuGet - Solution** window.

2. Select **Browse**.

3. In **Package source**, choose **nuget.org**.

4. In the **Search** box, enter `Microsoft.CognitiveServices.Speech`, and then choose that package after it appears in the search results.

5. In the package status pane next to the search results, select your **helloworld** project.

6. Select **Install**.

7. In the **Preview Changes** dialog box, select **OK**.

8. In the **License Acceptance** dialog box, view the license, and then select **I Accept**. The package installation begins, and when installation is complete, the **Output** pane displays a message similar to the following text: `Successfully installed 'Microsoft.CognitiveServices.Speech 1.15.0' to helloworld`.

Open your project in Visual Studio Next, open your project in Visual Studio.

Launch Visual Studio 2019. Load your project and open helloworld.cpp. Start with some boilerplate code Let's

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

Open your project in Visual Studio

Next, open your project in Visual Studio.

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

Start with some boilerplate code

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

```
#include <iostream>
#include <speechapi_cxx.h>

using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Intent;

int main()
{
    std::cout << "Hello World!\n";

    auto 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.

```
auto intentRecognizer = IntentRecognizer::FromConfig(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.

```
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:

```
std::cout << "Say something ..." << std::endl;
auto result = intentRecognizer->RecognizeOnceAsync().get();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, let's just print the result.

Insert this code below `auto result = intentRecognizer->RecognizeOnceAsync().get();`:

```
auto entities = result->GetEntities();

switch (result->Reason)
{
case ResultReason::RecognizedSpeech:
case ResultReason::RecognizedIntent:
    std::cout << "RECOGNIZED: Text = " << result->Text.c_str() << std::endl;
    std::cout << "  Intent Id = " << result->IntentId.c_str() << std::endl;
    if (entities.find("floorName") != entities.end())
    {
        std::cout << "  Floor name: = " << entities["floorName"].c_str() << std::endl;
    }

    if (entities.find("action") != entities.end())
    {
        std::cout << "  Action: = " << entities["action"].c_str() << std::endl;
    }

    break;
case ResultReason::NoMatch:
{
    auto noMatch = NoMatchDetails::FromResult(result);
    switch (noMatch->Reason)
    {
    case NoMatchReason::NotRecognized:
        std::cout << "NOMATCH: Speech was detected, but not recognized." << std::endl;
        break;
    case NoMatchReason::InitialSilenceTimeout:
        std::cout << "NOMATCH: The start of the audio stream contains only silence, and the service timed
out waiting for speech." << std::endl;
        break;
    case NoMatchReason::InitialBabbleTimeout:
        std::cout << "NOMATCH: The start of the audio stream contains only noise, and the service timed out
waiting for speech." << std::endl;
        break;
    case NoMatchReason::KeywordNotRecognized:
        std::cout << "NOMATCH: Keyword not recognized" << std::endl;
        break;
    }
    break;
}
case ResultReason::Canceled:
{
    auto cancellation = CancellationDetails::FromResult(result);

    if (!cancellation->ErrorDetails.empty())
    {
        std::cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails.c_str() << std::endl;
        std::cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}
default:
    break;
}
```

Check your code

At this point, your code should look like this:

```
#include <iostream>
#include <speechapi_cxx.h>

using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Intent;
```

```
int main()
{
    auto config = SpeechConfig::FromSubscription("YOUR_SUBSCRIPTION_KEY", "YOUR_SUBSCRIPTION_REGION");
    auto intentRecognizer = IntentRecognizer::FromConfig(config);

    intentRecognizer->AddIntent("Take me to floor {floorName}.", "ChangeFloors");
    intentRecognizer->AddIntent("Go to floor {floorName}.", "ChangeFloors");
    intentRecognizer->AddIntent("{action} the door.", "OpenCloseDoor");

    std::cout << "Say something ..." << std::endl;

    auto result = intentRecognizer->RecognizeOnceAsync().get();
    auto entities = result->GetEntities();

    switch (result->Reason)
    {
    case ResultReason::RecognizedSpeech:
    case ResultReason::RecognizedIntent:
        std::cout << "RECOGNIZED: Text = " << result->Text.c_str() << std::endl;
        std::cout << "  Intent Id = " << result->IntentId.c_str() << std::endl;
        if (entities.find("floorName") != entities.end())
        {
            std::cout << "  Floor name: = " << entities["floorName"].c_str() << std::endl;
        }

        if (entities.find("action") != entities.end())
        {
            std::cout << "  Action: = " << entities["action"].c_str() << std::endl;
        }

        break;
    case ResultReason::NoMatch:
    {
        auto noMatch = NoMatchDetails::FromResult(result);
        switch (noMatch->Reason)
        {
        case NoMatchReason::NotRecognized:
            std::cout << "NOMATCH: Speech was detected, but not recognized." << std::endl;
            break;
        case NoMatchReason::InitialSilenceTimeout:
            std::cout << "NOMATCH: The start of the audio stream contains only silence, and the service
timed out waiting for speech." << std::endl;
            break;
        case NoMatchReason::InitialBabbleTimeout:
            std::cout << "NOMATCH: The start of the audio stream contains only noise, and the service timed
out waiting for speech." << std::endl;
            break;
        case NoMatchReason::KeywordNotRecognized:
            std::cout << "NOMATCH: Keyword not recognized." << std::endl;
            break;
        }
        break;
    }
    case ResultReason::Canceled:
    {
        auto cancellation = CancellationDetails::FromResult(result);

        if (!cancellation->ErrorDetails.empty())
        {
            std::cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails.c_str() << std::endl;
            std::cout << "CANCELED: Did you update the subscription info?" << std::endl;
        }
    }
    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
  Floor name: = seven
```

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

SDK, REST, and CLI developer resources for Language Understanding (LUIS)

11/23/2021 • 3 minutes to read • Edit Online

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/{ve query={query}[&verbose][&log][&show-all-intents]`

PURPOSE	URL
V3 Prediction - slot prediction on runtime endpoint	`https://{your-resource-name}.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/{appId}/slots/{slotN query={query}[&verbose][&log][&show-all-intents]`

The following table explains the parameters, denoted with curly braces `{}`, in the previous table.

PARAMETER	PURPOSE
`your-resource-name`	Azure resource name
`q` or `query`	utterance text sent from client application such as chat bot
`version`	10 character version name
`slot`	`production` or `staging`

REST query string parameters

V3 API query string parameters include:

QUERY PARAMETER	LUIS PORTAL NAME	TYPE	VERSION	DEFAULT	PURPOSE
`log`	Save logs	boolean	V2 & V3	false	Store query in log file. Default value is false.
`query`	-	string	V3 only	No default - it is required in the GET request	**In V2**, the utterance to be predicted is in the `q` parameter. **In V3**, the functionality is passed in the `query` parameter.
`show-all-intents`	Include scores for all intents	boolean	V3 only	false	Return all intents with the corresponding score in the **prediction.intents** object. Intents are returned as objects in a parent `intents` object. This allows programmatic access without needing to find the intent in an array: `prediction.intents.give`. In V2, these were returned in an array.

QUERY PARAMETER	LUIS PORTAL NAME	TYPE	VERSION	DEFAULT	PURPOSE
`verbose`	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 `show-all-intents`. **In V3**, this parameter only provides entity metadata details of entity prediction.
`timezoneOffset`	-	string	V2	-	Timezone applied to datetimeV2 entities.
`datetimeReference`	-	string	V3	-	Timezone applied to datetimeV2 entities. Replaces `timezoneOffset` 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

11/23/2021 • 4 minutes to read • Edit Online

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.

Multi-language LUIS apps

If you need a multi-language 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.

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`	✔	✔	✔	✔
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

LANGUAGE	LOCALE	PREBUILT DOMAIN	PREBUILT ENTITY	PHRASE LIST RECOMMENDATIONS	**SENTIMENT ANALYSIS AND KEY PHRASE EXTRACTION
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)	✔		
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 `Ich fahre einen krankenwagen` as an utterance, it is turned to `Ich fahre einen kranken wagen`. Allowing the marking of `kranken` and `wagen` independently as different entities.
German `de-de`	1.0.2	Tokenizes words by splitting them on spaces. If a user enters `Ich fahre einen krankenwagen` as an utterance, it remains a single token. Thus `krankenwagen` 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 `Ik ga naar de kleuterschool` as an utterance, it is turned to `Ik ga naar de kleuter school`. Allowing the marking of `kleuter` and `school` independently as different entities.
Dutch `nl-nl`	1.0.1	Tokenizes words by splitting them on spaces. If a user enters `Ik ga naar de kleuterschool` as an utterance, it remains a single token. Thus `kleuterschool` 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.

Speech service supported regions

11/23/2021 • 5 minutes to read • Edit Online

The Speech service allows your application to convert audio to text, perform speech translation, and covert text to speech. The service is available in multiple regions with unique endpoints for the Speech SDK and REST APIs.

The Speech portal, where you can perform custom configurations to your speech experience for all regions, is available at speech.microsoft.com.

Keep in mind the following points when considering regions:

- If your application uses a Speech SDK, you provide the region identifier, such as `westus`, when creating a speech configuration. Make sure the region matches the region of your subscription.
- If your application uses one of the Speech service's REST APIs, the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

> **NOTE**
>
> Speech Services doesn't store/process customer data outside the region the customer deploys the service instance in.

Speech SDK

In the Speech SDK, the region is specified as a parameter (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

Speech-to-text, text-to-speech, and translation

The Speech service is available in these regions for **speech recognition**, **text-to-speech**, and **translation**:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`

GEOGRAPHY	REGION	REGION IDENTIFIER
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

If you plan to train a custom model with audio data, use one of the regions with dedicated hardware for faster training. You can use the REST API to copy the fully trained model to another region later.

Intent recognition

Available regions for **intent recognition** via the Speech SDK are the following:

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 in these regions:

GLOBAL REGION	REGION	REGION IDENTIFIER
North America	West US	`westus`
North America	West US 2	`westus2`
North America	East US	`eastus`
North America	East US 2	`eastus2`
North America	West Central US	`westcentralus`
North America	South Central US	`southcentralus`
Europe	West Europe	`westeurope`
Europe	North Europe	`northeurope`
Asia	East Asia	`eastasia`
Asia	Southeast Asia	`southeastasia`
India	Central India	`centralindia`

Speaker Recognition

Available regions for **Speaker Recognition** are the following:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Central India	`centralindia`
Australia	Australia East	`australiaeast`
Europe	North Europe	`northeurope`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	West Europe	`westeurope`
Europe	UK South	`uksouth`

REST APIs

The Speech service also exposes REST endpoints for speech-to-text, text-to-speech and speaker recognition requests.

Speech-to-text

For speech-to-text reference documentation, see Speech-to-text REST API.

The endpoint for the REST API has this format:

```
https://<REGION_IDENTIFIER>.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
```

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

GEOGRAPHY	REGION	REGION IDENTIFIER
Americas	Central US	`centralus`
Americas	East US	`eastus`
Americas	East US 2	`eastus2`
Americas	North Central US	`northcentralus`
Americas	South Central US	`southcentralus`
Americas	West Central US	`westcentralus`
Americas	West US	`westus`
Americas	West US 2	`westus2`
Americas	Canada Central	`canadacentral`
Americas	Brazil South	`brazilsouth`
Asia Pacific	East Asia	`eastasia`
Asia Pacific	Southeast Asia	`southeastasia`
Asia Pacific	Australia East	`australiaeast`
Asia Pacific	Central India	`centralindia`
Asia Pacific	Japan East	`japaneast`
Asia Pacific	Japan West	`japanwest`
Asia Pacific	Korea Central	`koreacentral`

GEOGRAPHY	REGION	REGION IDENTIFIER
Europe	North Europe	`northeurope`
Europe	West Europe	`westeurope`
Europe	France Central	`francecentral`
Europe	Switzerland North	`switzerlandnorth`
Europe	UK South	`uksouth`

> **NOTE**
>
> The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is:
>
> ```
> https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
> ```
>
> .

Text-to-speech

For text-to-speech reference documentation, see Text-to-speech REST API.

Neural and standard voices

Use this table to determine **availability of neural and standard voices** by region/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`
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/v`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`

REGION	ENDPOINT
South Central US	`https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v`
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`

> **TIP**
>
> Voices in preview are only available in these 3 regions: East US, West Europe and Southeast Asia.

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	`https://australiaeast.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Brazil South	`https://brazilsouth.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
Canada Central	`https://canadacentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
Central US	`https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East Asia	`https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US	`https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
East US 2	`https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
France Central	`https://francecentral.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
India Central	`https://centralindia.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
Japan East	`https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
Japan West	`https://japanwest.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

REGION	ENDPOINT
Korea Central	`https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v` `deploymentId={deploymentId}`
North Central US	`https://northcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
North Europe	`https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1` `deploymentId={deploymentId}`
South Central US	`https://southcentralus.voice.speech.microsoft.com/cognitiveservices` `deploymentId={deploymentId}`
Southeast Asia	`https://southeastasia.voice.speech.microsoft.com/cognitiveservices/` `deploymentId={deploymentId}`
UK South	`https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West Europe	`https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US	`https://westus.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`
West US 2	`https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?` `deploymentId={deploymentId}`

Custom Neural Voice

The following table details regional support for Custom Neural Voice features.

FEATURE	SUPPORTED REGIONS
Voice model hosting	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Realtime Characters	East US, West US 2, South Central US, Southeast Asia, UK South, West Europe, Australia East
Long Audio Characters	East US, West Europe, Southeast Asia, UK South, India Central
Custom Neural Training	East US, Southeast Asia, UK South

Long audio API

The Long audio API is available in multiple regions with unique endpoints.

REGION	ENDPOINT
East US	`https://eastus.customvoice.api.speech.microsoft.com`
India Central	`https://centralindia.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`

Speaker Recognition

For speaker recognition reference documentation, see Speaker Recognition REST API. Available regions are the same as Speaker Recognition SDK.

What is Speaker Recognition?

11/23/2021 • 4 minutes to read • Edit Online

Speaker Recognition is used to determine who is speaking in an audio clip. The service can verify and identify speakers by their unique voice characteristics 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), or 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. Apply to use it through the Azure Cognitive Services Speaker Recognition Limited Access Review. After approval, you can access the Speaker Recognition APIs. For more information, please visit 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. It can be used to verify individuals for secure, frictionless customer engagements in a wide range of solutions, from customer identity verification in call centers to contact-less facility access.

How does Speaker Verification work?

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. **Text-independent** verification means 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, while 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 to activate the enrollment. It doesn't have any restrictions on the audio sample to be verified, as it only extracts voice features to score similarity.

The APIs are not intended to determine whether the audio is from a live person or an imitation/recording of an enrolled speaker.

Speaker Identification

Speaker Identification is used to 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:

- Support solutions for remote meeting productivity
- Build multi-user device personalization

How does Speaker Identification work?

Enrollment for speaker identification is **text-independent**, which means that there are no restrictions on what the speaker says in the audio, besides the initial activation phrase to activate the enrollment. 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 does not 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 service must be aware of Microsoft's policies on customer data. You should ensure that you have received the appropriate permissions from the users for Speaker Recognition. You can find more details in Data and privacy for Speaker Recognition. For more information, see theCognitive Services pageon the Microsoft Trust Center.

Common questions and solutions

QUESTION	SOLUTION
What scenarios can Speaker Recognition be used for?	Call center customer verification, voice-based patient check-in, meeting transcription, multi-user device personalization
What is 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, or **enrolled** voice.
What's the difference between text-dependent and text-independent verification?	Text-dependent verification requires a specific pass-phrase for both enrollment and recognition. Text-independent verification requires a longer voice sample that must start with a particular activation phrase for enrollment, but anything can be spoken, including during recognition.
What languages are supported?	English, French, Spanish, Chinese, German, Italian, Japanese and Portuguese
What Azure regions are supported?	Speaker Recognition is a preview service, and currently only available in the West US region.
What audio formats are supported?	Mono 16 bit, 16kHz PCM-encoded WAV
Accept and **Reject** responses aren't accurate, how do you tune the threshold?	Since the optimal threshold varies highly with scenarios, the API decides whether to “Accept” or “Reject” simply based on a default threshold of 0.5. Advanced users are advised to override the default decision and fine tune the result based on your own scenario.

QUESTION	SOLUTION
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 are not retained or stored.

Next steps

- Complete the Speaker Recognition basics article for a run-through of common design patterns you can use in your applications.
- See the video tutorial for text-independent speaker verification.

Responsible use of AI

- Transparency Note for Speaker Recognition
- Characteristics and limitations for Speaker Recognition
- Limited access for Speaker Recognition
- Guidance for integration and responsible use with Speaker Recognition
- Data and privacy for Speaker Recognition

Get started with Speaker Recognition

11/23/2021 • 30 minutes to read • Edit Online

In this quickstart, you learn basic design patterns for Speaker Recognition 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 on left nav for a list of the supported platforms.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

> **IMPORTANT**
> Microsoft limits access to Speaker Recognition. Apply to use it through the Azure Cognitive Services Speaker Recognition Limited Access Review. After approval, you can access the Speaker Recognition APIs.

Install the Speech SDK

Before you can do anything, you'll need to 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.

```
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 using the Speech SDK, you need to create a `SpeechConfig`. In this example, you create a `SpeechConfig` 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.

```
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 using a **text-dependent** strategy, which requires a specific pass-phrase to use for both enrollment and verification. See the reference docs for a list of supported pass-phrases.

Start by creating the following function in your `Program` class to enroll a voice profile.

```
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"))
    using (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 is created, you specify how you will input audio samples, 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 required, for enrollment. In each iteration, `client.EnrollProfileAsync(profile, audioInput)` will prompt you to speak the pass-phrase into your microphone, and add the sample to the voice profile.

After enrollment is completed, you call `await SpeakerVerify(config, profile, profileMapping)` to verify against the profile you just created. Add another function to define `SpeakerVerify`.

```
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 pass-phrase again, but this time it will validate it against your voice profile and return a similarity score ranging from 0.0-1.0. The `result` object also returns `Accept` or `Reject`, based on whether or not the pass-phrase matches.

Next, modify your `Main()` function to call the new functions you created. Additionally, note that you create a `Dictionary<string, string>` to pass by reference through your function calls. The reason for this is that the service does not allow storing a human-readable name with a created `VoiceProfile`, and 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, the service cannot store it**.

```
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, and you are prompted to speak the phrase *My voice is my passport, verify me* three times for enrollment, and one additional time for verification. The result returned is the similarity score, which you can use to create your own custom thresholds for verification.

```
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:

- Does not 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.

```
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"))
    using (var phraseResult = await
client.GetActivationPhrasesAsync(VoiceProfileType.TextIndependentVerification, "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. Again, the similarity score is returned.

```
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 very similar to **text-independent verification**, with the main difference being able 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**, and 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.

```
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))
    using (var phraseResult = await
client.GetActivationPhrasesAsync(VoiceProfileType.TextIndependentVerification, "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.

```
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. This will prompt you to create a new voice profile for each name in this list, so you can add more names to create additional profiles for friends or colleagues.

```
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, and you are prompted to speak to enroll voice samples for the first profile. After the enrollment is completed, you are prompted to repeat this process for each name in the list `profileNames`. After each enrollment is finished, you are prompted to have **anyone** speak, and the service will attempt to identify this person from among your enrolled voice profiles.

This example returns only the closest match and it's similarity score, but you can get the full response that includes the top five similarity scores by adding `string json = result.Properties.GetProperty(PropertyId.SpeechServiceResponse_JsonResult)` to your `SpeakerIdentification` function.

Changing audio input type

The examples in this article use the default device microphone as input for audio samples. However, in scenarios where you need to use audio files instead of microphone input, simply 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, using a microphone for enrollment and files for verification, for example.

Deleting 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.

```
public static async Task DeleteProfile(SpeechConfig config, string profileId)
{
    using (var client = new VoiceProfileClient(config))
    {
        var profile = new VoiceProfile(profileId);
        await client.DeleteProfileAsync(profile);
    }
}
```

In this quickstart, you learn basic design patterns for Speaker Recognition 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 on left nav for a list of the supported platforms.

Skip to samples on GitHub

If you want to skip straight to sample code, see the C++ quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

> **IMPORTANT**
> Microsoft limits access to Speaker Recognition. Apply to use it through the Azure Cognitive Services Speaker Recognition Limited Access Review. After approval, you can access the Speaker Recognition APIs.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, use the following instructions:

- Linux
- macOS
- Windows

Import dependencies

To run the examples in this article, add the following statements at the top of your .cpp file.

```
#include <iostream>
#include <stdexcept>
// Note: Install the NuGet package Microsoft.CognitiveServices.Speech.
#include <speechapi_cxx.h>

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

// Note: Change the locale if desired.
auto profile_locale = "en-us";
auto audio_config = Audio::AudioConfig::FromDefaultMicrophoneInput();
auto ticks_per_second = 10000000;
```

Create a speech configuration

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

```
shared_ptr<SpeechConfig> GetSpeechConfig()
{
    auto subscription_key = 'PASTE_YOUR_SPEECH_SUBSCRIPTION_KEY_HERE';
    auto region = 'PASTE_YOUR_SPEECH_ENDPOINT_REGION_HERE';
    auto config = SpeechConfig::FromSubscription(subscription_key, region);
    return config;
}
```

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 using a **text-dependent** strategy, which requires a specific passphrase to use for both enrollment and verification. See the reference docs for a list of supported passphrases.

TextDependentVerification function

Start by creating the `TextDependentVerification` function.

```
void TextDependentVerification(shared_ptr<VoiceProfileClient> client, shared_ptr<SpeakerRecognizer>
recognizer)
{
    std::cout << "Text Dependent Verification:\n\n";
    // Create the profile.
    auto profile = client->CreateProfileAsync(VoiceProfileType::TextDependentVerification,
profile_locale).get();
    std::cout << "Created profile ID: " << profile->GetId() << "\n";
    AddEnrollmentsToTextDependentProfile(client, profile);
    SpeakerVerify(profile, recognizer);
    // Delete the profile.
    client->DeleteProfileAsync(profile);
}
```

This function creates a VoiceProfile object with the CreateProfileAsync method. Note there are three types of `VoiceProfile`:

- TextIndependentIdentification
- TextDependentVerification
- TextIndependentVerification

In this case you pass `VoiceProfileType::TextDependentVerification` to `CreateProfileAsync`.

You then call two helper functions that you'll define next, `AddEnrollmentsToTextDependentProfile` and `SpeakerVerify`. Finally, call DeleteProfileAsync to clean up the profile.

AddEnrollmentsToTextDependentProfile function

Define the following function to enroll a voice profile.

```
void AddEnrollmentsToTextDependentProfile(shared_ptr<VoiceProfileClient> client, shared_ptr<VoiceProfile>
profile)
{
    shared_ptr<VoiceProfileEnrollmentResult> enroll_result = nullptr;
    while (enroll_result == nullptr || enroll_result-
>GetEnrollmentInfo(EnrollmentInfoType::RemainingEnrollmentsCount) > 0)
    {
        std::cout << "Please say the passphrase, \"My voice is my passport, verify me.\"\n";
        enroll_result = client->EnrollProfileAsync(profile, audio_config).get();
        std::cout << "Remaining enrollments needed: " << enroll_result-
>GetEnrollmentInfo(EnrollmentInfoType::RemainingEnrollmentsCount) << ".\n";
    }
    std::cout << "Enrollment completed.\n\n";
}
```

In this function, you enroll audio samples in a `while` loop that tracks the number of samples remaining, and required, for enrollment. In each iteration, EnrollProfileAsync prompts you to speak the passphrase into your microphone, and adds the sample to the voice profile.

SpeakerVerify function

Define `SpeakerVerify` as follows.

```
void SpeakerVerify(shared_ptr<VoiceProfile> profile, shared_ptr<SpeakerRecognizer> recognizer)
{
    shared_ptr<SpeakerVerificationModel> model = SpeakerVerificationModel::FromProfile(profile);
    std::cout << "Speak the passphrase to verify: \"My voice is my passport, verify me.\"\n";
    shared_ptr<SpeakerRecognitionResult> result = recognizer->RecognizeOnceAsync(model).get();
    std::cout << "Verified voice profile for speaker: " << result->ProfileId << ". Score is: " << result-
>GetScore() << ".\n\n";
}
```

In this function, you create a SpeakerVerificationModel object with the SpeakerVerificationModel::FromProfile method, passing in the VoiceProfile object you created earlier.

Next, SpeechRecognizer::RecognizeOnceAsync prompts you to speak the passphrase again, but this time it will validate it against your voice profile and return a similarity score ranging from 0.0-1.0. The SpeakerRecognitionResult object also returns `Accept` or `Reject`, based on whether or not the passphrase matches.

Text-independent verification

In contrast to **text-dependent** verification, **text-independent** verification does not require three audio samples, but *does* require 20 seconds of total audio.

TextIndependentVerification function

Start by creating the `TextIndependentVerification` function.

```
void TextIndependentVerification(shared_ptr<VoiceProfileClient> client, shared_ptr<SpeakerRecognizer>
recognizer)
{
    std::cout << "Text Independent Verification:\n\n";
    // Create the profile.
    auto profile = client->CreateProfileAsync(VoiceProfileType::TextIndependentVerification,
profile_locale).get();
    std::cout << "Created profile ID: " << profile->GetId() << "\n";
    AddEnrollmentsToTextIndependentProfile(client, profile);
    SpeakerVerify(profile, recognizer);
    // Delete the profile.
    client->DeleteProfileAsync(profile);
}
```

Like the `TextDependentVerification` function, this function creates a VoiceProfile object with the CreateProfileAsync method.

In this case you pass `VoiceProfileType::TextIndependentVerification` to `CreateProfileAsync`.

You then call two helper functions: `AddEnrollmentsToTextIndependentProfile`, which you'll define next, and `SpeakerVerify`, which you defined already. Finally, call DeleteProfileAsync to clean up the profile.

AddEnrollmentsToTextIndependentProfile

Define the following function to enroll a voice profile.

```
void AddEnrollmentsToTextIndependentProfile(shared_ptr<VoiceProfileClient> client, shared_ptr<VoiceProfile>
profile)
{
    shared_ptr<VoiceProfileEnrollmentResult> enroll_result = nullptr;
    while (enroll_result == nullptr || enroll_result-
>GetEnrollmentInfo(EnrollmentInfoType::RemainingEnrollmentsSpeechLength) > 0)
    {
        std::cout << "Continue speaking to add to the profile enrollment sample.\n";
        enroll_result = client->EnrollProfileAsync(profile, audio_config).get();
        std::cout << "Remaining audio time needed: " << enroll_result-
>GetEnrollmentInfo(EnrollmentInfoType::RemainingEnrollmentsSpeechLength) / ticks_per_second << "
seconds.\n";
    }
    std::cout << "Enrollment completed.\n\n";
}
```

In this function, you enroll audio samples in a `while` loop that tracks the number of seconds of audio remaining, and required, for enrollment. In each iteration, EnrollProfileAsync prompts you to speak into your microphone, and adds the sample to the voice profile.

Speaker identification

Speaker Identification is used to determine **who** is speaking from a given group of enrolled voices. The process is very similar to **text-independent verification**, with the main difference being able to verify against multiple voice profiles at once, rather than verifying against a single profile.

TextIndependentIdentification function

Start by creating the `TextIndependentIdentification` function.

```
void TextIndependentIdentification(shared_ptr<VoiceProfileClient> client, shared_ptr<SpeakerRecognizer>
recognizer)
{
    std::cout << "Speaker Identification:\n\n";
    // Create the profile.
    auto profile = client->CreateProfileAsync(VoiceProfileType::TextIndependentIdentification,
profile_locale).get();
    std::cout << "Created profile ID: " << profile->GetId() << "\n";
    AddEnrollmentsToTextIndependentProfile(client, profile);
    SpeakerIdentify(profile, recognizer);
    // Delete the profile.
    client->DeleteProfileAsync(profile);
}
```

Like the `TextDependentVerification` and `TextIndependentVerification` functions, this function creates a VoiceProfile object with the CreateProfileAsync method.

In this case you pass `VoiceProfileType::TextIndependentIdentification` to `CreateProfileAsync`.

You then call two helper functions: `AddEnrollmentsToTextIndependentProfile`, which you defined already, and `SpeakerIdentify`, which you'll define next. Finally, call DeleteProfileAsync to clean up the profile.

SpeakerIdentify function

Define the `SpeakerIdentify` function as follows.

```
void SpeakerIdentify(shared_ptr<VoiceProfile> profile, shared_ptr<SpeakerRecognizer> recognizer)
{
    shared_ptr<SpeakerIdentificationModel> model = SpeakerIdentificationModel::FromProfiles({ profile });
    // Note: We need at least four seconds of audio after pauses are subtracted.
    std::cout << "Please speak for at least ten seconds to identify who it is from your list of enrolled
speakers.\n";
    shared_ptr<SpeakerRecognitionResult> result = recognizer->RecognizeOnceAsync(model).get();
    std::cout << "The most similar voice profile is: " << result->ProfileId << " with similarity score: " <<
result->GetScore() << ".\n\n";
}
```

In this function, you create a SpeakerIdentificationModel object with the SpeakerIdentificationModel::FromProfiles method. `SpeakerIdentificationModel::FromProfiles` accepts a list of VoiceProfile objects. In this case, you'll just pass in the `VoiceProfile` object you created earlier. However, if you want, you can pass in multiple `VoiceProfile` objects, each enrolled with audio samples from a different voice.

Next, SpeechRecognizer::RecognizeOnceAsync prompts you to speak again. This time it compares your voice to the enrolled voice profiles and returns the most similar voice profile.

Main function

Finally, define the `main` function as follows.

```
int main()
{
    auto speech_config = GetSpeechConfig();
    auto client = VoiceProfileClient::FromConfig(speech_config);
    auto recognizer = SpeakerRecognizer::FromConfig(speech_config, audio_config);
    TextDependentVerification(client, recognizer);
    TextIndependentVerification(client, recognizer);
    TextIndependentIdentification(client, recognizer);
    std::cout << "End of quickstart.\n";
}
```

This function simply calls the functions you defined previously. First, though, it creates a VoiceProfileClient object and a SpeakerRecognizer object.

```
auto speech_config = GetSpeechConfig();
auto client = VoiceProfileClient::FromConfig(speech_config);
auto recognizer = SpeakerRecognizer::FromConfig(speech_config, audio_config);
```

The `VoiceProfileClient` is used to create, enroll and delete voice profiles. The `SpeakerRecognizer` is used to validate speech samples against one or more enrolled voice profiles.

Changing audio input type

The examples in this article use the default device microphone as input for audio samples. However, in scenarios where you need to use audio files instead of microphone input, simply change the following line:

```
auto audio_config = Audio::AudioConfig::FromDefaultMicrophoneInput();
```

to:

```
auto audio_config = Audio::AudioConfig::FromWavFileInput(path/to/your/file.wav);
```

Or replace any use of `audio_config` with Audio::AudioConfig::FromWavFileInput. You can also have mixed inputs, using a microphone for enrollment and files for verification, for example.

In this quickstart, you learn basic design patterns for Speaker Recognition 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 on left nav for a list of the supported platforms.

Skip to samples on GitHub

If you want to skip straight to sample code, see the JavaScript quickstart samples on GitHub.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

> **IMPORTANT**
> Speaker Recognition is currently *only* supported in Azure Speech resources created in the `westus` region.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK for JavaScript . Depending on your platform, use the following instructions:

- Node.js
- Web browser

Additionally, depending on the target environment use one of the following:

- script
- import
- require

Download and extract the Speech SDK for JavaScript *microsoft.cognitiveservices.speech.sdk.bundle.js* file, and place it in a folder accessible to your HTML file.

```
<script src="microsoft.cognitiveservices.speech.sdk.bundle.js"></script>;
```

> **TIP**
> If you're targeting a web browser, and using the `<script>` tag; the `sdk` prefix is not needed. The `sdk` prefix is an alias used to name the `require` module.

Import dependencies

To run the examples in this article, add the following statements at the top of your .js file.

```
"use strict";

/* To run this sample, install:
npm install microsoft-cognitiveservices-speech-sdk
*/
var sdk = require("microsoft-cognitiveservices-speech-sdk");
var fs = require("fs");

// Note: Change the locale if desired.
const profile_locale = "en-us";

/* Note: passphrase_files and verify_file should contain paths to audio files that contain \"My voice is my
passport, verify me.\"
You can obtain these files from:
https://github.com/Azure-Samples/cognitive-services-speech-
sdk/tree/fa6428a0837779cbeae172688e0286625e340942/quickstart/javascript/node/speaker-
recognition/verification
*/
const passphrase_files = ["myVoiceIsMyPassportVerifyMe01.wav", "myVoiceIsMyPassportVerifyMe02.wav",
"myVoiceIsMyPassportVerifyMe03.wav"];
const verify_file = "myVoiceIsMyPassportVerifyMe04.wav";
/* Note: identify_file should contain a path to an audio file that uses the same voice as the other files,
but contains different speech. You can obtain this file from:
https://github.com/Azure-Samples/cognitive-services-speech-
sdk/tree/fa6428a0837779cbeae172688e0286625e340942/quickstart/javascript/node/speaker-
recognition/identification
*/
const identify_file = "aboutSpeechSdk.wav";

var subscription_key = 'PASTE_YOUR_SPEECH_SUBSCRIPTION_KEY_HERE';
var region = 'PASTE_YOUR_SPEECH_ENDPOINT_REGION_HERE';

const ticks_per_second = 10000000;
```

These statements import the required libraries and get your Speech service subscription key and region from your environment variables. They also specify paths to audio files that you will use in the following tasks.

Create helper function

Add the following helper function to read audio files into streams for use by the Speech service.

```
/* From: https://github.com/Azure-Samples/cognitive-services-speech-
sdk/blob/fa6428a0837779cbeae172688e0286625e340942/quickstart/javascript/node/speaker-
recognition/verification/dependent-verification.js#L8
*/
function GetAudioConfigFromFile (file)
{
    let pushStream = sdk.AudioInputStream.createPushStream();
    fs.createReadStream(file).on("data", function(arrayBuffer) {
        pushStream.write(arrayBuffer.buffer);
    }).on("end", function() {
        pushStream.close();
    });
    return sdk.AudioConfig.fromStreamInput(pushStream);
}
```

In this function, you use the AudioInputStream.createPushStream and AudioConfig.fromStreamInput methods to create an AudioConfig object. This `AudioConfig` object represents an audio stream. You will use several of these `AudioConfig` objects during the following tasks.

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 using a **text-dependent** strategy, which requires a specific passphrase to use for both enrollment and verification. See the reference docs for a list of supported passphrases.

TextDependentVerification function

Start by creating the `TextDependentVerification` function.

```
async function TextDependentVerification(client, speech_config)
{
    console.log ("Text Dependent Verification:\n");
    var profile = null;
    try {
        // Create the profile.
        profile = await new Promise ((resolve, reject) => {
            client.createProfileAsync (sdk.VoiceProfileType.TextDependentVerification, profile_locale,
result => { resolve(result); }, error => { reject(error); });
        });
        console.log ("Created profile ID: " + profile.profileId);
        await AddEnrollmentsToTextDependentProfile(client, profile, passphrase_files);
        const audio_config = GetAudioConfigFromFile(verify_file);
        const recognizer = new sdk.SpeakerRecognizer(speech_config, audio_config);
        await SpeakerVerify(profile, recognizer);
    }
    catch (error) {
        console.log ("Error:\n" + error);
    }
    finally {
        if (profile !== null) {
            console.log ("Deleting profile ID: " + profile.profileId);
            await new Promise ((resolve, reject) => {
                client.deleteProfileAsync (profile, result => { resolve(result); }, error => {
reject(error); });
            });
        }
    }
}
```

This function creates a VoiceProfile object with the VoiceProfileClient.createProfileAsync method. Note there are three types of `VoiceProfile`:

- TextIndependentIdentification
- TextDependentVerification
- TextIndependentVerification

In this case, you pass `VoiceProfileType.TextDependentVerification` to `VoiceProfileClient.createProfileAsync`.

You then call two helper functions that you'll define next, `AddEnrollmentsToTextDependentProfile` and `SpeakerVerify`. Finally, call VoiceProfileClient.deleteProfileAsync to remove the profile.

AddEnrollmentsToTextDependentProfile function

Define the following function to enroll a voice profile.

```
async function AddEnrollmentsToTextDependentProfile(client, profile, audio_files)
{
    for (var i = 0; i < audio_files.length; i++) {
        console.log ("Adding enrollment to text dependent profile...");
        const audio_config = GetAudioConfigFromFile (audio_files[i]);
        const result = await new Promise ((resolve, reject) => {
            client.enrollProfileAsync (profile, audio_config, result => { resolve(result); }, error => {
reject(error); });
        });
        if (result.reason === sdk.ResultReason.Canceled) {
            throw(JSON.stringify(sdk.VoiceProfileEnrollmentCancellationDetails.fromResult(result)));
        }
        else {
            console.log ("Remaining enrollments needed: " + result.privDetails["remainingEnrollmentsCount"]
+ ".");
        }
    };
    console.log ("Enrollment completed.\n");
}
```

In this function, you call the `GetAudioConfigFromFile` function you defined earlier to create `AudioConfig` objects from audio samples. These audio samples contain a passphrase such as "My voice is my passport, verify me." You then enroll these audio samples using the VoiceProfileClient.enrollProfileAsync method.

SpeakerVerify function

Define `SpeakerVerify` as follows.

```
async function SpeakerVerify(profile, recognizer)
{
    const model = sdk.SpeakerVerificationModel.fromProfile(profile);
    const result = await new Promise ((resolve, reject) => {
        recognizer.recognizeOnceAsync (model, result => { resolve(result); }, error => { reject(error); });
    });
    console.log ("Verified voice profile for speaker: " + result.profileId + ". Score is: " + result.score +
".\n");
}
```

In this function, you create a SpeakerVerificationModel object with the SpeakerVerificationModel.FromProfile method, passing in the VoiceProfile object you created earlier.

Next, you call the SpeechRecognizer.recognizeOnceAsync method to validate an audio sample that contains the same passphrase as the audio samples you enrolled previously. `SpeechRecognizer.recognizeOnceAsync` returns a SpeakerRecognitionResult object, whose `score` property contains a similarity score ranging from 0.0-1.0. The `SpeakerRecognitionResult` object also contains a `reason` property of type ResultReason. If the verification was

successful, the `reason` property should have value `RecognizedSpeaker`.

Text-independent verification

In contrast to **text-dependent** verification, **text-independent** verification:

- Does not require a certain passphrase to be spoken, anything can be spoken
- Does not require three audio samples, but *does* require 20 seconds of total audio

TextIndependentVerification function

Start by creating the `TextIndependentVerification` function.

```
async function TextIndependentVerification(client, speech_config)
{
    console.log ("Text Independent Verification:\n");
    var profile = null;
    try {
        // Create the profile.
        profile = await new Promise ((resolve, reject) => {
            client.createProfileAsync (sdk.VoiceProfileType.TextIndependentVerification, profile_locale,
result => { resolve(result); }, error => { reject(error); });
        });
        console.log ("Created profile ID: " + profile.profileId);
        await AddEnrollmentsToTextIndependentProfile(client, profile, [identify_file]);
        const audio_config = GetAudioConfigFromFile(passphrase_files[0]);
        const recognizer = new sdk.SpeakerRecognizer(speech_config, audio_config);
        await SpeakerVerify(profile, recognizer);
    }
    catch (error) {
        console.log ("Error:\n" + error);
    }
    finally {
        if (profile !== null) {
            console.log ("Deleting profile ID: " + profile.profileId);
            await new Promise ((resolve, reject) => {
                client.deleteProfileAsync (profile, result => { resolve(result); }, error => {
reject(error); });
            });
        }
    }
}
```

Like the `TextDependentVerification` function, this function creates a VoiceProfile object with the VoiceProfileClient.createProfileAsync method.

In this case, you pass `VoiceProfileType.TextIndependentVerification` to `createProfileAsync`.

You then call two helper functions: `AddEnrollmentsToTextIndependentProfile`, which you'll define next, and `SpeakerVerify`, which you defined already. Finally, call VoiceProfileClient.deleteProfileAsync to remove the profile.

AddEnrollmentsToTextIndependentProfile

Define the following function to enroll a voice profile.

```
async function AddEnrollmentsToTextIndependentProfile(client, profile, audio_files)
{
    for (var i = 0; i < audio_files.length; i++) {
        console.log ("Adding enrollment to text independent profile...");
        const audio_config = GetAudioConfigFromFile (audio_files[i]);
        const result = await new Promise ((resolve, reject) => {
            client.enrollProfileAsync (profile, audio_config, result => { resolve(result); }, error => {
reject(error); });
        });
        if (result.reason === sdk.ResultReason.Canceled) {
            throw(JSON.stringify(sdk.VoiceProfileEnrollmentCancellationDetails.fromResult(result)));
        }
        else {
            console.log ("Remaining audio time needed: " +
(result.privDetails["remainingEnrollmentsSpeechLength"] / ticks_per_second) + " seconds.");
        }
    }
    console.log ("Enrollment completed.\n");
}
```

In this function, you call the `GetAudioConfigFromFile` function you defined earlier to create `AudioConfig` objects from audio samples. You then enroll these audio samples using the VoiceProfileClient.enrollProfileAsync method.

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**, with the main difference being able to verify against multiple voice profiles at once, rather than verifying against a single profile.

TextIndependentIdentification function

Start by creating the `TextIndependentIdentification` function.

```
async function TextIndependentIdentification(client, speech_config)
{
    console.log ("Text Independent Identification:\n");
    var profile = null;
    try {
        // Create the profile.
        profile = await new Promise ((resolve, reject) => {
            client.createProfileAsync (sdk.VoiceProfileType.TextIndependentIdentification, profile_locale,
result => { resolve(result); }, error => { reject(error); });
        });
        console.log ("Created profile ID: " + profile.profileId);
        await AddEnrollmentsToTextIndependentProfile(client, profile, [identify_file]);
        const audio_config = GetAudioConfigFromFile(passphrase_files[0]);
        const recognizer = new sdk.SpeakerRecognizer(speech_config, audio_config);
        await SpeakerIdentify(profile, recognizer);
    }
    catch (error) {
        console.log ("Error:\n" + error);
    }
    finally {
        if (profile !== null) {
            console.log ("Deleting profile ID: " + profile.profileId);
            await new Promise ((resolve, reject) => {
                client.deleteProfileAsync (profile, result => { resolve(result); }, error => {
reject(error); });
            });
        }
    }
}
```

Like the `TextDependentVerification` and `TextIndependentVerification` functions, this function creates a VoiceProfile object with the VoiceProfileClient.createProfileAsync method.

In this case, you pass `VoiceProfileType.TextIndependentIdentification` to `VoiceProfileClient.createProfileAsync` .

You then call two helper functions: `AddEnrollmentsToTextIndependentProfile`, which you defined already, and `SpeakerIdentify`, which you'll define next. Finally, call VoiceProfileClient.deleteProfileAsync to remove the profile.

SpeakerIdentify function

Define the `SpeakerIdentify` function as follows.

```
async function SpeakerIdentify(profile, recognizer)
{
    const model = sdk.SpeakerIdentificationModel.fromProfiles([profile]);
    const result = await new Promise ((resolve, reject) => {
        recognizer.recognizeOnceAsync (model, result => { resolve(result); }, error => { reject(error); });
    });
    console.log ("The most similar voice profile is: " + result.profileId + " with similarity score: " +
result.score + ".\n");
}
```

In this function, you create a SpeakerIdentificationModel object with the SpeakerIdentificationModel.fromProfiles method, passing in the VoiceProfile object you created earlier.

Next, you call the SpeechRecognizer.recognizeOnceAsync method and pass in an audio sample. `SpeechRecognizer.recognizeOnceAsync` tries to identify the voice for this audio sample based on the `VoiceProfile` objects you used to create the `SpeakerIdentificationModel`. It returns a SpeakerRecognitionResult object, whose `profileId` property identifies the matching `VoiceProfile`, if any, while the `score` property contains a similarity score ranging from 0.0-1.0.

Main function

Finally, define the `main` function as follows.

```
async function main() {
    const speech_config = sdk.SpeechConfig.fromSubscription(subscription_key, region);
    const client = new sdk.VoiceProfileClient(speech_config);

    await TextDependentVerification(client, speech_config);
    await TextIndependentVerification(client, speech_config);
    await TextIndependentIdentification(client, speech_config);
    console.log ("End of quickstart.");
}
main();
```

This function creates a VoiceProfileClient object, which is used to create, enroll, and delete voice profiles. Then it calls the functions you defined previously.

In this quickstart, you learn basic design patterns for Speaker Recognition 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 on left nav for a list of the supported platforms.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

> **IMPORTANT**
>
> Speaker Recognition is currently *only* supported in Azure Speech resources created in the `westus` 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 using a **text-dependent** strategy, which requires a specific passphrase to use for both enrollment and verification. See the reference docs for a list of supported passphrases.

Start by creating a voice profile. You will need to insert your Speech service subscription key and region into each of the curl commands in this article.

```
# Note Change locale if needed.
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-dependent/profiles' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: application/json' \
--data-raw '{
    '\''locale'\'':'\''en-us'\''
}'
```

Note there are three types of voice profile:

- Text-dependent verification
- Text-independent verification
- Text-independent identification

In this case, you create a text-dependent verification voice profile. You should receive the following response.

```
{
    "remainingEnrollmentsCount": 3,
    "locale": "en-us",
    "createdDateTime": "2020-09-29T14:54:29.683Z",
    "enrollmentStatus": "Enrolling",
    "modelVersion": null,
    "profileId": "714ce523-de76-4220-b93f-7c1cc1882d6e",
    "lastUpdatedDateTime": null,
    "enrollmentsCount": 0,
    "enrollmentsLength": 0.0,
    "enrollmentSpeechLength": 0.0
}
```

Next, you enroll the voice profile. For the `--data-binary` parameter value, specify an audio file on your computer that contains one of the supported passphrases, such as "my voice is my passport, verify me." You can record such an audio file with an app such as Windows Voice Recorder, or you can generate it using text -to-speech.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-
dependent/profiles/INSERT_PROFILE_ID_HERE/enrollments' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_FILE_PATH_HERE'
```

You should receive the following response.

```
{
    "remainingEnrollmentsCount": 2,
    "passPhrase": "my voice is my passport verify me",
    "profileId": "714ce523-de76-4220-b93f-7c1cc1882d6e",
    "enrollmentStatus": "Enrolling",
    "enrollmentsCount": 1,
    "enrollmentsLength": 3.5,
    "enrollmentsSpeechLength": 2.88,
    "audioLength": 3.5,
    "audioSpeechLength": 2.88
}
```

This response tells you that you need to enroll two more audio samples.

After you have enrolled a total of three audio samples, you should receive the following response.

```
{
    "remainingEnrollmentsCount": 0,
    "passPhrase": "my voice is my passport verify me",
    "profileId": "714ce523-de76-4220-b93f-7c1cc1882d6e",
    "enrollmentStatus": "Enrolled",
    "enrollmentsCount": 3,
    "enrollmentsLength": 10.5,
    "enrollmentsSpeechLength": 8.64,
    "audioLength": 3.5,
    "audioSpeechLength": 2.88
}
```

Now you are ready to verify an audio sample against the voice profile. This audio sample should contain the same passphrase as the samples you used to enroll the voice profile.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-
dependent/profiles/INSERT_PROFILE_ID_HERE/verify' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_FILE_PATH_HERE'
```

You should receive the following response.

```
{
    "recognitionResult": "Accept",
    "score": 1.0
}
```

The `Accept` means the passphrase matched and the verification was successful. The response also contains a similarity score ranging from 0.0-1.0.

To finish, delete the voice profile.

```
curl --location --request DELETE \
'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-dependent/profiles/INSERT_PROFILE_ID_HERE' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE'
```

There is no response.

Text-independent verification

In contrast to **text-dependent** verification, **text-independent** verification:

- Does not require a certain passphrase to be spoken, anything can be spoken
- Does not require three audio samples, but *does* require 20 seconds of total audio

Start by creating a text-independent verification profile.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-independent/profiles' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: application/json' \
--data-raw '{
    '\''locale'\'':'\''en-us'\''
}'
```

You should receive the following response.

```
{
    "remainingEnrollmentsSpeechLength": 20.0,
    "locale": "en-us",
    "createdDateTime": "2020-09-29T16:08:52.409Z",
    "enrollmentStatus": "Enrolling",
    "modelVersion": null,
    "profileId": "3f85dca9-ffc9-4011-bf21-37fad2beb4d2",
    "lastUpdatedDateTime": null,
    "enrollmentsCount": 0,
    "enrollmentsLength": 0.0,
    "enrollmentSpeechLength": 0.0
}
```

Next, enroll the voice profile. Again, rather than submitting three audio samples, you need to submit audio samples that contain a total of 20 seconds of audio.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-
independent/profiles/INSERT_PROFILE_ID_HERE/enrollments' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_FILE_PATH_HERE'
```

Once you have submitted enough audio samples, you should receive the following response.

```
{
    "remainingEnrollmentsSpeechLength": 0.0,
    "profileId": "3f85dca9-ffc9-4011-bf21-37fad2beb4d2",
    "enrollmentStatus": "Enrolled",
    "enrollmentsCount": 1,
    "enrollmentsLength": 33.16,
    "enrollmentsSpeechLength": 29.21,
    "audioLength": 33.16,
    "audioSpeechLength": 29.21
}
```

Now you are ready to verify an audio sample against the voice profile. Again, this audio sample does not need to contain a passphrase. It can contain any speech, as long as it contains a total of at least four seconds of audio.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-
independent/profiles/INSERT_PROFILE_ID_HERE/verify' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_FILE_PATH_HERE'
```

You should receive the following response.

```
{
    "recognitionResult": "Accept",
    "score": 0.9196669459342957
}
```

The `Accept` means the verification was successful. The response also contains a similarity score ranging from 0.0-1.0.

To finish, delete the voice profile.

```
curl --location --request DELETE 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-
independent/profiles/INSERT_PROFILE_ID_HERE' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE'
```

There is no response.

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**, with the main difference being able to verify against multiple voice profiles at once, rather than verifying against a single profile.

Start by creating a text-independent identification profile.

```
# Note Change locale if needed.
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/identification/v2.0/text-independent/profiles'
\
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: application/json' \
--data-raw '{
    '\''locale'\'':'\''en-us'\''
}'
```

You should receive the following response.

```
{
    "remainingEnrollmentsSpeechLength": 20.0,
    "locale": "en-us",
    "createdDateTime": "2020-09-22T17:25:48.642Z",
    "enrollmentStatus": "Enrolling",
    "modelVersion": null,
    "profileId": "de99ab38-36c8-4b82-b137-510907c61fe8",
    "lastUpdatedDateTime": null,
    "enrollmentsCount": 0,
    "enrollmentsLength": 0.0,
    "enrollmentSpeechLength": 0.0
}
```

Next, you enroll the voice profile. Again, you need to submit audio samples that contain a total of 20 seconds of audio. These samples do not need to contain a passphrase.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/identification/v2.0/text-
independent/profiles/INSERT_PROFILE_ID_HERE/enrollments' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_FILE_PATH_HERE'
```

Once you have submitted enough audio samples, you should receive the following response.

```
{
    "remainingEnrollmentsSpeechLength": 0.0,
    "profileId": "de99ab38-36c8-4b82-b137-510907c61fe8",
    "enrollmentStatus": "Enrolled",
    "enrollmentsCount": 2,
    "enrollmentsLength": 36.69,
    "enrollmentsSpeechLength": 31.95,
    "audioLength": 33.16,
    "audioSpeechLength": 29.21
}
```

Now you are ready to identify an audio sample using the voice profile. The identify command accepts a comma-delimited list of possible voice profile IDs. In this case, you'll just pass in the ID of the voice profile you created previously. However, if you want, you can pass in multiple voice profile IDs where each voice profile is enrolled with audio samples from a different voice.

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/identification/v2.0/text-
independent/profiles/identifySingleSpeaker?profileIds=INSERT_PROFILE_ID_HERE' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \
--header 'Content-Type: audio/wav' \
--data-binary 'INSERT_FILE_PATH_HERE'
```

You should receive the following response.

```
Success:
{
    "identifiedProfile": {
        "profileId": "de99ab38-36c8-4b82-b137-510907c61fe8",
        "score": 0.9083486
    },
    "profilesRanking": [
        {
            "profileId": "de99ab38-36c8-4b82-b137-510907c61fe8",
            "score": 0.9083486
        }
    ]
}
```

The response contains the ID of the voice profile that most closely matches the audio sample you submitted. It also contains a list of candidate voice profiles, ranked in order of similarity.

To finish, delete the voice profile.

```
curl --location --request DELETE \
'INSERT_ENDPOINT_HERE/speaker/identification/v2.0/text-independent/profiles/INSERT_PROFILE_ID_HERE' \
--header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE'
```

There is no response.

Next steps

- See the Speaker Recognition reference documentation for detail on classes and functions.
- See C# and C++ samples on GitHub.

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

Keyword recognition

11/23/2021 • 6 minutes to read • Edit Online

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 multi-stage 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 spanning across the edge and cloud:

Accuracy of keyword recognition is measured via the following metrics:

- **Correct accept rate (CA)** – Measures the system's ability to recognize the keyword when it is spoken by an end-user. This is also known as the true positive rate.
- **False accept rate (FA)** – Measures the system's ability to filter out audio that is not the keyword spoken by an end-user. This 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 is not supported.

Custom Keyword for on-device models

The Custom Keyword portal on Speech Studio allows you to 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 using Custom Keyword for generating models, including both Basic and Advanced models. There is also no cost for running models on-device with the Speech SDK.

Types of models

Custom Keyword allows you 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 may not have optimal accuracy characteristics.
Advanced	Best suited for product integration purposes. Models are generated with adaptation of a common base model 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 creating a new model, Custom Keyword automatically generates possible pronunciations of the provided keyword. You can listen to each pronunciation and choose all that closely represent the way you expect end-users to say the keyword. All other pronunciations should not be selected.

It is important to be deliberate about the pronunciations you select to ensure the best accuracy characteristics. For example, choosing more pronunciations than needed can lead to higher false accept rates. Choosing too few pronunciations, where not all expected variations are covered, can lead to lower correct accept rates.

Testing models

Once on-device models are generated by Custom Keyword, they can be tested directly on the portal. The portal allows you 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. There is no tuning or training required for Keyword Verification to work with your keyword. Incremental model updates are continually deployed to the service to improve accuracy and latency, completely transparent to client applications.

Pricing

Keyword Verification is always used in combination with Speech-to-text, and there is no cost to using Keyword Verification beyond the cost of Speech-to-text.

Keyword Verification and Speech-to-text

When Keyword Verification is used, it is always in combination with Speech-to-text. Both services run in parallel. This means that audio is sent to both services for simultaneous processing.

Running Keyword Verification and Speech-to-text in parallel yields the following benefits:

- **No additional latency on Speech-to-text results** – Parallel execution means Keyword Verification adds no latency, and the client receives Speech-to-text results just as quickly. If Keyword Verification determines the keyword was not present in the audio, Speech-to-text processing is terminated, which protects against unnecessary Speech-to-text processing. However, network and cloud model processing increases the user-perceived latency of voice activation. For details, see Recommendations and guidelines.
- **Forced keyword prefix in Speech-to-text results** – Speech-to-text processing will ensure that the results sent to the client are prefixed with the keyword. This allows for increased accuracy in the Speech-to-text results for speech that follows the keyword.
- **Increased Speech-to-text timeout** – Due to the expected presence of the keyword at the beginning of audio, Speech-to-text will allow for a longer pause of up to 5 seconds after the keyword, before determining end of speech and terminating Speech-to-text processing. This ensures the end-user experience is correctly handled for both staged commands (*<keyword> <pause> <command>*) and chained commands (*<keyword> <command>*).

Keyword Verification responses and latency considerations

For each request to the service, Keyword Verification will return 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 does not include network cost between the client and Azure Speech services.

KEYWORD VERIFICATION RESPONSE	DESCRIPTION
Accepted	Indicates the service believed the keyword was present in the audio stream provided as part of the request.
Rejected	Indicates the service believed the keyword was not 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 will process a maximum of two seconds of audio to search for the keyword. If the keyword is determined not to be present in the two seconds, the service will time out and signal a rejected response to the client.

Using Keyword Verification with on-device models from Custom Keyword

The Speech SDK facilitates seamless use of on-device models generated using Custom Keyword with Keyword Verification and Speech-to-text. It transparently handles:

- Audio gating to Keyword Verification & Speech recognition based on the outcome of on-device model.
- Communicating the keyword to the Keyword Verification service.

- Communicating any additional metadata to the cloud for orchestrating the end-to-end scenario.

You do not need to explicitly specify any configuration parameters. All necessary information will automatically be extracted from the on-device model generated by Custom Keyword.

The sample and tutorials linked below 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 facilitates easy use of personalized on-device keyword recognition models generated with Custom Keyword and the Keyword Verification service. To ensure your product needs can be met, the SDK supports two scenarios:

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's Keyword Verification and Speech-to-text services. This is the most common scenario.	• 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.	• 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.

Get started with Custom Keyword

11/23/2021 • 3 minutes to read • Edit Online

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

The steps in this article require a Speech subscription and the Speech SDK. If you don't already have a subscription, try the Speech service for free. To get the SDK, see the install guide for your platform.

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).

4. Select your project's name from the list.

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.

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.

```
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.

```
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 quickstart 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.

See the sample on GitHub for using your Custom Keyword model with the Python SDK.

See the sample on GitHub for using your Custom Keyword model with the Objective C SDK.

Next steps

Get the Speech SDK

Recommendations and guidelines for keyword recognition

11/23/2021 • 2 minutes to read • Edit Online

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.

Keyword recognition region support

11/23/2021 • 2 minutes to read • Edit Online

REGION	CUSTOM KEYWORD (BASIC MODELS)	CUSTOM KEYWORD (ADVANCED MODELS)	KEYWORD VERIFICATION
West US	Yes	No	Yes
West US 2	Yes	Yes	Yes
East US	Yes	Yes	Yes
East US 2	Yes	Yes	Yes
West Central US	Yes	No	Yes
South Central US	Yes	Yes	Yes
West Europe	Yes	Yes	Yes
North Europe	Yes	Yes	Yes
UK South	Yes	Yes	No
East Asia	Yes	No	Yes
Southeast Asia	Yes	Yes	Yes
Central India	Yes	Yes	Yes
Japan East	Yes	No	Yes
Japan West	Yes	No	No
Australia East	Yes	Yes	No
Brazil South	Yes	No	No
Canada Central	Yes	No	No
Korea Central	Yes	No	No
France Central	Yes	No	No
North Central US	Yes	Yes	No
Central US	Yes	No	No
South Africa North	Yes	No	No

What is Conversation Transcription (Preview)?

11/23/2021 • 3 minutes to read • Edit Online

Conversation Transcription is a speech-to-text solution that combines speech recognition, speaker identification, and sentence attribution to each speaker (also known as *diarization*) to provide real-time and/or asynchronous transcription of any conversation. Conversation Transcription distinguishes speakers in a conversation to determine who said what and when, and makes it easy for developers to add speech-to-text to their applications that perform multi-speaker diarization.

Key features

- **Timestamps** - each speaker utterance has a timestamp, so that you can easily find when a phrase was said.
- **Readable transcripts** - transcripts have formatting and punctuation added automatically to ensure the text closely matches what was being said.
- **User profiles** - user profiles are generated by collecting user voice samples and sending them to signature generation.
- **Speaker identification** - speakers are identified using user profiles and a *speaker identifier* is assigned to each.
- **Multi-speaker diarization** - determine who said what by synthesizing the audio stream with each speaker identifier.
- **Real-time transcription** – provide live transcripts of who is saying what and when while the conversation is happening.
- **asynchronous transcription** – provide transcripts with higher accuracy by using a multichannel audio stream.

> **NOTE**
>
> Although Conversation Transcription does not put a limit on the number of speakers in the room, it is optimized for 2-10 speakers per session.

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 is important to have transcription in real time. Conversation Transcription in real-time mode takes meeting audio and determines who is saying what, allowing all meeting participants to follow the transcript and participate in the meeting without a delay.

Improved efficiency

Meeting participants can focus on the meeting and leave note-taking to Conversation Transcription. Participants can actively engage in the meeting and quickly follow up on next steps, using the transcript instead of taking notes and potentially missing something during the meeting.

How it works

This is a high-level overview of how Conversation Transcription works.

Expected inputs

- **Multi-channel audio stream** – For specification and design details, see Microsoft Speech Device SDK Microphone. To learn more or purchase a development kit, see Get Microsoft Speech Device SDK.
- **User voice samples** – Conversation Transcription needs user profiles in advance of the conversation for speaker identification. You will need to collect audio recordings from each user, then send the recordings to the Signature Generation Service to validate the audio and generate user profiles.

> **NOTE**
>
> User voice samples for voice signatures are required for speaker identification. Speakers who do not have voice samples will be recognized as "Unidentified". Unidentified speakers can still be differentiated when the `DifferentiateGuestSpeakers` property is enabled (see example below). The transcription output will then show speakers as "Guest_0", "Guest_1", etc. instead of recognizing as pre-enrolled specific speaker names.
>
> ```
> config.SetProperty("DifferentiateGuestSpeakers", "true");
> ```

Real-time vs. asynchronous

Conversation Transcription offers three transcription modes:

Real-time

Audio data is processed live to return speaker identifier + transcript. Select this mode if your transcription solution requirement is to provide conversation participants a live transcript view of their ongoing conversation. For example, building an application to make meetings more accessible the deaf and hard of hearing participants is an ideal use case for real-time transcription.

Asynchronous

Audio data is batch processed to return speaker identifier and transcript. Select this mode if your transcription solution requirement is to provide higher accuracy without live transcript view. For example, if you want to build an application to allow meeting participants to easily catch up on missed meetings, then use the asynchronous transcription mode to get high-accuracy transcription results.

Real-time plus asynchronous

Audio data is processed live to return speaker identifier + transcript, and, in addition, a request is created to also get a high-accuracy transcript through asynchronous processing. Select this mode if your application has a need

for real-time transcription but also requires a higher accuracy transcript for use after the conversation or meeting occurred.

Language support

Currently, Conversation Transcription supports all speech-to-text languages in the following regions: `centralus`, `eastasia`, `eastus`, `westeurope`. If you require additional locale support, contact the Conversation Transcription Feature Crew.

Next steps

Transcribe conversations in real time

Get started with real-time Conversation Transcription

11/23/2021 • 8 minutes to read • Edit Online

The Speech SDK's **ConversationTranscriber** API allows you to transcribe meetings and other conversations with the ability to add, remove, and identify multiple participants by streaming audio to the Speech service using `PullStream` or `PushStream`. You first create voice signatures for each participant using the REST API, and then use the voice signatures with the 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.
- The Speech Devices SDK provides suitable devices and a sample app demonstrating Conversation Transcription.

Prerequisites

This article assumes that you have an Azure account and Speech service subscription. If you don't have an account and subscription, try the Speech service for free.

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK for JavaScript . Depending on your platform, use the following instructions:

- Node.js
- Web Browser

Create voice signatures

(You can skip this step if you do not want to use pre-enrolled user profiles to identify specific participants.)

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. 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 thirty 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 JavaScript. Note that you need to substitute real information for your `subscriptionKey`, `region`, and the path to a sample `.wav` file.

```
const fs = require('fs');
const axios = require('axios');
const formData = require('form-data');

const subscriptionKey = 'your-subscription-key';
const region = 'your-region';

async function createProfile() {
    let form = new formData();
    form.append('file', fs.createReadStream('path-to-voice-sample.wav'));
    let headers = form.getHeaders();
    headers['Ocp-Apim-Subscription-Key'] = subscriptionKey;

    let url =
`https://signature.${region}.cts.speech.microsoft.com/api/v1/Signature/GenerateVoiceSignatureFromFormData`;
    let response = await axios.post(url, form, { headers: headers });

    // get signature from response, serialize to json string
    return JSON.stringify(response.data.Signature);
}

async function main() {
    // use this voiceSignature string with conversation transcription calls below
    let voiceSignatureString = await createProfile();
    console.log(voiceSignatureString);
}
main();
```

Running this script returns a voice signature string in the variable `voiceSignatureString`. 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 do not 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 a push stream to use for transcription, and writes the sample `.wav` file to it.
- Creates a `Conversation` using `createConversationAsync()`.
- Creates a `ConversationTranscriber` using the constructor.
- Adds participants to the conversation. The strings `voiceSignatureStringUser1` and `voiceSignatureStringUser2` should come as output from the steps above.

- Registers to events and begins transcription.
- If you want to differentiate speakers without providing voice samples, please enable `DifferentiateGuestSpeakers` feature as in Conversation Transcription Overview.

```
(function() {
    "use strict";
    var sdk = require("microsoft-cognitiveservices-speech-sdk");
    var fs = require("fs");

    var subscriptionKey = "your-subscription-key";
    var region = "your-region";
    var filepath = "audio-file-to-transcribe.wav"; // 8-channel audio

    var speechTranslationConfig = sdk.SpeechTranslationConfig.fromSubscription(subscriptionKey, region);
    var audioConfig = sdk.AudioConfig.fromWavFileInput(fs.readFileSync(filepath));
    speechTranslationConfig.setProperty("ConversationTranscriptionInRoomAndOnline", "true");

    // en-us by default. Adding this code to specify other languages, like zh-cn.
    speechTranslationConfig.speechRecognitionLanguage = "en-US";

    // create conversation and transcriber
    var conversation = sdk.Conversation.createConversationAsync(speechTranslationConfig, "myConversation");
    var transcriber = new sdk.ConversationTranscriber(audioConfig);

    // attach the transcriber to the conversation
    transcriber.joinConversationAsync(conversation,
    function () {
        // add first participant using voiceSignature created in enrollment step
        var user1 = sdk.Participant.From("user1@example.com", "en-us", voiceSignatureStringUser1);
        conversation.addParticipantAsync(user1,
        function () {
            // add second participant using voiceSignature created in enrollment step
            var user2 = sdk.Participant.From("user2@example.com", "en-us", voiceSignatureStringUser2);
            conversation.addParticipantAsync(user2,
            function () {
                transcriber.sessionStarted = function(s, e) {
                console.log("(sessionStarted)");
                };
                transcriber.sessionStopped = function(s, e) {
                console.log("(sessionStopped)");
                };
                transcriber.canceled = function(s, e) {
                console.log("(canceled)");
                };
                transcriber.transcribed = function(s, e) {
                console.log("(transcribed) text: " + e.result.text);
                console.log("(transcribed) speakerId: " + e.result.speakerId);
                };

                // begin conversation transcription
                transcriber.startTranscribingAsync(
                function () { },
                function (err) {
                    console.trace("err - starting transcription: " + err);
                });
        },
        function (err) {
            console.trace("err - adding user1: " + err);
        });
    },
    function (err) {
        console.trace("err - adding user2: " + err);
    });
    },
    function (err) {
    console.trace("err - " + err);
    });
}());
```

Install the Speech SDK

Before you can do anything, you'll need to install the Speech SDK. Depending on your platform, use the following instructions:

- .NET Framework
- .NET Core
- Unity
- UWP
- Xamarin

Create voice signatures

(You can skip this step if you do not want to use pre-enrolled user profiles to identify specific participants.)

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. 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 thirty 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# . Note that you need to substitute real information for your `subscriptionKey`, `region`, and the path to a sample `.wav` file.

```
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/GenerateVoiceSignatureFromByteArray", 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 do not 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, please enable `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.

```
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 update the subscription info?");
                        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 ROOBO device sample code Azure Kinect Dev Kit sample code

Speech service for telephony data

11/23/2021 • 10 minutes to read • Edit Online

Telephony data that is generated through landlines, mobile phones, and radios are typically low quality, and narrowband in the range of 8 KHz, which creates challenges when converting speech-to-text. The latest speech recognition models from the Speech service excel at transcribing this telephony data, even in cases when the data is difficult for a human to understand. These models are trained with large volumes of telephony data, and have best-in-market recognition accuracy, even in noisy environments.

A common scenario for speech-to-text is transcribing large volumes of telephony data that may come from various systems, such as Interactive Voice Response (IVR). The audio these systems provide can be stereo or mono, and raw with little-to-no post processing done on the signal. Using the Speech service and the Unified speech model, a business can get high-quality transcriptions, whatever systems are used to capture audio.

Telephony data can be used to better understand your customers' needs, identify new marketing opportunities, or evaluate the performance of call center agents. After the data is transcribed, a business can use the output for purposes such as improved telemetry, identifying key phrases, or analyzing customer sentiment.

The technologies outlined in this page are by Microsoft internally for various support call processing services, both in real-time and batch mode.

Let's review some of the technology and related features the Speech service offers.

> **IMPORTANT**
>
> The Speech service Unified model is trained with diverse data and offers a single-model solution to a number of scenario from Dictation to Telephony analytics.

Azure Technology for Call Centers

Beyond the functional aspect of the Speech service features, their primary purpose – when applied to the call center – is to improve the customer experience. Three clear domains exist in this regard:

- Post-call analytics, which is essentially batch processing of call recordings after the call.
- Real-time analytics, which is processing of the audio signal to extract various insights as the call is taking place (with sentiment being a prominent use case).
- Voice assistants (bots), either driving the dialogue between the customer and the bot in an attempt to solve the customer's issue with no agent participation, or being the application of artificial intelligence (AI) protocols to assist the agent.

A typical architecture diagram of the implementation of a batch scenario is depicted in the picture below

Speech Analytics Technology Components

Whether the domain is post-call or real-time, Azure offers a set of mature and emerging technologies to improve the customer experience.

Speech to text (STT)

Speech-to-text is the most sought-after feature in any call center solution. Because many of the downstream analytics processes rely on transcribed text, the word error rate (*WER*) is of utmost importance. One of the key challenges in call center transcription is the noise that's prevalent in the call center (for example other agents speaking in the background), the rich variety of language locales and dialects as well as the low quality of the actual telephone signal. WER is highly correlated with how well the acoustic and language models are trained for a given locale, thus the ability to customize the model to your locale is important. Our latest Unified version 4.x models are the solution to both transcription accuracy and latency. Trained with tens of thousands of hours of acoustic data and billions of lexical information, Unified models are the most accurate models in the market to transcribe call center data.

Sentiment

Gauging whether the customer had a good experience is one of the most important areas of Speech analytics when applied to the call center space. Our Batch Transcription API offers sentiment analysis per utterance. You can aggregate the set of values obtained as part of a call transcript to determine the sentiment of the call for both your agents and the customer.

Silence (non-talk)

It is not uncommon for 35 percent of a support call to be what we call non-talk time. Some scenarios for which non-talk occurs are: agents looking up prior case history with a customer, agents using tools that allow them to access the customer's desktop and perform functions, customers sitting on hold waiting for a transfer, and so on. It is extremely important to gauge when silence is occurring in a call as there are number of important customer sensitivities that occur around these types of scenarios and where they occur in the call.

Translation

Some companies are experimenting with providing translated transcripts from foreign language support calls so that delivery managers can understand the world-wide experience of their customers. Our translation capabilities are unsurpassed. We can translate audio-to-audio or audio-to-text for a large number of locales.

Text to Speech

Text-to-speech is another important area in implementing bots that interact with the customers. The typical pathway is that the customer speaks, their voice is transcribed to text, the text is analyzed for intents, a response is synthesized based on the recognized intent, and then an asset is either surfaced to the customer or a synthesized voice response is generated. Of course all of this has to occur quickly – thus low-latency is an important component in the success of these systems.

Our end-to-end latency is considerably low for the various technologies involved such as Speech-to-text, LUIS,

Bot Framework, Text-to-speech.

Our new voices are also indistinguishable from human voices. You can use our voices to give your bot its unique personality.

Search

Another staple of analytics is to identify interactions where a specific event or experience has occurred. This is typically done with one of two approaches; either an ad hoc search where the user simply types a phrase and the system responds, or a more structured query where an analyst can create a set of logical statements that identify a scenario in a call, and then each call can be indexed against that set of queries. A good search example is the ubiquitous compliance statement "this call shall be recorded for quality purposes... ". Many companies want to make sure that their agents are providing this disclaimer to customers before the call is actually recorded. Most analytics systems have the ability to trend the behaviors found by query/search algorithms, and this reporting of trends is ultimately one of the most important functions of an analytics system. Through Cognitive services directory your end to end solution can be significantly enhanced with indexing and search capabilities.

Key Phrase Extraction

This area is one of the more challenging analytics applications and one that is benefiting from the application of AI and machine learning. The primary scenario in this case is to infer customer intent. Why is the customer calling? What is the customer problem? Why did the customer have a negative experience? Our Language service provides a set of analytics out of the box for quickly upgrading your end-to-end solution for extracting those important keywords or phrases.

Let's now have a look at the batch processing and the real-time pipelines for speech recognition in a bit more detail.

Batch transcription of call center data

For transcribing bulk audio we developed the Batch Transcription API. The Batch Transcription API was developed to transcribe large amounts of audio data asynchronously. With regard to transcribing call center data, our solution is based on these pillars:

- **Accuracy** - With fourth-generation Unified models, we offer unsurpassed transcription quality.
- **Latency** - We understand that when doing bulk transcriptions, the transcriptions are needed quickly. The transcription jobs initiated via the Batch Transcription API will be queued immediately, and once the job starts running it's performed faster than real-time transcription.
- **Security** - We understand that calls may contain sensitive data. Rest assured that security is one of our highest priorities. Our service has obtained ISO, SOC, HIPAA, PCI certifications.

Call centers generate large volumes of audio data on a daily basis. If your business stores telephony data in a central location, such as Azure Storage, you can use the Batch Transcription API to asynchronously request and receive transcriptions.

A typical solution uses these services:

- The Speech service is used to transcribe speech-to-text. A standard subscription (S0) for the Speech service is required to use the Batch Transcription API. Free subscriptions (F0) will not work.
- Azure Storage is used to store telephony data, and the transcripts returned by the Batch Transcription API. This storage account should use notifications, specifically for when new files are added. These notifications are used to trigger the transcription process.
- Azure Functions is used to create the shared access signatures (SAS) URI for each recording, and trigger the HTTP POST request to start a transcription. Additionally, Azure Functions is used to create requests to retrieve and delete transcriptions using the Batch Transcription API.

Internally we are using the above technologies to support Microsoft customer calls in Batch mode.

Real-time transcription for call center data

Some businesses are required to transcribe conversations in real-time. Real-time transcription can be used to identify key-words and trigger searches for content and resources relevant to the conversation, for monitoring sentiment, to improve accessibility, or to provide translations for customers and agents who aren't native speakers.

For scenarios that require real-time transcription, we recommend using the Speech SDK. Currently, speech-to-text is available in more than 20 languages, and the SDK is available in C++, C#, Java, Python, JavaScript, Objective-C, and Go. Samples are available in each language on GitHub. For the latest news and updates, see Release notes.

Internally we are using the above technologies to analyze in real-time Microsoft customer calls as they happen, as illustrated in the following diagram.

A word on IVRs

The Speech service can be easily integrated in any solution by using either the Speech SDK or the REST API. However, call center transcription may require additional technologies. Typically, a connection between an IVR

system and Azure is required. Although we do not offer such components, here is a description what a connection to an IVR entails.

Several IVR or telephony service products (such as Genesys or AudioCodes) offer integration capabilities that can be leveraged to enable inbound and outbound audio pass-through to an Azure service. Basically, a custom Azure service might provide a specific interface to define phone call sessions (such as Call Start or Call End) and expose a WebSocket API to receive inbound stream audio that is used with the Speech service. Outbound responses, such as conversation transcription or connections with the Bot Framework, can be synthesized with Microsoft's text-to-speech service and returned to the IVR for playback.

Another scenario is direct integration with Session Initiation Protocol (SIP). An Azure service connects to a SIP Server, thus getting an inbound stream and an outbound stream, which is used for the speech-to-text and text-to-speech phases. To connect to a SIP Server there are commercial software offerings, such as Ozeki SDK, or the Teams calling and meetings API (currently in beta), that are designed to support this type of scenario for audio calls.

Customize existing experiences

The Speech service works well with built-in models. However, you may want to further customize and tune the experience for your product or environment. Customization options range from acoustic model tuning to unique voice fonts for your brand. After you've built a custom model, you can use it with any of the Speech service features in real-time or batch mode.

SPEECH SERVICE	MODEL	DESCRIPTION
Speech-to-text	Acoustic model	Create a custom acoustic model for applications, tools, or devices that are used in particular environments like in a car or on a factory floor, each with specific recording conditions. Examples include accented speech, specific background noises, or using a specific microphone for recording.
	Language model	Create a custom language model to improve transcription of industry-specific vocabulary and grammar, such as medical terminology, or IT jargon.
	Pronunciation model	With a custom pronunciation model, you can define the phonetic form and display for a word or term. It's useful for handling customized terms, such as product names or acronyms. All you need to get started is a pronunciation file, which is a simple `.txt` file.
Text-to-speech	Voice font	Custom voice fonts allow you to create a recognizable, one-of-a-kind voice for your brand. It only takes a small amount of data to get started. The more data that you provide, the more natural and human-like your voice font will sound.

Sample code

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

- Speech-to-text and speech translation samples (SDK)
- Batch transcription samples (REST)
- Text-to-speech samples (REST)

Reference docs

- Speech SDK
- Speech Devices SDK
- REST API: Speech-to-text
- REST API: Text-to-speech
- REST API: Batch transcription and customization

Next steps

Get a Speech service subscription key for free

Asynchronous Conversation Transcription

11/23/2021 • 6 minutes to read • Edit Online

In this article, asynchronous Conversation Transcription is demonstrated using the **RemoteConversationTranscriptionClient** API. If you have configured Conversation Transcription to do asynchronous transcription and have a `conversationId`, you can obtain the transcription associated with that `conversationId` using the **RemoteConversationTranscriptionClient** API.

Asynchronous vs. real-time + asynchronous

With asynchronous transcription, you stream the conversation audio, but don't need a transcription returned in real time. Instead, after the audio is sent, use the `conversationId` of `Conversation` to query for the status of the asynchronous transcription. When the asynchronous transcription is ready, you'll get a `RemoteConversationTranscriptionResult`.

With real-time plus asynchronous, you get the transcription in real time, but also get the transcription by querying with the `conversationId` (similar to asynchronous scenario).

Two steps are required to accomplish asynchronous transcription. The first step is to upload the audio, choosing either asynchronous only or real-time plus asynchronous. The second step is to get the transcription results.

Upload the audio

The first step for asynchronous transcription is to send the audio to the Conversation Transcription Service using the Speech SDK (version 1.13.0 or above).

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.

```
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);
    await Task.WhenAny(finishedTaskCompletionSource.Task, Task.Delay(TimeSpan.FromSeconds(10)));
    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.SetServiceProperty("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:

```
// 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.

```
// 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");
```

Upload the audio

Before asynchronous transcription can be performed, you need to send the audio to Conversation Transcription Service using Microsoft Cognitive Speech client SDK (version 1.8.0 or above).

This example code shows how to create conversation transcriber for asynchronous-only mode. In order to stream audio to the transcriber, you will need to add audio streaming code derived from Transcribe conversations in real time with the Speech SDK. Refer to the **Limitations** section of that topic to see the supported platforms and languages APIs.

```
// Create the speech config object
// Substitute real information for "YourSubscriptionKey" and "Region"
SpeechConfig speechConfig = SpeechConfig.fromSubscription("YourSubscriptionKey", "Region");
speechConfig.setProperty("ConversationTranscriptionInRoomAndOnline", "true");

// Set the property for asynchronous transcription
speechConfig.setServiceProperty("transcriptionMode", "async", ServicePropertyChannel.UriQueryParameter);

// Set the property for real-time plus asynchronous transcription
//speechConfig.setServiceProperty("transcriptionMode", "RealTimeAndAsync",
ServicePropertyChannel.UriQueryParameter);

// pick a conversation Id that is a GUID.
String conversationId = UUID.randomUUID().toString();

// Create a Conversation
Future<Conversation> conversationFuture = Conversation.createConversationAsync(speechConfig,
conversationId);
Conversation conversation = conversationFuture.get();
```

```
// Create an audio stream from a wav file or from the default microphone if you want to stream live audio
from the supported devices
// Replace with your own audio file name and Helper class which implements AudioConfig using
PullAudioInputStreamCallback
PullAudioInputStreamCallback wavfilePullStreamCallback =
Helper.OpenWavFile("16kHz16Bits8channelsOfRecordedPCMAudio.wav");
// Create an audio stream format assuming the file used above is 16kHz, 16 bits and 8 channel pcm wav file
AudioStreamFormat audioStreamFormat = AudioStreamFormat.getWaveFormatPCM((long)16000, (short)16,(short)8);
// Create an input stream
AudioInputStream audioStream = AudioInputStream.createPullStream(wavfilePullStreamCallback,
audioStreamFormat);

// Create a conversation transcriber
ConversationTranscriber transcriber = new ConversationTranscriber(AudioConfig.fromStreamInput(audioStream));

// join a conversation
transcriber.joinConversationAsync(conversation);

// Add the event listener for the realtime events
transcriber.transcribed.addEventListener((o, e) -> {
    System.out.println("Conversation transcriber Recognized:" + e.toString());
});

transcriber.canceled.addEventListener((o, e) -> {
    System.out.println("Conversation transcriber canceled:" + e.toString());
    try {
        transcriber.stopTranscribingAsync().get();
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    } catch (ExecutionException ex) {
        ex.printStackTrace();
    }
});

transcriber.sessionStopped.addEventListener((o, e) -> {
    System.out.println("Conversation transcriber stopped:" + e.toString());

    try {
        transcriber.stopTranscribingAsync().get();
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    } catch (ExecutionException ex) {
        ex.printStackTrace();
    }
});

// start the transcription.
Future<?> future = transcriber.startTranscribingAsync();
...
```

If you want real-time *plus* asynchronous, comment and uncomment the appropriate lines of code as follows:

```
// Set the property for asynchronous transcription
//speechConfig.setServiceProperty("transcriptionMode", "async", ServicePropertyChannel.UriQueryParameter);

// Set the property for real-time plus asynchronous transcription
speechConfig.setServiceProperty("transcriptionMode", "RealTimeAndAsync",
ServicePropertyChannel.UriQueryParameter);
```

Get transcription results

For the code shown here, you need **remote-conversation version 1.8.0**, supported only for Java (1.8.0 or above) on Windows, and Linux.

Obtaining the async conversation client SDK

You can obtain **remote-conversation** by editing your pom.xml file as follows.

1. At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK:

```
<repositories>
  <repository>
    <id>maven-cognitiveservices-speech</id>
    <name>Microsoft Cognitive Services Speech Maven Repository</name>
    <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
  </repository>
</repositories>
```

2. Also add a `dependencies` element, with the remoteconversation-client-sdk 1.8.0 as a dependency:

```
<dependencies>
  <dependency>
    <groupId>com.microsoft.cognitiveservices.speech.remoteconversation</groupId>
    <artifactId>remote-conversation</artifactId>
    <version>1.8.0</version>
  </dependency>
</dependencies>
```

3. Save the changes

Sample transcription code

After you have the `conversationId`, create a remote conversation transcription client **RemoteConversationTranscriptionClient** at the client application to query the status of the asynchronous transcription. Use **getTranscriptionOperation** method in **RemoteConversationTranscriptionClient** to get a PollerFlux object. The PollerFlux object will have information about the remote operation status **RemoteConversationTranscriptionOperation** and the final result **RemoteConversationTranscriptionResult**. Once the operation has finished, get **RemoteConversationTranscriptionResult** by calling **getFinalResult** on a SyncPoller. In this code we simply print the result contents to system output.

```
// Create the speech config object
SpeechConfig speechConfig = SpeechConfig.fromSubscription("YourSubscriptionKey", "Region");

// Create a remote Conversation Transcription client
RemoteConversationTranscriptionClient client = new RemoteConversationTranscriptionClient(speechConfig);

// Get the PollerFlux for the remote operation
PollerFlux<RemoteConversationTranscriptionOperation, RemoteConversationTranscriptionResult>
remoteTranscriptionOperation = client.getTranscriptionOperation(conversationId);

// Subscribe to PollerFlux to get the remote operation status
remoteTranscriptionOperation.subscribe(
        pollResponse -> {
            System.out.println("Poll response status : " + pollResponse.getStatus());
            System.out.println("Poll response status : " + pollResponse.getValue().getServiceStatus());
        }
);

// Obtain the blocking operation using getSyncPoller
SyncPoller<RemoteConversationTranscriptionOperation, RemoteConversationTranscriptionResult>
blockingOperation =  remoteTranscriptionOperation.getSyncPoller();

// Wait for the operation to finish
blockingOperation.waitForCompletion();

// Get the final result response
RemoteConversationTranscriptionResult resultResponse = blockingOperation.getFinalResult();

// Print the result
if(resultResponse != null) {
    if(resultResponse.getConversationTranscriptionResults() != null) {
        for (int i = 0; i < resultResponse.getConversationTranscriptionResults().size(); i++) {
            ConversationTranscriptionResult result =
resultResponse.getConversationTranscriptionResults().get(i);

System.out.println(result.getProperties().getProperty(PropertyId.SpeechServiceResponse_JsonResult.name()));

System.out.println(result.getProperties().getProperty(PropertyId.SpeechServiceResponse_JsonResult));
            System.out.println(result.getOffset());
            System.out.println(result.getDuration());
            System.out.println(result.getUserId());
            System.out.println(result.getReason());
            System.out.println(result.getResultId());
            System.out.println(result.getText());
            System.out.println(result.toString());
        }
    }
}

System.out.println("Operation finished");
```

Next steps

Explore our samples on GitHub

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

What is Multi-device Conversation (Preview)?

11/23/2021 • 4 minutes to read • Edit Online

Multi-device conversation makes it easy to create a speech or text conversation between multiple clients and coordinate the messages sent between them.

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 language(s).
- **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 language(s) 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
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

11/23/2021 • 13 minutes to read • Edit Online

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.

You can view or download all Speech SDK C# Samples on GitHub.

Prerequisites

Before you get started, make sure to:

- Create an Azure 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:

```
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.Write("left");
```

```
                                break;

                            case ParticipantChangedReason.Updated:
                                Console.Write("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**:

```
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($"\tTranslated into '{entry.Key}': '{entry.Value}'");
            }
        };
        conversationTranslator.Transcribed += (s, e) =>
        {
            Console.WriteLine($"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}'");
            }
        };

        // 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:

```
// 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

You can view or download all Speech SDK C++ Samples on GitHub.

Prerequisites

Before you get started, make sure to:

- Create an Azure Speech resource
- Setup your development environment and create an empty project

Add sample code

1. From Visual Studio, open the source file **helloworld.cpp**.
2. Replace all the code with the following snippet:

```
#include "pch.h"

#define WIN32_LEAN_AND_MEAN
#include <Windows.h>

#include <iostream>
#include <thread>
#include <future>
#include <string>
#include <speechapi_cxx.h>

using namespace std::chrono_literals;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Audio;
using namespace Microsoft::CognitiveServices::Speech::Transcription;

// Create a promise we will set when the user presses Ctrl-C
std::promise<bool> promise;
// Create the future we will use to wait for the promise to be fulfilled
std::future<bool> future = promise.get_future();
```

```
void StartNewConversation()
{
    // Set these
    std::string subscriptionKey("YourSubscriptionKey");

    // Replace below with your own region identifier from here: https://aka.ms/speech/sdkregion
    std::string region("YourServiceRegion");

    std::string speechLanguage("en-US");

    // Create the conversation object you'll need to manage the conversation
    auto speechConfig = SpeechConfig::FromSubscription(subscriptionKey, region);
    speechConfig->SetSpeechRecognitionLanguage(speechLanguage);
    auto conversation = Conversation::CreateConversationAsync(speechConfig).get();

    // Start the conversation so you and others can join
    conversation->StartConversationAsync().get();

    // Get the conversation ID. It will be up to your scenario to determine how this is shared with
other participants.
    std::cout << "Created conversation: " << conversation->GetConversationId() << std::endl;

    // You can now call various commands to manage the room. For example:
    conversation->MuteAllParticipantsAsync().get();

    // Create the conversation translator you'll need to send audio, send IMs, and receive
conversation events
    auto audioConfig = AudioConfig::FromDefaultMicrophoneInput();
    auto conversationTranslator = ConversationTranslator::FromConfig(audioConfig);

    // Add any event handlers
    conversationTranslator->SessionStarted += [](const SessionEventArgs& args)
    {
        std::cout << "Session started " << args.SessionId << std::endl;
    };
    conversationTranslator->SessionStopped += [](const SessionEventArgs& args)
    {
        std::cout << "Session stopped " << args.SessionId << std::endl;
    };
    conversationTranslator->Canceled += [](const ConversationTranslationCanceledEventArgs& args)
    {
        switch (args.Reason)
        {
            case CancellationReason::EndOfStream:
                std::cout << "End of audio reached" << std::endl;
                break;

            case CancellationReason::Error:
                std::cout << "Canceled due to error. " << (long)args.ErrorCode << ": " <<
args.ErrorDetails << std::endl;
                break;
        }
    };
    conversationTranslator->ConversationExpiration += [](const ConversationExpirationEventArgs& args)
    {
        std::cout << "Conversation will expire in " << args.ExpirationTime.count() << " minutes" <<
std::endl;
    };
    conversationTranslator->ParticipantsChanged += [](const ConversationParticipantsChangedEventArgs&
args)
    {
        std::cout << "The following participant(s) have ";
        switch (args.Reason)
        {
            case ParticipantChangedReason::JoinedConversation:
                std::cout << "joined";
                break;
```

```
            case ParticipantChangedReason::LeftConversation:
                std::cout << "left";
                break;

            case ParticipantChangedReason::Updated:
                std::cout << "been updated";
                break;
        }

        std::cout << ":" << std::endl;

        for(std::shared_ptr<Participant> participant : args.Participants)
        {
            std::cout << "\t" << participant->DisplayName << std::endl;
        }
    };
    conversationTranslator->Transcribing += [](const ConversationTranslationEventArgs& args)
    {
        std::cout << "Received a partial transcription from " << args.Result->ParticipantId << ": "
<< args.Result->Text << std::endl;
        for (const auto& entry : args.Result->Translations)
        {
            std::cout << "\t" << entry.first << ": " << entry.second << std::endl;
        }
    };
    conversationTranslator->Transcribed += [](const ConversationTranslationEventArgs& args)
    {
        std::cout << "Received a transcription from " << args.Result->ParticipantId << ": " <<
args.Result->Text << std::endl;
        for (const auto& entry : args.Result->Translations)
        {
            std::cout << "\t" << entry.first << ": " << entry.second << std::endl;
        }
    };
    conversationTranslator->TextMessageReceived += [](const ConversationTranslationEventArgs& args)
    {
        std::cout << "Received an instant message from " << args.Result->ParticipantId << ": " <<
args.Result->Text << std::endl;
        for (const auto& entry : args.Result->Translations)
        {
            std::cout << "\t" << entry.first << ": " << entry.second << std::endl;
        }
    };

    // Join the conversation so you can start receiving events
    conversationTranslator->JoinConversationAsync(conversation, "Test Host").get();

    // Send an instant message
    conversationTranslator->SendTextMessageAsync("This is a short test message").get();

    // Start sending audio
    conversationTranslator->StartTranscribingAsync().get();
    std::cout << "Started transcribing. Press Ctrl + C to stop" << std::endl;
    future.get(); // wait for Ctrl - C to be pressed

    // Stop audio capture
    conversationTranslator->StopTranscribingAsync().get();

    // Leave the conversation. You will stop receiving events
    conversationTranslator->LeaveConversationAsync().get();

    // Once you are done, remember to delete the conversation.
    conversation->EndConversationAsync().get(); // You will not be able to rejoin after this
    conversation->DeleteConversationAsync().get(); // All participants still in the room will be
ejected
}

int main()
{
```

```
{
    // Register a handler for the Ctrl - C callback
    SetConsoleCtrlHandler(
        [](DWORD dwCtrlType) -> BOOL
        {
            if (dwCtrlType == CTRL_C_EVENT)
            {
                // Signal that the user has pressed ctrl + C
                promise.set_value(true);
                return TRUE;
            }

            return FALSE;
        },
        TRUE);

    StartNewConversation();
}
```

3. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
4. Replace the string `YourServiceRegion` with the region associated with your subscription.
5. 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` on your keyboard to stop audio capture.

> **NOTE**
>
> You may see a message from Visual Studio about an exception similar to: `Exception thrown at 0x76EB90BF (KernelBase.dll) in helloworld.exe: 0x40010005: Control-C.` You can safely ignore this.
>
> Press F5 to continue.

Build and run the application to join an existing conversation

1. Copy and paste the following function into your **helloworld.cpp** just before the `int main()` function:

```
void JoinExistingConversation(const std::string& conversationId)
{
    std::string speechLanguage("en-US");

    // You'll now need to create a ConversationTranslator to send audio, send IMs, and receive
conversation events
    auto audioConfig = AudioConfig::FromDefaultMicrophoneInput();
    auto conversationTranslator = ConversationTranslator::FromConfig(audioConfig);

    // Attach event handlers here. For example:
    conversationTranslator->Transcribing += [](const ConversationTranslationEventArgs& args)
    {
        std::cout << "Received a partial transcription from " << args.Result->ParticipantId << ": "
<< args.Result->Text << std::endl;
        for (const auto& entry : args.Result->Translations)
        {
            std::cout << "\t" << entry.first << ": " << entry.second << std::endl;
        }
    };
    conversationTranslator->Transcribed += [](const ConversationTranslationEventArgs& args)
    {
        std::cout << "Received a transcription from " << args.Result->ParticipantId << ": " <<
args.Result->Text << std::endl;
        for (const auto& entry : args.Result->Translations)
        {
            std::cout << "\t" << entry.first << ": " << entry.second << std::endl;
        }
    };
    conversationTranslator->TextMessageReceived += [](const ConversationTranslationEventArgs& args)
    {
        std::cout << "Received an instant message from " << args.Result->ParticipantId << ": " <<
args.Result->Text << std::endl;
        for (const auto& entry : args.Result->Translations)
        {
            std::cout << "\t" << entry.first << ": " << entry.second << std::endl;
        }
    };

    // Join the conversation
    conversationTranslator->JoinConversationAsync(conversationId, "participant",
speechLanguage).get();

    // Start sending audio
    conversationTranslator->StartTranscribingAsync().get();
    std::cout << "Started transcribing. Press Ctrl + C to stop" << std::endl;
    future.get(); // wait for Ctrl - C to be pressed

    // Stop audio capture
    conversationTranslator->StopTranscribingAsync().get();

    // Once you are done, leave the conversation
    conversationTranslator->LeaveConversationAsync().get();
}
```

2. Replace `StartNewConversation();` in your `int main()` function with:

```
// Set this to the conversation you want to join
JoinExistingConversation("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 `YourConversationId` in your `int main()` function with the conversation ID from the previous step.
5. From the menu bar, select **Build** > **Build Solution** to build the application. The code should compile without errors.
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.

> **NOTE**
> You may see a message from Visual Studio about an exception similar to:
> `Exception thrown at 0x76EB90BF (KernelBase.dll) in helloworld.exe: 0x40010005: Control-C.` You can safely ignore this.
>
> Press F5 to continue.

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

What is a voice assistant?

11/23/2021 • 3 minutes to read • Edit Online

Voice assistants using the Speech service empowers developers to create natural, human-like conversational interfaces for their applications and experiences.

The voice assistant service provides fast, reliable interaction between a device and an assistant implementation that uses either (1) Direct Line Speech (via Azure Bot Service) for adding voice capabilities to your bots, or, (2) Custom Commands for voice commanding scenarios.

Choosing an assistant solution

The first step to creating a voice assistant is to decide what it should do. The Speech service provides multiple, complementary solutions for crafting your assistant interactions. You can add voice in and voice out capabilities to your flexible and versatile bot built using Azure Bot Service with the Direct Line Speech channel, or leverage the simplicity of authoring a Custom Commands app for straightforward voice commanding scenarios.

IF YOU WANT...	THEN CONSIDER...	FOR EXAMPLE...
Open-ended conversation with robust skills integration and full deployment control	Azure Bot Service bot with Direct Line Speech channel	• "I need to go to Seattle" • "What kind of pizza can I order?"
Voice commanding or simple task-oriented conversations with simplified authoring and hosting	Custom Commands	• "Turn on the overhead light" • "Make it 5 degrees warmer" • Other samples available here

We recommend Direct Line Speech as the best default choice if you aren't yet sure what you'd like your assistant to handle. It offers integration with a rich set of tools and authoring aids such as the Virtual Assistant Solution and Enterprise Template and the QnA Maker service to build on common patterns and use your existing knowledge sources.

Custom Commands 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.

Reference Architecture for building a voice assistant 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 with a custom keyword like "Hey Contoso." An app does this with a custom keyword engine in the Speech SDK, which can be configured with a custom keyword that you can generate here. Voice assistants can use service-side keyword verification to improve the accuracy of the keyword activation (versus the device alone).
Speech to text	Voice assistants convert real-time audio into recognized text using 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 using 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 TTS voice that gives a voice to your brand. To learn more, contact us.

Getting started with voice assistants

We offer quickstarts designed to have you running code in less than 10 minutes. This table includes a list of voice assistant quickstarts, organized by language.

- Quickstart: Create a custom voice assistant using Direct Line Speech
- Quickstart: Build a voice commanding app using Custom Commands

Sample code and Tutorials

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 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

Customization

Voice assistants built using Azure Speech services can use the full range of customization options.

- Custom Speech
- Custom Voice
- Custom Keyword

> **NOTE**
>
> Customization options vary by language/locale (see Supported languages).

Next steps

- Get a Speech service subscription key for free
- Learn more about Custom Commands

- Learn more about Direct Line Speech
- Get the Speech SDK

What is Custom Commands?

11/23/2021 • 2 minutes to read • Edit Online

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

Next steps

- Get a Speech service subscription key for free
- View our Voice Assistants repo on GitHub for samples
- Go to the Speech Studio to try out Custom Commands
- Get the Speech SDK

Create a voice assistant using Custom Commands

11/23/2021 • 3 minutes to read • Edit Online

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 subscriptions created in these regions:

- West US
- West US2
- East US
- East US2
- West Central US
- North Europe
- West Europe
- East Asia
- Southeast Asia
- Central India

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 subscriptions.

   > **NOTE**
   > If you don't see the select subscription page, you can navigate there by choosing "Speech resources" from the settings menu on the top bar.

3. Select your Speech subscription, 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 subscription.

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.

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**.

> **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**.

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

11/23/2021 • 19 minutes to read • Edit Online

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.

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

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:

```
Turn on the tv
```

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	`ConfirmationResponse`
Conditions	None
Actions	**Send speech response** > **Simple editor** > **First variation** > `Ok, setting temperature to 40 degrees`

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	`ConfirmationResponse`
Conditions	None
Actions	**Send speech response** > **Simple editor** > **First variation** > `Ok, setting an alarm for 9 am tomorrow`

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**.

   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.

g. Select **Save** to save all configurations of the parameter.

Add a SubjectDevice parameter

1. 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	`TemperatureValue`
Required	Selected
Response for required parameter	**Simple editor** > `What temperature would you like?`
Type	`Number`

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**

CONFIGURATION	SUGGESTED VALUE
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	`DateTime`
Required	Selected
Response for required parameter	Simple editor > `For what time?`
Type	DateTime
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 > `Ok, alarm set for {DateTime}`

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:

```
{
    "fan" : [],
    "refrigerator" : [
        "fridge"
    ],
    "lights" : [
        "bulb",
        "bulbs",
        "light"
        "light bulb"
    ],
    "tv" : [
        "telly",
        "television"
        ]
}
```

Next go the **SubjectDevice** parameter settings page. Set up the following properties.

SETTING	SUGGESTED VALUE
Configuration	**Accept predefined inputs from external catalog**
Catalog endpoint	`getDevices`

SETTING	SUGGESTED VALUE
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
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

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:

      a. Change the name to **Confirm temperature**.
      b. The condition **All required parameters** has already been added.
      c. Add a new action: **Type** > **Send speech response** > **Are you sure you want to set the temperature as {TemperatureValue} degrees?**
      d. In the **Expectations** section, leave the default value of **Expecting confirmation from user**.

   b. Modify the **Confirmation succeeded** interaction rule to handle a successful confirmation (the user said yes).

      a. Change the name to **Confirmation temperature succeeded**.
      b. Leave the existing **Confirmation was successful** condition.
      c. Add a new condition: **Type** > **Required parameters** > **TemperatureValue**.
      d. 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).

   a. Change the name to **Confirmation temperature denied**.
   b. Leave the existing **Confirmation was denied** condition.
   c. Add a new condition: **Type** > **Required parameters** > **TemperatureValue**.
   d. Add a new action: **Type** > **Send speech response** > **No problem. What temperature then?**.
   e. 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

```
- Ok, turning {OnOff} the {SubjectDevice}
- Done, turned {OnOff} the {SubjectDevice}
- 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	`SubjectContext`
Required	Unselected
Type	**String**
Default value	`all`
Configuration	**Accept predefined input values from internal catalog**
Predefined input values	`room`, `bathroom`, `all`

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**.

> **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

11/23/2021 • 10 minutes to read • Edit Online

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 subscription key for Speech Services. Get one for free or create it on the Azure portal
- 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**

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

To create a Visual Studio project for Universal Windows Platform (UWP) development, you need to set up Visual Studio development options, create the project, select the target architecture, set up audio capture, and install the Speech SDK.

Set up Visual Studio development options

To start, make sure you're set up correctly in Visual Studio for UWP development:

1. Open Visual Studio 2019 to display the **Start** window.

2. Select **Continue without code** to go to the Visual Studio IDE.

3. From the Visual Studio menu bar, select **Tools** > **Get Tools and Features** to open Visual Studio Installer and view the **Modifying** dialog box.

4. In the **Workloads** tab, under **Windows**, find the **Universal Windows Platform development** workload. If the check box next to that workload is already selected, close the **Modifying** dialog box, and go to step 6.

5. Select the **Universal Windows Platform development** check box, select **Modify**, and then in the **Before we get started** dialog box, select **Continue** to install the UWP development workload. Installation of the new feature may take a while.

6. Close Visual Studio Installer.

Create the project and select the target architecture

Next, create your project:

1. In the Visual Studio menu bar, choose **File** > **New** > **Project** to display the **Create a new project** window.

2. Find and select **Blank App (Universal Windows)**. Make sure that you select the C# version of this project type (as opposed to Visual Basic).

3. Select **Next** to display the **Configure your new project** screen.

4. In **Project name**, enter `helloworld`.

5. In **Location**, navigate to and select or create the folder to save your project in.

6. Select **Create** to go to the **New Universal Windows Platform Project** window.

7. In **Minimum version** (the second drop-down box), choose **Windows 10 Fall Creators Update (10.0; Build 16299)**, which is the minimum requirement for the Speech SDK.

8. In **Target version** (the first drop-down box), choose a value identical to or later than the value in **Minimum version**.

9. Select **OK**. You're returned to the Visual Studio IDE, with the new project created and visible in the **Solution Explorer** pane.

Now select your target platform architecture. In the Visual Studio toolbar, find the **Solution Platforms** drop-down box. (If you don't see it, choose **View** > **Toolbars** > **Standard** to display the toolbar containing **Solution Platforms**.) If you're running 64-bit Windows, choose **x64** in the drop-down box. 64-bit Windows can also run 32-bit applications, so you can choose **x86** if you prefer.

> **NOTE**
>
> The Speech SDK supports all Intel-compatible processors, but **only x64** versions of ARM processors.

Set up audio capture

Allow the project to capture audio input:

1. In **Solution Explorer**, double-click **Package.appxmanifest** to open the package application manifest.

2. Select the **Capabilities** tab.

3. Select the box for the **Microphone** capability.

4. From the menu bar, choose **File** > **Save Package.appxmanifest** to save your changes.

Install the Speech SDK

Finally, install the Speech SDK NuGet package, and reference the Speech SDK in your project:

1. In **Solution Explorer**, right-click your solution, and choose **Manage NuGet Packages for Solution** to go to the **NuGet - Solution** window.

2. Select **Browse**.

3. In **Package source**, choose **nuget.org**.

4. In the **Search** box, enter `Microsoft.CognitiveServices.Speech`, and then choose that package after it appears in the search results.

5. In the package status pane next to the search results, select your **helloworld** project.

6. Select **Install**.

7. In the **Preview Changes** dialog box, select **OK**.

8. In the **License Acceptance** dialog box, view the license, and then select **I Accept**. The package installation begins, and when installation is complete, the **Output** pane displays a message similar to the following text: `Successfully installed 'Microsoft.CognitiveServices.Speech 1.15.0' to helloworld`.

Step 3: Add sample code

In this step, we add the XAML code that defines the user interface of the application, and add the C# code-behind implementation.

XAML code

Create the application's user interface by adding the XAML code.

1. In **Solution Explorer**, open `MainPage.xaml`

2. In the designer's XAML view, replace the entire contents with the following code snippet:

```
<Page
    x:Class="helloworld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:helloworld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid>
        <StackPanel Orientation="Vertical" HorizontalAlignment="Center"
                    Margin="20,50,0,0" VerticalAlignment="Center" Width="800">
            <Button x:Name="EnableMicrophoneButton" Content="Enable Microphone"
                    Margin="0,10,10,0" Click="EnableMicrophone_ButtonClicked"
                    Height="35"/>
            <Button x:Name="ListenButton" Content="Talk"
                    Margin="0,10,10,0" Click="ListenButton_ButtonClicked"
                    Height="35"/>
            <StackPanel x:Name="StatusPanel" Orientation="Vertical"
                        RelativePanel.AlignBottomWithPanel="True"
                        RelativePanel.AlignRightWithPanel="True"
                        RelativePanel.AlignLeftWithPanel="True">
                <TextBlock x:Name="StatusLabel" Margin="0,10,10,0"
                           TextWrapping="Wrap" Text="Status:" FontSize="20"/>
                <Border x:Name="StatusBorder" Margin="0,0,0,0">
                    <ScrollViewer VerticalScrollMode="Auto"
                                  VerticalScrollBarVisibility="Auto" MaxHeight="200">
                        <!-- Use LiveSetting to enable screen readers to announce
                             the status update. -->
                        <TextBlock
                            x:Name="StatusBlock" FontWeight="Bold"
                            AutomationProperties.LiveSetting="Assertive"
                            MaxWidth="{Binding ElementName=Splitter, Path=ActualWidth}"
                            Margin="10,10,10,20" TextWrapping="Wrap"  />
                    </ScrollViewer>
                </Border>
            </StackPanel>
        </StackPanel>
        <MediaElement x:Name="mediaElement"/>
    </Grid>
</Page>
```

The Design view is updated to show the application's user interface.

C# code-behind source

Add the code-behind source so that the application works as expected. The code-behind source includes:

- Required `using` statements for the `Speech` and `Speech.Dialog` namespaces
- A simple implementation to ensure microphone access, wired to a button handler
- Basic UI helpers to present messages and errors in the application
- A landing point for the initialization code path that will be populated later
- A helper to play back text-to-speech (without streaming support)
- An empty button handler to start listening that will be populated later

Add the code-behind source as follows:

1. In **Solution Explorer**, open the code-behind source file `MainPage.xaml.cs` (grouped under `MainPage.xaml`)

2. Replace the file's contents with the following code:

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Dialog;
using System;
using System.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.FromElement(StatusBlock);
            if (peer != null)
            {
                peer.RaiseAutomationEvent(
                    Windows.UI.Xaml.Automation.Peers.AutomationEvents.LiveRegionChanged);
            }
        }

        // Waits for and accumulates all audio associated with a given
        // PullAudioOutputStream and then plays it to the MediaElement. Long spoken
        // audio will create extra latency and a streaming playback solution
        // (that plays audio while it continues to be received) should be used --
        // see the samples for examples of this.
        private void SynchronouslyPlayActivityAudio(
            PullAudioOutputStream activityAudio)
        {
            var playbackStreamWithHeader = new MemoryStream();
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("RIFF"), 0, 4); // ChunkID
            playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); //
ChunkSize: max
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("WAVE"), 0, 4); // Format
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("fmt "), 0, 4); // Subchunk1ID
            playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 4); // Subchunk1Size: PCM
            playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // AudioFormat: PCM
            playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // NumChannels: mono
            playbackStreamWithHeader.Write(BitConverter.GetBytes(16000), 0, 4); // SampleRate: 16kHz
            playbackStreamWithHeader.Write(BitConverter.GetBytes(32000), 0, 4); // ByteRate
            playbackStreamWithHeader.Write(BitConverter.GetBytes(2), 0, 2); // BlockAlign
            playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 2); // BitsPerSample: 16-bit
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("data"), 0, 4); // Subchunk2ID
            playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); //
Subchunk2Size

            byte[] pullBuffer = new byte[2056];

            uint lastRead = 0;
            do
```

```
            {
                lastRead = activityAudio.Read(pullBuffer);
                playbackStreamWithHeader.Write(pullBuffer, 0, (int)lastRead);
            }
            while (lastRead == pullBuffer.Length);

            var task = Dispatcher.RunAsync(
                Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
            {
                mediaElement.SetSource(
                    playbackStreamWithHeader.AsRandomAccessStream(), "audio/wav");
                mediaElement.Play();
            });
        }

        private void InitializeDialogServiceConnector()
        {
            // New code will go here
        }

        private async void ListenButton_ButtonClicked(
            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"

1. Right-client your solution.
2. Choose **Manage NuGet Packages for Solution**, Select **Updates**
3. 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`

```
// This code creates the `DialogServiceConnector` with your subscription information.
// create a DialogServiceConfig by providing a Custom Commands application id and Cognitive Services
subscription key
// the RecoLanguage property is optional (default en-US); note that only en-US is supported in
Preview
const string speechCommandsApplicationId = "YourApplicationId"; // Your application id
const string speechSubscriptionKey = "YourSpeechSubscriptionKey"; // Your subscription key
const string region = "YourServiceRegion"; // The subscription service region.

var speechCommandsConfig = CustomCommandsConfig.FromSubscription(speechCommandsApplicationId,
speechSubscriptionKey, region);
speechCommandsConfig.SetProperty(PropertyId.SpeechServiceConnection_RecoLanguage, "en-us");
connector = new DialogServiceConnector(speechCommandsConfig);
```

4. Replace the strings `YourApplicationId`, `YourSpeechSubscriptionKey`, and `YourServiceRegion` with your own values for your app, speech subscription, and region

5. Append the following code snippet to the end of the method body of `InitializeDialogServiceConnector`

```
//
// This code sets up handlers for events relied on by `DialogServiceConnector` to communicate its
activities,
// speech recognition results, and other information.
//
// ActivityReceived is the main way your client will receive messages, audio, and events
connector.ActivityReceived += (sender, activityReceivedEventArgs) =>
{
    NotifyUser(
        $"Activity received, hasAudio={activityReceivedEventArgs.HasAudio} activity=
{activityReceivedEventArgs.Activity}");

    if (activityReceivedEventArgs.HasAudio)
    {
        SynchronouslyPlayActivityAudio(activityReceivedEventArgs.Audio);
    }
};

// Canceled will be signaled when a turn is aborted or experiences an error condition
connector.Canceled += (sender, canceledEventArgs) =>
{
    NotifyUser($"Canceled, reason={canceledEventArgs.Reason}");
    if (canceledEventArgs.Reason == CancellationReason.Error)
    {
        NotifyUser(
            $"Error: code={canceledEventArgs.ErrorCode}, details={canceledEventArgs.ErrorDetails}");
    }
};

// Recognizing (not 'Recognized') will provide the intermediate recognized text
// while an audio stream is being processed
connector.Recognizing += (sender, recognitionEventArgs) =>
{
    NotifyUser($"Recognizing! in-progress text={recognitionEventArgs.Result.Text}");
};

// Recognized (not 'Recognizing') will provide the final recognized text
// once audio capture is completed
connector.Recognized += (sender, recognitionEventArgs) =>
{
    NotifyUser($"Final speech-to-text result: '{recognitionEventArgs.Result.Text}'");
};

// SessionStarted will notify when audio begins flowing to the service for a turn
connector.SessionStarted += (sender, sessionEventArgs) =>
{
    NotifyUser($"Now Listening! Session started, id={sessionEventArgs.SessionId}");
};

// SessionStopped will notify when a turn is complete and
// it's safe to begin listening again
connector.SessionStopped += (sender, sessionEventArgs) =>
{
    NotifyUser($"Listening complete. Session ended, id={sessionEventArgs.SessionId}");
};
```

6. Add the following code snippet to the body of the `ListenButton_ButtonClicked` method in the `MainPage` class

```
// This code sets up `DialogServiceConnector` to listen, since you already established the
configuration and
// registered the event handlers.
if (connector == null)
{
    InitializeDialogServiceConnector();
    // Optional step to speed up first interaction: if not called,
    // connection happens automatically on first use
    var connectTask = connector.ConnectAsync();
}

try
{
    // Start sending audio
    await connector.ListenOnceAsync();
}
catch (Exception ex)
{
    NotifyUser($"Exception: {ex.ToString()}", NotifyType.ErrorMessage);
}
```

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

11/23/2021 • 2 minutes to read • Edit Online

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 subscription key for Speech service: Get one for free or create it on the Azure portal
- 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**

```
{
   "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`.

```
<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-client 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

```
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.

```
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($"Received request to set unsupported device {device} to {state}");
                    break;
            }
        }
        else {
            NotifyUser($"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"

Enable Microphone

Talk

Status:

Activity received, hasAudio=False activity={"channelData":{"dynamicGrammarConfiguration":{"dynamicGrammarInfo":{"groups":[],"referenceGrammars":["luis/622453c1-4b3d-4939-a050-641f54a34166_PRODUCTION"]},"ignoreStaticBotGrammar":true}},"channelId":"directlinespeech","conversation":{"id":"27df821f-bee9-4f59-b82f-529d8e01d4e0","isGroup":false},"from":{"id":"dscprodsu01br01","name":"dscprodsu01br01"},"id":"82dcaab0-d7d4-4433-9fbc-957d6dd0299c","name":"setDynamicGrammar","recipient":{"id":"27df821f-bee9-4f59-b82f-529d8e01d4e0|0000"},"replyToId":"ECF1162B-F3BB-47C4-8841-8E79915721D5","serviceUrl":"urn:botframework:websocket:directlinespeech","timestamp":"2020-05-01T06:10:24.163434Z","type":"event"}
Listening complete. Session ended, id=ECF1162B-F3BB-47C4-8841-8E79915721D5

TV on

Fan Off

Next steps

How to: set up web endpoints

Set up web endpoints

11/23/2021 • 7 minutes to read • Edit Online

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 subscription key for Speech service: Get one for free or create it on the Azure portal
- 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.

```
{
  "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.

ATTRIBUTE	EXPLANATION
currentCommand	The command that's currently active in the conversation.
name	The name of the command. The `parameters` attribute is a map with the current values of the parameters.
currentGlobalParameters	A map like `parameters`, 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.

```
{
  "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.

```
{
  "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:

```
{
  "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	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.

SETTING	SUGGESTED VALUE	DESCRIPTION
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:

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.
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.

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**

```
{
   "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.

```
<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.

```
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

Export Custom Commands application as a remote skill

11/23/2021 • 3 minutes to read • Edit Online

In this article, you will learn how to export a Custom Commands application as a remote skill.

Prerequisites

- Understanding of Bot Framework Skill
- Understanding of Skill Manifest
- How to invoke a skill from a Bot Framework Bot
- An exisiting Custom Commands application. In case you don't have any Custom Commands application, try out with - Quickstart: Create a voice assistant using Custom Commands

Custom Commands as remote skills

- Bot Framework Skills are re-usable conversational skill building-blocks covering conversational use-cases enabling you to add extensive functionality to a Bot within minutes. To read more on this, go to Bot Framework Skill.
- A Custom Commands application can be exported as a skill. This skill can then be invoked over the remote skills protocol from a Bot Framework bot.

Configure an application to be exposed as a remote skill

Application level settings

1. In the left panel, select **Settings** > **Remote skills**.
2. Set **Remote skills enabled** toggle to on.

Authentication to skills

1. If you want to enable authentication, add Microsoft Application Ids of the Bot Framework Bots you want to configure to call the custom commands application.

2. If you have at least one entry added to the list, authentication will be enabled on the application, and only the allowed bots will be able to call the application.

> **TIP**
> To disable authentication, delete all the Microsoft Application Ids from the allowed list.

Enable/disable commands to be exposed as skills

You have the option to choose which commands you want to export over Remote Skills.

1. To expose a command over skills, select **Enable a new command** under the **Enable commands for skills**.
2. From the dropdown, select the command you intend to add.
3. Select **Save**.

Configure triggering utterances for commands

Custom Commands uses the example sentences which are configured for the commands in order to generate the skills triggering utterances. These **triggering utterances** will be used to generate the **dispatcher** section skill manifest.

As an author, you might want to control which of your **example sentences** are used to generate the triggering utterances for skills.

1. By default, all the **Triggering examples** from a command will be included the manifest file.
2. If you want to explicitly eliminate any one example, select **Edit** icon on the command from **Enabled commands for skills** section.

3. Next, on the example sentences you want to omit, **right click** > **Disable Example Sentence**.

4. Select **Save**.

5. You will notice that you can't add a new example in this window. If there's a need to do so, proceed to the exit out of the settings section and select the relevant command from **Commands** accordion. At this point, you can add the new entry in the **Example sentences** section. This change will be automatically reflected in the remote skills settings value for the command.

> **IMPORTANT**
>
> In case your existing example sentences have references to **String** > **Catalog** data-type, those sentences will be automatically omitted from the skills triggering utterances list.

Download skill manifest

1. After, you have **published** your application, you can download the skill manifest file.
2. Use the skill manifest to configure your Bot Framework consumer bot to call in to the Custom Commands skill.

> **IMPORTANT**
>
> You must **publish** your Custom Commands application in order to download the skill manifest.
> Additionally, if you made **any changes** to the application, you need to publish the application again for the latest changes to be reflected in the manifest file.

> **NOTE**
>
> If you face any issues with publishing the application and the error directs to skills triggering utterances, please re-check the configuration for **Enabled commands for skills**. Each of the exposed commands must have at least one valid triggering utterance.

Next steps

Update a command from the client

Update a command from a client app

11/23/2021 • 3 minutes to read • Edit Online

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`):

```
{
  "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
type	The activity is of type `"event"`.
name	The name of the event needs to be `"RemoteUpdate"`.
value	The attribute `"value"` contains the attributes required to update the current command.
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`.

ATTRIBUTE	EXPLANATION
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:

```
{
  "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"`.

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`.

4. Save, train, and test your application.
5. In the testing window, send an activity to update the client context.

```
{
   "type": "event",
   "name": "RemoteUpdate",
   "value": {
     "clientContext": {
       "deviceId": "12345",
       "deviceName": "My device"
     },
     "processTurn": false
   }
}
```

6. Send the text `get device info`.

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

11/23/2021 • 2 minutes to read • Edit Online

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):

```
{
  "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.
name	The name of the command. The `parameters` attribute is a map with the current values of the parameters.
currentGlobalParameters	A map like `parameters`, but used for global parameters.

The output of the Azure function needs to support the following format:

```
{
  "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:

```
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**.

5. Create a new interaction rule called **IncrementRule** and add a **Call web endpoint** action.

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.

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).

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

11/23/2021 • 3 minutes to read • Edit Online

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

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	Subscription key used for both applications
Culture	Culture of the applications (i.e. en-us)

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/export.sh -r <region> -s <subscriptionkey> -c en-us -a <appid> -f apps/myapp.json
```

ARGUMENTS	DESCRIPTION
region	region of the application, i.e. westus2.
subscriptionkey	subscription key of your speech resource.
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.

```
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	Subscription key used for both applications
Culture	Culture of the applications (i.e. 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

11/23/2021 • 2 minutes to read • Edit Online

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 programatically 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

11/23/2021 • 2 minutes to read • Edit Online

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

KEYWORD	VARIATIONS
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

11/23/2021 • 3 minutes to read • Edit Online

This article describes how to debug when you see errors while running Custom Commands application.

Connection failed

If your 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

11/23/2021 • 5 minutes to read • Edit Online

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.

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

11/23/2021 • 6 minutes to read • Edit Online

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 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.

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.

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?

11/23/2021 • 2 minutes to read • Edit Online

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.

Getting started with Direct Line Speech

The first steps for creating a voice assistant using Direct Line Speech are to get a speech subscription key, create a new bot associated with that subscription, and connect the bot to the Direct Line Speech channel.

For a complete, step-by-step guide on creating a simple voice assistant using Direct Line Speech, 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 for getting started quickly.

Reference docs

- Speech SDK
- Azure Bot Service

Next steps

- Get a Speech service subscription key for free
- Get the Speech SDK
- Create and deploy a basic bot
- Get the Virtual Assistant Solution and Enterprise Template

Quickstart: Create a custom voice assistant

11/23/2021 • 27 minutes to read • Edit Online

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.

You can view or download all Speech SDK C# Samples on GitHub.

Prerequisites

Before you get started, make sure to:

- Create an Azure 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:

```
<Page
    x:Class="helloworld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:helloworld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid>
        <StackPanel Orientation="Vertical" HorizontalAlignment="Center"
                    Margin="20,50,0,0" VerticalAlignment="Center" Width="800">
            <Button x:Name="EnableMicrophoneButton" Content="Enable Microphone"
                    Margin="0,0,10,0" Click="EnableMicrophone_ButtonClicked"
                    Height="35"/>
            <Button x:Name="ListenButton" Content="Talk to your bot"
                    Margin="0,10,10,0" Click="ListenButton_ButtonClicked"
                    Height="35"/>
            <StackPanel x:Name="StatusPanel" Orientation="Vertical"
                        RelativePanel.AlignBottomWithPanel="True"
                        RelativePanel.AlignRightWithPanel="True"
                        RelativePanel.AlignLeftWithPanel="True">
                <TextBlock x:Name="StatusLabel" Margin="0,10,10,0"
                           TextWrapping="Wrap" Text="Status:" FontSize="20"/>
                <Border x:Name="StatusBorder" Margin="0,0,0,0">
                    <ScrollViewer VerticalScrollMode="Auto"
                                  VerticalScrollBarVisibility="Auto" MaxHeight="200">
                        <!-- Use LiveSetting to enable screen readers to announce
                             the status update. -->
                        <TextBlock
                            x:Name="StatusBlock" FontWeight="Bold"
                            AutomationProperties.LiveSetting="Assertive"
                            MaxWidth="{Binding ElementName=Splitter, Path=ActualWidth}"
                            Margin="10,10,10,20" TextWrapping="Wrap"  />
                    </ScrollViewer>
                </Border>
            </StackPanel>
        </StackPanel>
        <MediaElement x:Name="mediaElement"/>
    </Grid>
</Page>
```

The Design view is updated to show the application's user interface.

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

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Dialog;
using System;
using System.Diagnostics;
```

```
using System.IO;
using System.Text;
using Windows.Foundation;
using Windows.Storage.Streams;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;

namespace helloworld
{
    public sealed partial class MainPage : Page
    {
        private DialogServiceConnector connector;

        private enum NotifyType
        {
            StatusMessage,
            ErrorMessage
        };

        public MainPage()
        {
            this.InitializeComponent();
        }

        private async void EnableMicrophone_ButtonClicked(
            object sender, RoutedEventArgs e)
        {
            bool isMicAvailable = true;
            try
            {
                var mediaCapture = new Windows.Media.Capture.MediaCapture();
                var settings =
                    new Windows.Media.Capture.MediaCaptureInitializationSettings();
                settings.StreamingCaptureMode =
                    Windows.Media.Capture.StreamingCaptureMode.Audio;
                await mediaCapture.InitializeAsync(settings);
            }
            catch (Exception)
            {
                isMicAvailable = false;
            }
            if (!isMicAvailable)
            {
                await Windows.System.Launcher.LaunchUriAsync(
                    new Uri("ms-settings:privacy-microphone"));
            }
            else
            {
                NotifyUser("Microphone was enabled", NotifyType.StatusMessage);
            }
        }

        private void NotifyUser(
            string strMessage, NotifyType type = NotifyType.StatusMessage)
        {
            // If called from the UI thread, then update immediately.
            // Otherwise, schedule a task on the UI thread to perform the update.
            if (Dispatcher.HasThreadAccess)
            {
                UpdateStatus(strMessage, type);
            }
            else
            {
                var task = Dispatcher.RunAsync(
                    Windows.UI.Core.CoreDispatcherPriority.Normal,
                    () => UpdateStatus(strMessage, type));
            }
        }
```

```
        }

        private void UpdateStatus(string strMessage, NotifyType type)
        {
            switch (type)
            {
                case NotifyType.StatusMessage:
                    StatusBorder.Background = new SolidColorBrush(
                        Windows.UI.Colors.Green);
                    break;
                case NotifyType.ErrorMessage:
                    StatusBorder.Background = new SolidColorBrush(
                        Windows.UI.Colors.Red);
                    break;
            }
            StatusBlock.Text += string.IsNullOrEmpty(StatusBlock.Text)
                ? strMessage : "\n" + strMessage;

            if (!string.IsNullOrEmpty(StatusBlock.Text))
            {
                StatusBorder.Visibility = Visibility.Visible;
                StatusPanel.Visibility = Visibility.Visible;
            }
            else
            {
                StatusBorder.Visibility = Visibility.Collapsed;
                StatusPanel.Visibility = Visibility.Collapsed;
            }
            // Raise an event if necessary to enable a screen reader
            // to announce the status update.
            var peer =
Windows.UI.Xaml.Automation.Peers.FrameworkElementAutomationPeer.FromElement(StatusBlock);
            if (peer != null)
            {
                peer.RaiseAutomationEvent(
                    Windows.UI.Xaml.Automation.Peers.AutomationEvents.LiveRegionChanged);
            }
        }

        // Waits for and accumulates all audio associated with a given
        // PullAudioOutputStream and then plays it to the MediaElement. Long spoken
        // audio will create extra latency and a streaming playback solution
        // (that plays audio while it continues to be received) should be used --
        // see the samples for examples of this.
        private void SynchronouslyPlayActivityAudio(
            PullAudioOutputStream activityAudio)
        {
            var playbackStreamWithHeader = new MemoryStream();
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("RIFF"), 0, 4); // ChunkID
            playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); //
ChunkSize: max
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("WAVE"), 0, 4); // Format
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("fmt "), 0, 4); // Subchunk1ID
            playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 4); // Subchunk1Size: PCM
            playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // AudioFormat: PCM
            playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // NumChannels: mono
            playbackStreamWithHeader.Write(BitConverter.GetBytes(16000), 0, 4); // SampleRate: 16kHz
            playbackStreamWithHeader.Write(BitConverter.GetBytes(32000), 0, 4); // ByteRate
            playbackStreamWithHeader.Write(BitConverter.GetBytes(2), 0, 2); // BlockAlign
            playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 2); // BitsPerSample: 16-bit
            playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("data"), 0, 4); // Subchunk2ID
            playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); //
Subchunk2Size

            byte[] pullBuffer = new byte[2056];

            uint lastRead = 0;
            do
            {
                lastRead = activityAudio.Read(pullBuffer);
```

```
                lastRead = activityAudio.Read(pullBuffer);
                playbackStreamWithHeader.Write(pullBuffer, 0, (int)lastRead);
            }
            while (lastRead == pullBuffer.Length);

            var task = Dispatcher.RunAsync(
                Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
            {
                mediaElement.SetSource(
                    playbackStreamWithHeader.AsRandomAccessStream(), "audio/wav");
                mediaElement.Play();
            });
        }

        private void InitializeDialogServiceConnector()
        {
            // New code will go here
        }

        private async void ListenButton_ButtonClicked(
            object sender, RoutedEventArgs e)
        {
            // New code will go here
        }
    }
}
```

1. Add the following code snippet to the method body of `InitializeDialogServiceConnector`. This code creates the `DialogServiceConnector` with your subscription information.

```
// Create a BotFrameworkConfig by providing a Speech service subscription key
// the 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.

```
// 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.

```
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**.

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

You can view or download all Speech SDK Java Samples on GitHub.

Choose your target environment

- Java Runtime
- Android

Prerequisites

Before you get started, make sure to:

- Create an Azure 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.

Create and configure project

1. Start Eclipse.

2. In the Eclipse Launcher, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.

3. In a moment, the main window of the Eclipse IDE appears. Close the **Welcome** screen if one is present.

4. From the Eclipse menu bar, create a new project by choosing **File** > **New** > **Project**.

5. The **New Project** dialog box appears. Select **Java Project**, and select **Next**.

6. The **New Java Project** wizard starts. In the **Project name** field, enter **quickstart**, and choose **JavaSE-1.8** as the execution environment. Select **Finish**.

7. If the **Open Associated Perspective?** window appears, select **Open Perspective**.

8. In the **Package explorer**, right-click the **quickstart** project. Choose **Configure** > **Convert to Maven Project** from the context menu.

9. The **Create new POM** window appears. In the **Group Id** field, enter *com.microsoft.cognitiveservices.speech.samples*, and in the **Artifact Id** field, enter *quickstart*. Then select **Finish**.

10. Open the *pom.xml* file and edit it.

    - At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK, as shown here:

    ```
    <repositories>
      <repository>
        <id>maven-cognitiveservices-speech</id>
        <name>Microsoft Cognitive Services Speech Maven Repository</name>
        <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
      </repository>
    </repositories>
    ```

    - Also add a `dependencies` element, with the Speech SDK version 1.18.0 as a dependency:

    ```
    <dependencies>
      <dependency>
        <groupId>com.microsoft.cognitiveservices.speech</groupId>
        <artifactId>client-sdk</artifactId>
        <version>1.19.0</version>
      </dependency>
    </dependencies>
    ```

    - Save the changes.

Additionally, to enable logging, update the *pom.xml* file to include the following dependency:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.5</version>
</dependency>
```

Add sample code

1. To add a new empty class to your Java project, select **File** > **New** > **Class**.

2. In the **New Java Class** window, enter *speechsdk.quickstart* in the **Package** field and *Main* in the **Name** field.

3. Open the newly created `Main` class, and replace the contents of the `Main.java` file with the following starting code:

```
package speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import com.microsoft.cognitiveservices.speech.audio.PullAudioOutputStream;
import com.microsoft.cognitiveservices.speech.dialog.BotFrameworkConfig;
import com.microsoft.cognitiveservices.speech.dialog.DialogServiceConnector;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.SourceDataLine;
import java.io.InputStream;

public class Main {
    final Logger log = LoggerFactory.getLogger(Main.class);

    public static void main(String[] args) {
        // New code will go here
    }

    private void playAudioStream(PullAudioOutputStream audio) {
        ActivityAudioStream stream = new ActivityAudioStream(audio);
        final ActivityAudioStream.ActivityAudioFormat audioFormat = stream.getActivityAudioFormat();
        final AudioFormat format = new AudioFormat(
                AudioFormat.Encoding.PCM_SIGNED,
                audioFormat.getSamplesPerSecond(),
                audioFormat.getBitsPerSample(),
                audioFormat.getChannels(),
                audioFormat.getFrameSize(),
                audioFormat.getSamplesPerSecond(),
                false);
        try {
            int bufferSize = format.getFrameSize();
            final byte[] data = new byte[bufferSize];

            SourceDataLine.Info info = new DataLine.Info(SourceDataLine.class, format);
            SourceDataLine line = (SourceDataLine) AudioSystem.getLine(info);
            line.open(format);

            if (line != null) {
                line.start();
                int nBytesRead = 0;
                while (nBytesRead != -1) {
                    nBytesRead = stream.read(data);
                    if (nBytesRead != -1) {
                        line.write(data, 0, nBytesRead);
                    }
                }
                line.drain();
                line.stop();
                line.close();
            }
            stream.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

4. In the `main` method, you first configure your `DialogServiceConfig` and use it to create a `DialogServiceConnector` instance. This instance connects to the Direct Line Speech channel to interact with your bot. An `AudioConfig` instance is also used to specify the source for audio input. In this example,

the default microphone is used with `AudioConfig.fromDefaultMicrophoneInput()`.

- Replace the string `YourSubscriptionKey` with your subscription key, which you can get from this website.
- Replace the string `YourServiceRegion` with the region associated with your subscription.

> **NOTE**
>
> Please refer to the list of supported regions for voice assistants and ensure your resources are deployed in one of those regions.

```
final String subscriptionKey = "YourSubscriptionKey"; // Your subscription key
final String region = "YourServiceRegion"; // Your speech subscription service region
final BotFrameworkConfig botConfig = BotFrameworkConfig.fromSubscription(subscriptionKey, region);

// Configure audio input from a microphone.
final AudioConfig audioConfig = AudioConfig.fromDefaultMicrophoneInput();

// Create a DialogServiceConnector instance.
final DialogServiceConnector connector = new DialogServiceConnector(botConfig, audioConfig);
```

5. The connector `DialogServiceConnector` relies on several events to communicate its bot activities, speech recognition results, and other information. Add these event listeners next.

```
// Recognizing will provide the intermediate recognized text while an audio stream is being
processed.
connector.recognizing.addEventListener((o, speechRecognitionResultEventArgs) -> {
    log.info("Recognizing speech event text: {}",
speechRecognitionResultEventArgs.getResult().getText());
});

// Recognized will provide the final recognized text once audio capture is completed.
connector.recognized.addEventListener((o, speechRecognitionResultEventArgs) -> {
    log.info("Recognized speech event reason text: {}",
speechRecognitionResultEventArgs.getResult().getText());
});

// SessionStarted will notify when audio begins flowing to the service for a turn.
connector.sessionStarted.addEventListener((o, sessionEventArgs) -> {
    log.info("Session Started event id: {} ", sessionEventArgs.getSessionId());
});

// SessionStopped will notify when a turn is complete and it's safe to begin listening again.
connector.sessionStopped.addEventListener((o, sessionEventArgs) -> {
    log.info("Session stopped event id: {}", sessionEventArgs.getSessionId());
});

// Canceled will be signaled when a turn is aborted or experiences an error condition.
connector.canceled.addEventListener((o, canceledEventArgs) -> {
    log.info("Canceled event details: {}", canceledEventArgs.getErrorDetails());
    connector.disconnectAsync();
});

// ActivityReceived is the main way your bot will communicate with the client and uses Bot Framework
activities.
connector.activityReceived.addEventListener((o, activityEventArgs) -> {
    final String act = activityEventArgs.getActivity().serialize();
        log.info("Received activity {} audio", activityEventArgs.hasAudio() ? "with" : "without");
        if (activityEventArgs.hasAudio()) {
            playAudioStream(activityEventArgs.getAudio());
        }
    });
```

6. Connect `DialogServiceConnector` to Direct Line Speech by invoking the `connectAsync()` method. To test your bot, you can invoke the `listenOnceAsync` method to send audio input from your microphone. Additionally, you can also use the `sendActivityAsync` method to send a custom activity as a serialized string. These custom activities can provide additional data that your bot uses in the conversation.

```
connector.connectAsync();
// Start listening.
System.out.println("Say something ...");
connector.listenOnceAsync();

// connector.sendActivityAsync(...)
```

7. Save changes to the `Main` file.

8. To support response playback, add an additional class that transforms the PullAudioOutputStream object returned from the getAudio() API to a Java InputStream for ease of handling. This `ActivityAudioStream` is a specialized class that handles audio response from the Direct Line Speech channel. It provides accessors to fetch audio format information that's required for handling playback. For that, select **File** > **New** > **Class**.

9. In the **New Java Class** window, enter *speechsdk.quickstart* in the **Package** field and *ActivityAudioStream* in the **Name** field.

10. Open the newly created `ActivityAudioStream` class, and replace it with the following code:

```
package com.speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.audio.PullAudioOutputStream;

import java.io.IOException;
import java.io.InputStream;

 public final class ActivityAudioStream extends InputStream {
     /**
      * The number of samples played per second (16 kHz).
      */
     public static final long SAMPLE_RATE = 16000;
     /**
      * The number of bits in each sample of a sound that has this format (16 bits).
      */
     public static final int BITS_PER_SECOND = 16;
     /**
      * The number of audio channels in this format (1 for mono).
      */
     public static final int CHANNELS = 1;
     /**
      * The number of bytes in each frame of a sound that has this format (2).
      */
     public static final int FRAME_SIZE = 2;

     /**
      * Reads up to a specified maximum number of bytes of data from the audio
      * stream, putting them into the given byte array.
      *
      * @param b   the buffer into which the data is read
      * @param off the offset, from the beginning of array <code>b</code>, at which
      *            the data will be written
      * @param len the maximum number of bytes to read
      * @return the total number of bytes read into the buffer, or -1 if there
      * is no more data because the end of the stream has been reached
      */
     @Override
     public int read(byte[] b, int off, int len) {
         byte[] tempBuffer = new byte[len];
         int n = (int) this.pullStreamImpl.read(tempBuffer);
         for (int i = 0; i < n; i++) {
             if (off + i > b.length) {
                 throw new ArrayIndexOutOfBoundsException(b.length);
             }
             b[off + i] = tempBuffer[i];
         }
         if (n == 0) {
             return -1;
         }
         return n;
     }

     /**
      * Reads the next byte of data from the activity audio stream if available.
      *
      * @return the next byte of data, or -1 if the end of the stream is reached
      * @see #read(byte[], int, int)
      * @see #read(byte[])
      * @see #available
      * <p>
      */
     @Override
     public int read() {
         byte[] data = new byte[1];
         int temp = read(data);
         if (temp <= 0) {
```

```
        if (temp < 0) {
            // we have a weird situation if read(byte[]) returns 0!
            return -1;
        }
        return data[0] & 0xFF;
    }

    /**
     * Reads up to a specified maximum number of bytes of data from the activity audio stream,
     * putting them into the given byte array.
     *
     * @param b the buffer into which the data is read
     * @return the total number of bytes read into the buffer, or -1 if there
     * is no more data because the end of the stream has been reached
     */
    @Override
    public int read(byte[] b) {
        int n = (int) pullStreamImpl.read(b);
        if (n == 0) {
            return -1;
        }
        return n;
    }

    /**
     * Skips over and discards a specified number of bytes from this
     * audio input stream.
     *
     * @param n the requested number of bytes to be skipped
     * @return the actual number of bytes skipped
     * @throws IOException if an input or output error occurs
     * @see #read
     * @see #available
     */
    @Override
    public long skip(long n) {
        if (n <= 0) {
            return 0;
        }
        if (n <= Integer.MAX_VALUE) {
            byte[] tempBuffer = new byte[(int) n];
            return read(tempBuffer);
        }
        long count = 0;
        for (long i = n; i > 0; i -= Integer.MAX_VALUE) {
            int size = (int) Math.min(Integer.MAX_VALUE, i);
            byte[] tempBuffer = new byte[size];
            count += read(tempBuffer);
        }
        return count;
    }

    /**
     * Closes this audio input stream and releases any system resources associated
     * with the stream.
     */
    @Override
    public void close() {
        this.pullStreamImpl.close();
    }

    /**
     * Fetch the audio format for the ActivityAudioStream. The ActivityAudioFormat defines the
sample rate, bits per sample, and the # channels.
     *
     * @return instance of the ActivityAudioFormat associated with the stream
     */
    public ActivityAudioStream.ActivityAudioFormat getActivityAudioFormat() {
        return activityAudioFormat;
    }
```

```
    }

    /**
     * Returns the maximum number of bytes that can be read (or skipped over) from this
     * audio input stream without blocking.
     *
     * @return the number of bytes that can be read from this audio input stream without blocking.
     * As this implementation does not buffer, this will be defaulted to 0
     */
    @Override
    public int available() {
        return 0;
    }

    public ActivityAudioStream(final PullAudioOutputStream stream) {
        pullStreamImpl = stream;
        this.activityAudioFormat = new ActivityAudioStream.ActivityAudioFormat(SAMPLE_RATE,
BITS_PER_SECOND, CHANNELS, FRAME_SIZE, AudioEncoding.PCM_SIGNED);
    }

    private PullAudioOutputStream pullStreamImpl;

    private ActivityAudioFormat activityAudioFormat;

    /**
     * ActivityAudioFormat is an internal format which contains metadata regarding the type of
arrangement of
     * audio bits in this activity audio stream.
     */
    static class ActivityAudioFormat {

        private long samplesPerSecond;
        private int bitsPerSample;
        private int channels;
        private int frameSize;
        private AudioEncoding encoding;

        public ActivityAudioFormat(long samplesPerSecond, int bitsPerSample, int channels, int
frameSize, AudioEncoding encoding) {
            this.samplesPerSecond = samplesPerSecond;
            this.bitsPerSample = bitsPerSample;
            this.channels = channels;
            this.encoding = encoding;
            this.frameSize = frameSize;
        }

        /**
         * Fetch the number of samples played per second for the associated audio stream format.
         *
         * @return the number of samples played per second
         */
        public long getSamplesPerSecond() {
            return samplesPerSecond;
        }

        /**
         * Fetch the number of bits in each sample of a sound that has this audio stream format.
         *
         * @return the number of bits per sample
         */
        public int getBitsPerSample() {
            return bitsPerSample;
        }

        /**
         * Fetch the number of audio channels used by this audio stream format.
         *
         * @return the number of channels
         */
```

```
        public int getChannels() {
            return channels;
        }

        /**
         * Fetch the default number of bytes in a frame required by this audio stream format.
         *
         * @return the number of bytes
         */
        public int getFrameSize() {
            return frameSize;
        }

        /**
         * Fetch the audio encoding type associated with this audio stream format.
         *
         * @return the encoding associated
         */
        public AudioEncoding getEncoding() {
            return encoding;
        }
    }

    /**
     * Enum defining the types of audio encoding supported by this stream.
     */
    public enum AudioEncoding {
        PCM_SIGNED("PCM_SIGNED");

        String value;

        AudioEncoding(String value) {
            this.value = value;
        }
    }
}
```

11. Save changes to the `ActivityAudioStream` file.

Build and run the app

Select F11, or select **Run** > **Debug**. The console displays the message "Say something." At this point, speak an English phrase or sentence that your bot can understand. Your speech is transmitted to your bot through the Direct Line Speech channel where it's recognized and processed by your bot. The response is returned as an activity. If your bot returns speech as a response, the audio is played back by using the `AudioPlayer` class.

Next steps

Explore Java samples on GitHub

You can view or download all Speech SDK Go Samples on GitHub.

Prerequisites

Before you get started:

- Create an Azure Speech Resource
- Setup 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.

Setup your environment

Update the go.mod file with the latest SDK version by adding this line

```
require (
    github.com/Microsoft/cognitive-services-speech-sdk-go v1.15.0
)
```

Start with some boilerplate code

Replace the contents of your source file (e.g. `quickstart.go`) with the below, which includes:

- "main" package definition
- importing the necessary modules from the Speech SDK
- variables for storing the bot information that will be replaced later in this quickstart
- a simple implementation using the microphone for audio input
- event handlers for various events that take place during a speech interaction

```
package main

import (
    "fmt"
    "time"

    "github.com/Microsoft/cognitive-services-speech-sdk-go/audio"
    "github.com/Microsoft/cognitive-services-speech-sdk-go/dialog"
    "github.com/Microsoft/cognitive-services-speech-sdk-go/speech"
)

func main() {
    subscription :=  "YOUR_SUBSCRIPTION_KEY"
    region := "YOUR_BOT_REGION"

    audioConfig, err := audio.NewAudioConfigFromDefaultMicrophoneInput()
    if err != nil {
        fmt.Println("Got an error: ", err)
        return
    }
    defer audioConfig.Close()
    config, err := dialog.NewBotFrameworkConfigFromSubscription(subscription, region)
    if err != nil {
        fmt.Println("Got an error: ", err)
        return
    }
    defer config.Close()
    connector, err := dialog.NewDialogServiceConnectorFromConfig(config, audioConfig)
    if err != nil {
        fmt.Println("Got an error: ", err)
        return
    }
    defer connector.Close()
    activityReceivedHandler := func(event dialog.ActivityReceivedEventArgs) {
        defer event.Close()
        fmt.Println("Received an activity.")
    }
    connector.ActivityReceived(activityReceivedHandler)
    recognizedHandle := func(event speech.SpeechRecognitionEventArgs) {
        defer event.Close()
        fmt.Println("Recognized ", event.Result.Text)
    }
    connector.Recognized(recognizedHandle)
    recognizingHandler := func(event speech.SpeechRecognitionEventArgs) {
        defer event.Close()
        fmt.Println("Recognizing ", event.Result.Text)
    }
    connector.Recognizing(recognizingHandler)
    connector.ListenOnceAsync()
    <-time.After(10 * time.Second)
}
```

Replace the `YOUR_SUBSCRIPTION_KEY` and `YOUR_BOT_REGION` values with actual values from the Speech resource.

- Navigate to the Azure portal, and open your Speech resource
- Under **Keys and Endpoint** on the left, there are two available subscription keys
  - Use either one as the `YOUR_SUBSCRIPTION_KEY` value replacement
- Under the **Overview** on the left, note the region and map it to the region identifier
  - Use the Region identifier as the `YOUR_BOT_REGION` value replacement, for example: `"westus"` for **West US**

> **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.

Code explanation

The Speech subscription key and region are required to create a speech configuration object. The configuration object is needed to instantiate a speech recognizer object.

The recognizer instance exposes multiple ways to recognize speech. In this example, speech is continuously recognized. This functionality lets the Speech service know that you're sending many phrases for recognition, and when the program terminates to stop recognizing speech. As results are yielded, the code will write them to the console.

Build and run

You're now set up to build your project and test your custom voice assistant using the Speech service.

1. Build your project, e.g. **"go build"**
2. Run the module and speak a phrase or sentence into your device's microphone. Your speech is transmitted to the Direct Line Speech channel and transcribed to text, which appears as output.

> **NOTE**
> The Speech SDK will default to recognizing using en-us for the language, see Specify source language for speech to text for information on choosing the source language.

Next steps

Explore Go samples on GitHub

Additional language and platform support

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

LANGUAGE	CODE SAMPLES
C#	.NET Framework, .NET Core, UWP, Unity, Xamarin
C++	Windows, Linux, macOS
Java	Android, JRE

LANGUAGE	CODE SAMPLES
JavaScript	Browser, Node.js
Objective-C	iOS, macOS
Python	Windows, Linux, macOS
Swift	iOS, macOS

Tutorial: Voice-enable your bot by using the Speech SDK

11/23/2021 • 21 minutes to read • Edit Online

You can use the Speech service in Azure Cognitive Services to voice-enable a chat bot.

In this tutorial, you'll use the Microsoft Bot Framework to create a bot that repeats what you say to it. 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.

This tutorial is designed for developers who are new to Azure, Bot Framework bots, Direct Line Speech, or the Speech SDK, and want to quickly build a working system with limited coding. You don't need experience or familiarity with these services.

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 though 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

If you want to use a different region for this tutorial, these factors might limit your choices:

- 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 and standard voices are available 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, let's build a bot. We're going to start with the echo bot sample, which (as the name implies) echoes the text that you've entered as its response. Don't worry, the sample code is ready for you to use without any changes. It's configured to work with the Direct Line Speech channel, which 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:

```
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:

```
samples\csharp_dotnetcore\02.echo-bot\EchoBot.sln
```

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
   ```

   Then select **Connect**.
4. 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:

   ```
   Publish Succeeded.
   Web App was published successfully https://EchoBot20190805125647.azurewebsites.net/
   ```

   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
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-####** 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 are not 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:

```
{
  "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 subscription key (or authorization token) and region name	On the **Settings** page of the app, make sure that you entered the subscription 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 subscription key does not support neural voices. See Neural 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:


```
{
    "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",
    "inputHint":"acceptingInput",
    "recipient":{
        "id":"129ebffe-772b-47f0-9812-7c5bfd4aca79|0000"
    },
    "replyToId":"67c823b4-4c7a-4828-9d6e-0b84fd052869",
    "serviceUrl":"urn:botframework:websocket:directlinespeech",
    "speak":"<speak version='1.0' xmlns='https://www.w3.org/2001/10/synthesis' xml:lang='en-US'><voice
name='Microsoft Server Speech Text to Speech Voice (en-US, AriaRUS)'>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 subscription 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 Microsoft's assistant, you can write an app that will continuously listen for a keyword of your choice. Keep in mind that a keyword can be single word or a 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:
   a. You see a transcription of what you spoke.
   b. 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 Stefan Apollo (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:

   ```
   var replyText = $"Echo: {turnContext.Activity.Text}";
   await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText), cancellationToken);
   ```

   Replace them with this code:

   ```
   var replyText = $"Echo: {turnContext.Activity.Text}";
   var replySpeak = @"<speak version='1.0' xmlns='https://www.w3.org/2001/10/synthesis' xml:lang='de-
   DE'>
                   <voice name='Microsoft Server Speech Text to Speech Voice (de-DE, Stefan, Apollo)'>"
   +
                   $"{replyText}" + "</voice></speak>";
   await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replySpeak), cancellationToken);
   ```

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**.

Next steps

Build your own client app by using the Speech SDK

See also

- 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.

Voice Assistants on Windows

11/23/2021 • 3 minutes to read • Edit Online

On Windows 10 version 2004 and up, 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:** Our design guidelines lay out the key work required to provide the best possible experiences for voice activation on Windows 10.
- **Visit the Getting Started page:** Start here for the steps to begin implementing voice assistants on Windows, from setting your development environment through an introduction to implementation guide.
- **Try out the sample app**: To experience these capabilities firsthand, visit the UWP Voice Assistant Sample page and follow the steps to get the sample client running.

Getting started with voice assistants on Windows

11/23/2021 • 2 minutes to read • Edit Online

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 code is verifie 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 from

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 Services subscription:** A subscription for Cognitive Speech Services for speech-to-text and text-to-speech conversions. Try Speech Services for free here.
- **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 Services subscription 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

11/23/2021 • 6 minutes to read • Edit Online

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`.

```
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.

```
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`:

```
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.

```
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

11/23/2021 • 2 minutes to read • Edit Online

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`.

To become eligible for inclusion in this list, contact Microsoft at winvoiceassistants@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

11/23/2021 • 8 minutes to read • Edit Online

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.
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	• Process the request in-app • Provides UI indicators for listening states • UI adapts as window sizes change

VOICE ACTIVATION VIEW	ASSISTANT REQUIREMENTS SUMMARY
Above lock	• 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	• 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.

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 multiturn 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.

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.

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

Enable logging in the Speech SDK

11/23/2021 • 4 minutes to read • Edit Online

Logging to file is an optional feature for the Speech SDK. During development logging provides additional information and diagnostics from the Speech SDK's core components. It can be enabled by setting the property `Speech_LogFilename` on a speech configuration object to the location and name of the log file. Logging 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.

> **NOTE**
> Logging is available since Speech SDK version 1.4.0 in all supported Speech SDK programming languages, with the exception of JavaScript.

Sample

The log file name is specified on a configuration object. Taking the `SpeechConfig` as an example and assuming that you have created an instance called `config`:

```
config.SetProperty(PropertyId.Speech_LogFilename, "LogfilePathAndName");
```

```
config.setProperty(PropertyId.Speech_LogFilename, "LogfilePathAndName");
```

```
config->SetProperty(PropertyId::Speech_LogFilename, "LogfilePathAndName");
```

```
config.set_property(speechsdk.PropertyId.Speech_LogFilename, "LogfilePathAndName")
```

```
[config setPropertyTo:@"LogfilePathAndName" byId:SPXSpeechLogFilename];
```

```
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:

```
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:

```
#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.

```
File dir = context.getExternalFilesDir(null);
File logFile = new File(dir, "logfile.txt");
config.setProperty(PropertyId.Speech_LogFilename, logFile.getAbsolutePath());
```

The code above will save a log file to the external storage in the root of an application-specific directory. A user can access the file with the file manager (usually in `Android/data/ApplicationName/logfile.txt`). The file will be deleted when the application is uninstalled.

You also need to request `WRITE_EXTERNAL_STORAGE` permission in the manifest file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="...">
  ...
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  ...
</manifest>
```

Within a Unity Android application, the log file can be created using the application persistent data path folder as follows:

```
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-packagename/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:

```
NSString *filePath = [
  [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject]
    stringByAppendingPathComponent:@"logfile.txt"];
[speechConfig setPropertyTo:filePath byId:SPXSpeechLogFilename];
```

To access a created file, add the below properties to the `Info.plist` property list of the application:

```
<key>UIFileSharingEnabled</key>
<true/>
<key>LSSupportsOpeningDocumentsInPlace</key>
<true/>
```

If you are using Swift on iOS, please use the following code snippet to enable logs:

```
let documentsDirectoryPathString = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask,
true).first!
let documentsDirectoryPath = NSURL(string: documentsDirectoryPathString)!
let logFilePath = documentsDirectoryPath.appendingPathComponent("swift.log")
self.speechConfig!.setPropertyTo(logFilePath!.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:

```
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

How to track Speech SDK memory usage

11/23/2021 • 2 minutes to read • Edit Online

The Speech SDK is based on a native code base that's projected into multiple programming languages through a series of interoperability layers. Each language-specific projection has idiomatically correct features to manage the object lifecycle. Additionally, the Speech SDK includes memory management tooling to track resource usage with object logging and object limits.

How to read object logs

If Speech SDK logging is enabled, tracking tags are emitted to enable historical object observation. These tags include:

- `TrackHandle` or `StopTracking`
- The object type
- The current number of objects that are tracked the type of the object, and the current number being tracked.

Here's a sample log:

```
(284): 8604ms SPX_DBG_TRACE_VERBOSE:  handle_table.h:90 TrackHandle
type=Microsoft::CognitiveServices::Speech::Impl::ISpxRecognitionResult handle=0x0x7f688401e1a0,
ptr=0x0x7f688401e1a0, total=19
```

Set a warning threshold

You have the option to create a warning threshold, and if that threshold is exceeded (assuming logging is enabled), a warning message is logged. The warning message contains a dump of all objects in existence along with their count. This information can be used to better understand issues.

To enable a warning threshold, it must be specified on a `SpeechConfig` object. This object is checked when a new recognizer is created. In the following examples, let's assume that you've created an instance of `SpeechConfig` called `config`:

```
config.SetProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

```
config->SetProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

```
config.setProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

```
speech_config.set_property_by_name("SPEECH-ObjectCountWarnThreshold", "10000")?
```

```
[config setPropertyTo:@"10000" byName:"SPEECH-ObjectCountWarnThreshold"];
```

> **TIP**
> The default value for this property is 10,000.

Set an error threshold

Using the Speech SDK, you can set the maximum number of objects allowed at a given time. If this setting is enabled, when the maximum number is hit, attempts to create new recognizer objects will fail. Existing objects will continue to work.

Here's a sample error:

```
Runtime error: The maximum object count of 500 has been exceeded.
The threshold can be adjusted by setting the SPEECH-ObjectCountErrorThreshold property on the SpeechConfig
object.
See https://docs.microsoft.com/azure/cognitive-services/speech-service/how-to-object-tracking-speech-sdk for
more detailed information.
Handle table dump by object type:
class Microsoft::CognitiveServices::Speech::Impl::ISpxRecognitionResult 0
class Microsoft::CognitiveServices::Speech::Impl::ISpxRecognizer 0
class Microsoft::CognitiveServices::Speech::Impl::ISpxAudioConfig 0
class Microsoft::CognitiveServices::Speech::Impl::ISpxSpeechConfig 0
```

To enable an error threshold, it must be specified on a `SpeechConfig` object. This object is checked when a new recognizer is created. In the following examples, let's assume that you've created an instance of `SpeechConfig` called `config`:

```
config.SetProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

```
config->SetProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

```
config.setProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

```
speech_config.set_property_by_name("SPEECH-ObjectCountErrorThreshold", "10000")?
```

```
[config setPropertyTo:@"10000" byName:"SPEECH-ObjectCountErrorThreshold"];
```

> **TIP**
> The default value for this property is the platform-specific maximum value for a `size_t` data type. A typical recognition will consume between 7 and 10 internal objects.

Next steps

- Learn more about the Speech SDK

Use Speech service through a private endpoint

11/23/2021 • 22 minutes to read • Edit Online

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

Private endpoints require a custom subdomain name for Cognitive Services. Use the following instructions to create one for your Speech resource.

> **WARNING**
>
> 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.
>
> 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
- PowerShell
- Azure CLI

To create a custom domain name by 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	**Microsoft.CognitiveServices/accounts**
Resource	**<your-speech-resource-name>**
Target sub-resource	**account**

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.

```
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.

```
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 v3.0, which is used for Batch transcription and Custom Speech. v3.0 is a successor of v2.0
- 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 v3.0 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 v3.0

Usually, Speech resources use Cognitive Services regional endpoints for communicating with the Speech-to-text REST API v3.0. These resources have the following naming format:

`{region}.api.cognitive.microsoft.com`.

This is a sample request URL:

```
https://westeurope.api.cognitive.microsoft.com/speechtotext/v3.0/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:

```
https://my-private-link-speech.cognitiveservices.azure.com/speechtotext/v3.0/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/v3.0/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 subscription 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:

```
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:

```
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 by using a custom voice model:

```
https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId=974481cc-b769-4b29-af70-2fb557b897c4
```

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`:

```
https://my-private-link-speech.cognitiveservices.azure.com/voice/cognitiveservices/v1?deploymentId=974481cc-b769-4b29-af70-2fb557b897c4
```

The same principle in Example 1 is applied, but the key element this time is `voice`.

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-ConnectionUrl`. 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-ConnectionUrl';
   value='wss://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
   ?traffictype=spx&language=en-US'
   ```

   So the URL that the application used in this example is:

   ```
   wss://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/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:

   ```
   var config = SpeechConfig.FromSubscription(subscriptionKey, 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:

- `subscriptionKey` 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/conversation/cognitiveservices/v1?language=en-US
  ```

Create a `SpeechConfig` instance:

```
var config = SpeechConfig.FromEndpoint(endPoint, subscriptionKey);
```

```
auto config = SpeechConfig::FromEndpoint(endPoint, subscriptionKey);
```

```
SpeechConfig config = SpeechConfig.fromEndpoint(endPoint, subscriptionKey);
```

```
import azure.cognitiveservices.speech as speechsdk
speech_config = speechsdk.SpeechConfig(endpoint=endPoint, subscription=subscriptionKey)
```

```
SPXSpeechConfiguration *speechConfig = [[SPXSpeechConfiguration alloc]
initWithEndpoint:endPoint subscription:subscriptionKey];
```

> **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.

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:

```
C:\>nslookup my-private-link-speech.cognitiveservices.azure.com
Server:  UnKnown
Address:  fe80::1

Non-authoritative answer:
Name:    apimgmthskquihpkz6d90kmhvnabrx3ms3pdubscpdfk1tsx3a.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.regional.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 v3.0

Speech-to-text REST API v3.0 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 subscription 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:

```
var config = SpeechConfig.FromSubscription(subscriptionKey, 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

11/23/2021 • 7 minutes to read • Edit Online

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.

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

Configure OpenSSL for Linux

11/23/2021 • 2 minutes to read • Edit Online

With the Speech SDK version 1.19.0 and higher, OpenSSL is dynamically configured to the host-system version. In previous versions, OpenSSL is statically linked to the core library of the SDK.

To ensure connectivity, verify that OpenSSL certificates have been installed in your system. Run a command:

```
openssl version -d
```

The output on Ubuntu/Debian based systems should be:

```
OPENSSLDIR: "/usr/lib/ssl"
```

Check whether there is `certs` subdirectory under OPENSSLDIR. In the example above, it would be `/usr/lib/ssl/certs`.

- If there is `/usr/lib/ssl/certs` and it contains many individual certificate files (with `.crt` or `.pem` extension), there is no need for further actions.
- If OPENSSLDIR is something else than `/usr/lib/ssl` and/or there is 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

- OPENSSLDIR is `/opt/ssl`. There is `certs` subdirectory with many `.crt` or `.pem` files. Set environment variable `SSL_CERT_DIR` to point at `/opt/ssl/certs` before running a program that uses the Speech SDK. For example:

```
export SSL_CERT_DIR=/opt/ssl/certs
```

- OPENSSLDIR is `/etc/pki/tls` (like on RHEL/CentOS based systems). There is `certs` subdirectory with a certificate bundle file, for example `ca-bundle.crt`. Set environment variable `SSL_CERT_FILE` to point at that file before running a program that uses the Speech SDK. For example:

```
export SSL_CERT_FILE=/etc/pki/tls/certs/ca-bundle.crt
```

Certificate revocation checks

When connecting to the Speech Service, the Speech SDK will verify that the TLS certificate used by the Speech Service has not been revoked. To conduct this check, the Speech SDK will need access to the CRL distribution points for Certificate Authorities used by Azure. A list of possible CRL download locations can be found in this document. If a certificate has been revoked or the CRL cannot be downloaded the Speech SDK will abort the connection and raise the Canceled event.

In the event the network where the Speech SDK is being used from is configured in a manner that does not permit access to the CRL download locations, the CRL check can either be disabled or set to not fail if the CRL

cannot be retrieved. This configuration is done through the configuration object used to create a Recognizer object.

To continue with the connection when a CRL cannot be retrieved set the property OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE.

```
config.SetProperty("OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE", "true");
```

```
config->SetProperty("OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE", "true");
```

```
config.setProperty("OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE", "true");
```

```
speech_config.set_property_by_name("OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE", "true")
```

```
[config setPropertyTo:@"true" byName:"OPENSSL_CONTINUE_ON_CRL_DOWNLOAD_FAILURE"];
```

When set to "true" an attempt will be made to retrieve the CRL and if the retrieval is successful the certificate will be checked for revocation, if the retrieval fails, the connection will be allowed to continue.

To completely disable certificate revocation checks, set the property OPENSSL_DISABLE_CRL_CHECK to "true".

```
config.SetProperty("OPENSSL_DISABLE_CRL_CHECK", "true");
```

```
config->SetProperty("OPENSSL_DISABLE_CRL_CHECK", "true");
```

```
config.setProperty("OPENSSL_DISABLE_CRL_CHECK", "true");
```

```
speech_config.set_property_by_name("OPENSSL_DISABLE_CRL_CHECK", "true")
```

```
[config setPropertyTo:@"true" byName:"OPENSSL_DISABLE_CRL_CHECK"];
```

> **NOTE**
>
> It is also worth noting that some distributions of Linux do not have a TMP or TMPDIR environment variable defined. This will cause the Speech SDK to download the Certificate Revocation List (CRL) every time, rather than caching the CRL to disk for reuse until they expire. To improve initial connection performance you can create an environment variable named TMPDIR and set it to the path of your chosen temporary directory..

Next steps

About the Speech SDK

Configure RHEL/CentOS 7 for Speech SDK

11/23/2021 • 2 minutes to read • Edit Online

To use the Speech SDK for C++ development on Red Hat Enterprise Linux (RHEL) 8 x64 and CentOS 8 x64, update the C++ compiler and the shared C++ runtime library on your system.

1. General setup

First install all general dependencies:

```
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
```

2. C/C++ compiler and runtime libraries

Install the prerequisite packages with this command:

```
sudo yum install -y gmp-devel mpfr-devel libmpc-devel
```

Next update the compiler and runtime libraries:

```
# 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.bz2 -O
tar jxf gcc-7.5.0.tar.bz2
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.

3. Environment settings

Run the following commands to complete the configuration:

```
# 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:$PATH
hash -r # reset cached paths in the current shell session just in case
export LD_LIBRARY_PATH=/path/to/extracted/SpeechSDK-Linux-<version>/lib/x64:$LD_LIBRARY_PATH
```

Next steps

About the Speech SDK

Install and run Docker containers for the Speech service APIs

11/23/2021 • 22 minutes to read • Edit Online

Containers enable you to run *some* of the Speech service APIs in your own environment. Containers are great for specific security and data governance requirements. In this article you'll learn how to download, install, and run a Speech container.

Speech containers enable customers to build a speech application architecture that is optimized for both robust cloud capabilities and edge locality. There are several containers 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 31st 2021. Consider migrating your applications to use the Neural text-to-speech container instead. Follow these steps for more information on updating your application.

CONTAINER	FEATURES	LATEST	RELEASE STATUS
Speech-to-text	Analyzes sentiment and transcribes continuous real-time speech or batch audio recordings with intermediate results.	2.16.0	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.	2.16.0	Generally Available
Text-to-speech	Converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.15.0	Generally Available
Speech Language Identification	Detect the language spoken in audio files.	1.5.0	preview
Neural Text-to-speech	Converts text to natural-sounding speech using deep neural network technology, allowing for more natural synthesized speech.	1.10.0	Generally Available

Prerequisites

> **IMPORTANT**
>
> - To use the speech containers you must submit an online request, and have it approved. See the **Request approval to the run the container** section below for more information.
> - *Generally Available* containers meet Microsoft's stability and support requirements. Containers in *Preview* are still under development.

You must meet the following prerequisites before using Speech service containers. If you don't have an Azure subscription, create a free account before you begin.

- 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.

Gathering required parameters

There are three primary parameters for all Cognitive Services' containers that are required. The end-user license agreement (EULA) must be present with a value of `accept`. Additionally, both an Endpoint URL and API Key are needed.

Endpoint URI `{ENDPOINT_URI}`

The **Endpoint** URI value is available on the Azure portal *Overview* page of the corresponding Cognitive Service resource. Navigate to the *Overview* page, hover over the Endpoint, and a `Copy to clipboard` icon will appear. Copy and use where needed.

Keys `{API_KEY}`

This key is used to start the container, and is available on the Azure portal's Keys page of the corresponding Cognitive Service resource. Navigate to the *Keys* page, and click on the `Copy to clipboard` icon.

> **IMPORTANT**
>
> These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely, for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

Host computer requirements and recommendations

The host is a x64-based computer that runs the Docker container. It can be a computer on your premises or a Docker hosting service in Azure, such as:

- Azure Kubernetes Service.
- Azure Container Instances.
- A Kubernetes cluster deployed to Azure Stack. For more information, see Deploy Kubernetes to Azure Stack.

Container requirements and recommendations

The following table describes the minimum and recommended allocation of resources for each Speech container.

CONTAINER	MINIMUM	RECOMMENDED
Speech-to-text	2 core, 2-GB memory	4 core, 4-GB memory
Custom Speech-to-text	2 core, 2-GB memory	4 core, 4-GB memory
Text-to-speech	1 core, 2-GB memory	2 core, 3-GB memory
Speech Language Identification	1 core, 1-GB memory	1 core, 1-GB memory
Neural Text-to-speech	6 core, 12-GB memory	8 core, 16-GB memory

- Each core must be at least 2.6 gigahertz (GHz) or faster.

Core and memory correspond to the `--cpus` and `--memory` settings, which are used as part of the `docker run` command.

> **NOTE**
> The minimum and recommended are based off of Docker limits, *not* the host machine resources. For example, speech-to-text containers memory map portions of a large language model, and it is *recommended* that the entire file fits in memory, which is an additional 4-6 GB. Also, the first run of either container may take longer, since models are being paged into memory.

Advanced Vector Extension support

The **host** is the computer that runs the docker container. The host *must support* Advanced Vector Extensions (AVX2). You can check for AVX2 support on Linux hosts with the following command:

```
grep -q avx2 /proc/cpuinfo && echo AVX2 supported || echo No AVX2 support detected
```

> **WARNING**
> The host computer is *required* to support AVX2. The container *will not* function correctly without AVX2 support.

Request approval to the 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 will review it and email you with a decision.

> **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 will be able to run the container after downloading 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 has not been approved.

Get the container image with `docker pull`

Container images for Speech are available in the following Container Registry.

- Speech-to-text
- Custom Speech-to-text
- Text-to-speech
- Neural Text-to-speech
- Speech Language Identification

CONTAINER	REPOSITORY
Speech-to-text	`mcr.microsoft.com/azure-cognitive-services/speechservices/speech-to-text:latest`

TIP

You can use the docker images command to list your downloaded container images. For example, the following command lists the ID, repository, and tag of each downloaded container image, formatted as a table:

```
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"

IMAGE ID         REPOSITORY                TAG
<image-id>       <repository-path/name>    <tag-name>
```

Docker pull for the Speech containers

- Speech-to-text
- Custom Speech-to-text
- Text-to-speech
- Neural Text-to-speech
- Speech Language Identification

Docker pull for the Speech-to-text container

Use the docker pull command to download a container image from Microsoft Container registry.

```
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 of the supported locales of the **speech-to-text** container, please see Speech-to-text image tags.

How to use the container

Once the container is on the host computer, use the following process to work with the container.

1. Run the container, with the required billing settings. More examples of the `docker run` command are available.
2. Query the container's prediction endpoint.

Run the container with `docker run`

Use the docker run command to run the container. Refer to gathering required parameters for details on how to get the `{Endpoint_URI}` and `{API_Key}` values. Additional examples of the `docker run` command are also available.

- Speech-to-text
- Custom Speech-to-text
- Text-to-speech
- Neural Text-to-speech
- Speech Language Identification

To run the Standard *Speech-to-text* container, execute the following `docker run` command.

```
docker run --rm -it -p 5000:5000 --memory 4g --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 4 gigabytes (GB) of memory.
- Exposes TCP port 5000 and allocates a pseudo-TTY for the container.
- Automatically removes the container after it exits. The container image is still available on the host computer.

> **NOTE**
> Containers support compressed audio input to Speech SDK 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`.

```
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
)
```

> **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://westus2.api.cognitive.microsoft.com/text/analytics/v3.0/sentiment`
- `https://localhost:5000/text/analytics/v3.0/sentiment`

> **NOTE**
> The Language service `v3.0` API is not 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 will need a Language service API resource endpoint. For example:

- `https://westus2.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 will need a key. If you're running Language service features locally, you may not need to provide this.

The key and endpoint are passed to the Speech container as arguments, as in the following example.

```
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 command above.
- 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:

```
    phrase="the tall man"
    recognizer = speechsdk.SpeechRecognizer(
        speech_config=dict_speech_config,
        audio_config=audio_config)
    phrase_list_grammer = speechsdk.PhraseListGrammar.from_recognizer(recognizer)
    phrase_list_grammer.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.

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	`ws://localhost:5000`	WS
Text-to-speech (including Standard and Neural), Speech Language identification	`http://localhost:5000`	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 using the Speech SDK with containers, you do not need to provide the Azure Speech resource subscription key or an authentication bearer token.

See the examples below.

- C#
- Python

Change from using this Azure-cloud initialization call:

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

To using this call with the container host:

```
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. Please use v1.12.x or earlier if you're using sentiment analysis in the Speech Service Python SDK.

- Simple format
- Detailed 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.

```
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` will return the sentiment result in root layer of the response.

```
{
   "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`.

```
speech_config.set_service_property(
    name='speechcontext-phraseDetection.sentimentanalysis.enabled',
    value='false',
    channel=speechsdk.ServicePropertyChannel.UriQueryParameter
)
```

Text-to-speech (Standard and Neural)

The container provides REST-based endpoint APIs. There are many sample source code projects for platform, framework, and language variations available.

With the Standard or 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 example SSML below:

```
<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 Azure Cognitive Services container running on the HOST together. You also can have multiple containers of the same Cognitive Services container running.

Validate that a container is running

There are several ways to validate that the container is running. Locate the *External IP* address and exposed port of the container in question, and open your favorite web browser. Use the various request URLs below to validate the container is running. The example request URLs listed below are `http://localhost:5000`, but your specific container may vary. Keep in mind that you're to rely on your container's *External IP* address and exposed port.

REQUEST URL	PURPOSE
`http://localhost:5000/`	The container provides a home page.
`http://localhost:5000/ready`	Requested with GET, this 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.
`http://localhost:5000/status`	Also requested with GET, this 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.
`http://localhost:5000/swagger`	The container provides a full set of documentation for the endpoints and a **Try it out** feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.

Stop the container

To shut down the container, in the command-line environment where the container is running, select `Ctrl+C`.

Troubleshooting

When starting or running the container, you may experience issues. Use an output mount and enable logging. Doing so will allow the container to generate log files that are helpful when troubleshooting issues.

> **TIP**
> For more troubleshooting information and guidance, see Cognitive Services containers frequently asked questions (FAQ).

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:

```
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:

```
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, 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 / 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
`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 downloading, installing, and running Speech containers. In summary:

- Speech provides four Linux containers for Docker, encapsulating various capabilities:
  - *Speech-to-text*
  - *Custom Speech-to-text*
  - *Text-to-speech*
  - *Custom Text-to-speech*
  - *Neural Text-to-speech*
  - *Speech Language Identification*
- Container images are downloaded from the container registry in Azure.
- Container images run in Docker.
- Whether using 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 instantiating a container.

> **IMPORTANT**
> Cognitive Services containers are not licensed to run without being connected to Azure for metering. Customers need to enable the containers to communicate billing information with the metering service at all times. Cognitive Services containers do not send customer data (e.g., the image or text that is being analyzed) to Microsoft.

Next steps

- Review configure containers for configuration settings
- Learn how to use Speech service containers with Kubernetes and Helm
- Use more Cognitive Services containers

Configure Speech service containers

11/23/2021 • 10 minutes to read • Edit Online

Speech containers enable customers to build one speech application architecture that is optimized to take advantage of both robust cloud capabilities and edge locality. The five speech containers we support now are, **speech-to-text**, **custom-speech-to-text**, **text-to-speech**, **neural-text-to-speech** and **custom-text-to-speech**.

The **Speech** container runtime environment is configured using the `docker run` command arguments. This container has several required settings, along with a few optional settings. Several examples of the command are available. The container-specific settings are the billing settings.

Configuration settings

The container has the following configuration settings:

REQUIRED	SETTING	PURPOSE
Yes	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	`InstrumentationKey`	String	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. Example: `InstrumentationKey=123456789`

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 gathering 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

REQUIRED	NAME	DATA TYPE	DESCRIPTION
Yes	`Eula`	String	License acceptance Example: `Eula=accept`

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.
`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 `UseTLS` 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, `http://proxy:8888` `<proxy-url>`
HTTP_PROXY_CREDS	string	Any credentials needed to authenticate against the proxy, for example, `username:password`. This value **must be in lower-case**.
`<proxy-user>`	string	The user for the proxy.
`<proxy-password>`	string	The password associated with `<proxy-user>` for the proxy.

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
HTTP_PROXY=<proxy-url> \
HTTP_PROXY_CREDS=<proxy-user>:<proxy-password> \
```

Logging settings

The `Logging` settings manage ASP.NET Core logging support for your container. You can use the same configuration settings and values for your container that you use for an ASP.NET Core application.

The following logging providers are supported by the container:

PROVIDER	PURPOSE
Console	The ASP.NET Core `Console` logging provider. All of the ASP.NET Core configuration settings and default values for this logging provider are supported.
Debug	The ASP.NET Core `Debug` logging provider. All of the ASP.NET Core configuration settings and default values for this logging provider are supported.
Disk	The JSON logging provider. This logging provider writes log data to the output mount.

This container command stores logging information in the JSON format to the output mount:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Disk:Format=json
```

This container command shows debugging information, prefixed with `dbug`, while the container is running:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Console:LogLevel:Default=Debug
```

Disk logging

The `Disk` logging provider supports the following configuration settings:

NAME	DATA TYPE	DESCRIPTION
`Format`	String	The output format for log files. **Note:** This value must be set to `json` to enable the logging provider. If this value is specified without also specifying an output mount while instantiating a container, an error occurs.
`MaxFileSize`	Integer	The maximum size, in megabytes (MB), of a log file. When the size of the current log file meets or exceeds this value, a new log file is started by the logging provider. If -1 is specified, the size of the log file is limited only by the maximum file size, if any, for the output mount. The default value is 1.

For more information about configuring ASP.NET Core logging support, see Settings file configuration.

Mount settings

Use bind mounts to read and write data to and from the container. You can specify an input mount or output mount by specifying the `--mount` option in the docker run command.

The Standard Speech containers don't use input or output mounts to store training or service data. However, custom speech containers rely on volume mounts.

The exact syntax of the host mount location varies depending on the host operating system. Additionally, the host computer's mount location may not be accessible due to a conflict between permissions used by the docker service account and the host mount location permissions.

OPTIONAL	NAME	DATA TYPE	DESCRIPTION
Not allowed	`Input`	String	Standard Speech containers do not use this. Custom speech containers use volume mounts.
Optional	`Output`	String	The target of the output mount. The default value is `/output`. This is the location of the logs. This includes container logs. Example: `--mount type=bind,src=c:\output,target=/outp`

Volume mount settings

The custom speech containers use volume mounts to persist custom models. You can specify a volume mount by adding the `-v` (or `--volume`) option to the docker run command.

Custom models are downloaded the first time that a new model is ingested as part of the custom speech container docker run command. Sequential runs of the same `ModelId` for a custom speech container will use the previously downloaded model. If the volume mount is not provided, custom models cannot be persisted.

The volume mount setting consists of three color `:` separated fields:

1. The first field is the name of the volume on the host machine, for example *C:\input*.
2. The second field is the directory in the container, for example */usr/local/models*.
3. The third field (optional) is a comma-separated list of options, for more information see use volumes.

Volume mount example

```
-v C:\input:/usr/local/models
```

This command mounts the host machine *C:\input* directory to the containers */usr/local/models* directory.

> **IMPORTANT**
> The volume mount settings are only applicable to **Custom Speech-to-text** and **Custom Text-to-speech** containers. The **Speech-to-text**, **Neural Text-to-speech** and **Text-to-speech** containers do not use volume mounts.

Example docker run commands

The following examples use the configuration settings to illustrate how to write and use `docker run` commands. Once running, the container continues to run until you stop it.

- **Line-continuation character**: The Docker commands in the following sections use the back slash, `\`, as a line continuation character. Replace or remove this based on your host operating system's requirements.
- **Argument order**: Do not change the order of the arguments unless you are familiar with Docker containers.

Replace {*argument_name*} with your own values:

PLACEHOLDER	VALUE	FORMAT OR EXAMPLE
{API_KEY}	The endpoint key of the `Speech` resource on the Azure `Speech` Keys page.	`xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
{ENDPOINT_URI}	The billing endpoint value is available on the Azure `Speech` Overview page.	See gathering 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.

- Speech-to-text
- Custom Speech-to-text
- Text-to-speech
- Custom Text-to-speech
- Neural Text-to-speech
- Speech Language Identification

Basic example for Speech-to-text

```
docker run --rm -it -p 5000:5000 --memory 4g --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 run --rm -it -p 5000:5000 --memory 4g --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

Batch processing kit for Speech containers

11/23/2021 • 6 minutes to read • Edit Online

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.
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.

```
docker pull docker.io/batchkit/speech-batch-kit:latest
```

Endpoint configuration

The batch client takes ayamlconfiguration file that specifies the on-prem container endpoints. The following example can be written to `/mnt/my_nfs/config.yaml`, which is used in the examples below.

```
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-lineparameters, 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 run --rm -ti-v /mnt/my_nfs:/my_nfs--entrypoint/bin/bash/mnt/my_nfs:/my_nfsdocker.io/batchkit/speech-batch-kit:latest
```

To run the batch client:

```
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_levelDEBUG -nbest1-mONESHOT-diarization None-languageen-US-strict_config
```

To run the batch client and container in a single command:

```
docker run --rm -ti-v /mnt/my_nfs:/my_nfsdocker.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-file_log_levelDEBUG -nbest1-mONESHOT-diarization None-languageen-US-strict_config
```

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 to 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
- Daemon
- REST

`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

Use Speech service containers with Kubernetes and Helm

11/23/2021 • 12 minutes to read • Edit Online

One option to manage your Speech containers on-premises is to use Kubernetes and Helm. Using Kubernetes and Helm to define the speech-to-text and text-to-speech container images, we'll create a Kubernetes package. This package will be deployed to a Kubernetes cluster on-premises. Finally, we'll explore how to test the deployed services and various configuration options. For more information about running Docker containers without Kubernetes orchestration, see install and run Speech service containers.

Prerequisites

The following prerequisites before using Speech containers on-premises:

REQUIRED	PURPOSE
Azure Account	If you don't have an Azure subscription, create a 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 to install a helm chart (container package definition).
Speech resource	In order to use these containers, you must have: A *Speech* 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://westus.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 `optimizedForAudioFile` 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 `optimizeForTurboMode` 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.

```
helm repo add microsoft https://microsoft.github.io/charts/repo
```

Next, we'll configure our Helm chart values. Copy and paste the following YAML into a file named `config-values.yaml`. For more information on customizing the **Cognitive Services Speech On-Premises Helm Chart**, see customize helm charts. Replace the `# {ENDPOINT_URI}` and `# {API_KEY}` comments with your own values.

```
# These settings are deployment specific and users can provide customizations
# speech-to-text configurations
speechToText:
  enabled: true
  numberOfConcurrentRequest: 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
  numberOfConcurrentRequest: 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.

```
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:

```
NAME:   onprem-speech
LAST DEPLOYED: Tue Jul  2 12:51:42 2019
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME                              READY  STATUS             RESTARTS  AGE
speech-to-text-7664f5f465-87w2d  0/1    Pending            0         0s
speech-to-text-7664f5f465-klbr8  0/1    ContainerCreating  0         0s
text-to-speech-56f8fb685b-4jtzh  0/1    ContainerCreating  0         0s
text-to-speech-56f8fb685b-frwxf  0/1    Pending            0         0s

==> v1/Service
NAME            TYPE           CLUSTER-IP    EXTERNAL-IP  PORT(S)       AGE
speech-to-text  LoadBalancer  10.0.252.106  <pending>    80:31811/TCP  1s
text-to-speech  LoadBalancer  10.0.125.187  <pending>    80:31247/TCP  0s

==> v1beta1/PodDisruptionBudget
NAME                                MIN AVAILABLE  MAX UNAVAILABLE  ALLOWED DISRUPTIONS  AGE
speech-to-text-poddisruptionbudget  N/A            20%              0                    1s
text-to-speech-poddisruptionbudget  N/A            20%              0                    1s

==> v1beta2/Deployment
NAME            READY  UP-TO-DATE  AVAILABLE  AGE
speech-to-text  0/2    2           0          0s
text-to-speech  0/2    2           0          0s

==> v2beta2/HorizontalPodAutoscaler
NAME                       REFERENCE                  TARGETS        MINPODS  MAXPODS  REPLICAS  AGE
speech-to-text-autoscaler  Deployment/speech-to-text  <unknown>/50%  2        10       0         0s
text-to-speech-autoscaler  Deployment/text-to-speech  <unknown>/50%  2        10       0         0s


NOTES:
cognitive-services-speech-onpremise has been installed!
Release is named onprem-speech
```

The Kubernetes deployment can take over several minutes to complete. To confirm that both pods and services are properly deployed and available, execute the following command:

```
kubectl get all
```

You should expect to see something similar to the following output:

```
NAME                                  READY     STATUS    RESTARTS   AGE
pod/speech-to-text-7664f5f465-87w2d   1/1       Running   0          34m
pod/speech-to-text-7664f5f465-klbr8   1/1       Running   0          34m
pod/text-to-speech-56f8fb685b-4jtzh   1/1       Running   0          34m
pod/text-to-speech-56f8fb685b-frwxf   1/1       Running   0          34m

NAME                     TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
service/kubernetes       ClusterIP      10.0.0.1       <none>           443/TCP        3h
service/speech-to-text   LoadBalancer   10.0.252.106   52.162.123.151   80:31811/TCP   34m
service/text-to-speech   LoadBalancer   10.0.125.187   65.52.233.162    80:31247/TCP   34m

NAME                             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/speech-to-text   2         2         2            2           34m
deployment.apps/text-to-speech   2         2         2            2           34m

NAME                                        DESIRED   CURRENT   READY     AGE
replicaset.apps/speech-to-text-7664f5f465   2         2         2         34m
replicaset.apps/text-to-speech-56f8fb685b   2         2         2         34m

NAME                                                                  REFERENCE                   TARGETS
MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/speech-to-text-autoscaler   Deployment/speech-to-text   1%/50%    2
10        2          34m
horizontalpodautoscaler.autoscaling/text-to-speech-autoscaler   Deployment/text-to-speech   0%/50%    2
10        2          34m
```

Verify Helm deployment with Helm tests

The installed Helm charts define *Helm tests*, which serve as a convenience for verification. These tests validate service readiness. To verify both **speech-to-text** and **text-to-speech** services, we'll execute the Helm test command.

```
helm test onprem-speech
```

> **IMPORTANT**
> These tests will fail if the POD status is not `Running` or if the deployment is not listed under the `AVAILABLE` column. Be patient as this can take over ten minutes to complete.

These tests will output various status results:

```
RUNNING: speech-to-text-readiness-test
PASSED: speech-to-text-readiness-test
RUNNING: text-to-speech-readiness-test
PASSED: text-to-speech-readiness-test
```

As an alternative to executing the *helm tests*, you could collect the *External IP* addresses and corresponding ports from the `kubectl get all` command. Using the IP and port, open a web browser and navigate to `http://<external-ip>:<port>:/swagger/index.html` to view the API swagger page(s).

Customize Helm charts

Helm charts are hierarchical. Being hierarchical allows for chart inheritance, it also caters to the concept of specificity, where settings that are more specific override inherited rules.

Speech (umbrella chart)

Values in the top-level "umbrella" chart override the corresponding sub-chart values. Therefore, all on-premises

customized values should be added here.

PARAMETER	DESCRIPTION	DEFAULT
`speechToText.enabled`	Whether the **speech-to-text** service is enabled.	`true`
`speechToText.verification.enabled`	Whether the `helm test` capability for **speech-to-text** service is enabled.	`true`
`speechToText.verification.image.registry`	The docker image repository that `helm test` uses to test **speech-to-text** service. Helm creates separate pod inside the cluster for testing and pulls the *test-use* image from this registry.	`docker.io`
`speechToText.verification.image.repository`	The docker image repository that `helm test` uses to test **speech-to-text** service. Helm test pod uses this repository to pull *test-use* image.	`antsu/on-prem-client`
`speechToText.verification.image.tag`	The docker image tag used with `helm test` for **speech-to-text** service. Helm test pod uses this tag to pull *test-use* image.	`latest`
`speechToText.verification.image.pullByHash`	Whether the *test-use* docker image is pulled by hash. If `true`, `speechToText.verification.image.hash` should be added, with valid image hash value.	`false`

`speechToText.verification.image.arguments`	The arguments used to execute the *test-use* docker image. Helm test pod passes these arguments to the container when running `helm test`.	`"./speech-to-text-client"`
		`"./audio/whatstheweatherlike.wav"`
		`"--expect=What's the weather like"`
		`"--host=$(SPEECH_TO_TEXT_HOST)"`
		`"--port=$(SPEECH_TO_TEXT_PORT)"`

`textToSpeech.enabled`	Whether the **text-to-speech** service is enabled.	`true`
`textToSpeech.verification.enabled`	Whether the `helm test` capability for **speech-to-text** service is enabled.	`true`
`textToSpeech.verification.image.registry`	The docker image repository that `helm test` uses to test **speech-to-text** service. Helm creates separate pod inside the cluster for testing and pulls the *test-use* image from this registry.	`docker.io`
`textToSpeech.verification.image.repository`	The docker image repository that `helm test` uses to test **speech-to-text** service. Helm test pod uses this repository to pull *test-use* image.	`antsu/on-prem-client`

PARAMETER	DESCRIPTION	DEFAULT
`textToSpeech.verification.image.tag`	The docker image tag used with `helm test` for **speech-to-text** service. Helm test pod uses this tag to pull *test-use* image.	`latest`
`textToSpeech.verification.image.pullByHash`	Whether the *test-use* docker image is pulled by hash. If `true`, `textToSpeech.verification.image.hash` should be added, with valid image hash value.	`false`

`textToSpeech.verification.image.arguments`	The arguments to execute with the *test-use* docker image. The helm test pod passes these arguments to container when running `helm test`.	`"./text-to-speech-client"`
		`"--input='What's the weather like'"`
		`"--host=$(TEXT_TO_SPEECH_HOST)"`
		`"--port=$(TEXT_TO_SPEECH_PORT)"`

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
`enabled`	Whether the **speech-to-text** service is enabled.	`false`
`numberOfConcurrentRequest`	The number of concurrent requests for the **speech-to-text** service. This chart automatically calculates CPU and memory resources, based on this value.	`2`
`optimizeForAudioFile`	Whether the service needs to optimize for audio input via audio files. If `true`, this chart will allocate more CPU resource to service.	`false`
`image.registry`	The **speech-to-text** docker image registry.	`containerpreview.azurecr.io`
`image.repository`	The **speech-to-text** docker image repository.	`microsoft/cognitive-services-speech-to-text`
`image.tag`	The **speech-to-text** docker image tag.	`latest`
`image.pullSecrets`	The image secrets for pulling the **speech-to-text** docker image.	
`image.pullByHash`	Whether the docker image is pulled by hash. If `true`, `image.hash` is required.	`false`

PARAMETER	DESCRIPTION	DEFAULT
`image.hash`	The **speech-to-text** docker image hash. Only used when `image.pullByHash: true`.	
`image.args.eula` (required)	Indicates you've accepted the license. The only valid value is `accept`	
`image.args.billing` (required)	The billing endpoint URI value is available on the Azure portal's Speech Overview page.	
`image.args.apikey` (required)	Used to track billing information.	
`service.type`	The Kubernetes service type of the **speech-to-text** service. See the Kubernetes service types instructions for more details and verify cloud provider support.	`LoadBalancer`
`service.port`	The port of the **speech-to-text** service.	`80`
`service.annotations`	The **speech-to-text** annotations for the service metadata. Annotations are key value pairs. `annotations:` `some/annotation1: value1` `some/annotation2: value2`	
`service.autoScaler.enabled`	Whether the Horizontal Pod Autoscaler is enabled. If `true`, the `speech-to-text-autoscaler` will be deployed in the Kubernetes cluster.	`true`
`service.podDisruption.enabled`	Whether the Pod Disruption Budget is enabled. If `true`, the `speech-to-text-poddisruptionbudget` will be deployed in the Kubernetes cluster.	`true`

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
`textanalytics.enabled`	Whether the **text-analytics** service is enabled	true/false	`false`
`textanalytics.image.registry`	The **text-analytics** docker image registry	valid docker image registry	

PARAMETER	DESCRIPTION	VALUES	DEFAULT
`textanalytics.image.repository`	The **text-analytics** docker image repository	valid docker image repository	
`textanalytics.image.tag`	The **text-analytics** docker image tag	valid docker image tag	
`textanalytics.image.pullSecrets`	The image secrets for pulling **text-analytics** docker image	valid secrets name	
`textanalytics.image.pullByHash`	Specifies if you are pulling docker image by hash. If `yes`, `image.hash` is required to have as well. If `no`, set it as 'false'. Default is `false`.	true/false	`false`
`textanalytics.image.hash`	The **text-analytics** docker image hash. Only use it with `image.pullByHash:true`.	valid docker image hash	
`textanalytics.image.args.eula`	One of the required arguments by **text-analytics** container, which indicates you've accepted the license. The value of this option must be: `accept`.	`accept`, if you want to use the container	
`textanalytics.image.args.billing`	One of the required arguments by **text-analytics** container, which specifies the billing endpoint URI. The billing endpoint URI value is available on the Azure portal's Speech Overview page.	valid billing endpoint URI	
`textanalytics.image.args.apikey`	One of the required arguments by **text-analytics** container, which is used to track billing information.	valid apikey	
`textanalytics.cpuRequest`	The requested CPU for **text-analytics** container	int	`3000m`
`textanalytics.cpuLimit`	The limited CPU for **text-analytics** container		`8000m`
`textanalytics.memoryRequest`	The requested memory for **text-analytics** container		`3Gi`
`textanalytics.memoryLimit`	The limited memory for **text-analytics** container		`8Gi`

PARAMETER	DESCRIPTION	VALUES	DEFAULT
`textanalytics.service.sentimentURI`	The sentiment analysis URI suffix, the whole URI is in format "http://`<service>`:`<port>`/`<sentimentURISuffix>`".		`text/analytics/v3.0-preview/sentiment`
`textanalytics.service.type`	The type of **text-analytics** service in Kubernetes. See Kubernetes service types	valid Kubernetes service type	`LoadBalancer`
`textanalytics.service.port`	The port of the **text-analytics** service	int	`50085`
`textanalytics.service.annotations`	The annotations users can add to **text-analytics** service metadata. For instance: **annotations:** **some/annotation1: value1** **some/annotation2: value2**	annotations, one per each line	
`textanalytics.serivce.autoScaler.enabled`	Whether Horizontal Pod Autoscaler is enabled. If enabled, `text-analytics-autoscaler` will be deployed in the Kubernetes cluster	true/false	`true`
`textanalytics.service.podDisruptionBudget.enabled`	Whether Pod Disruption Budget is enabled. If enabled, `text-analytics-poddisruptionbudget` will be deployed in the Kubernetes cluster	true/false	`true`

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
`enabled`	Whether the **text-to-speech** service is enabled.	`false`
`numberOfConcurrentRequest`	The number of concurrent requests for the **text-to-speech** service. This chart automatically calculates CPU and memory resources, based on this value.	`2`

PARAMETER	DESCRIPTION	DEFAULT
`optimizeForTurboMode`	Whether the service needs to optimize for text input via text files. If `true`, this chart will allocate more CPU resource to service.	`false`
`image.registry`	The **text-to-speech** docker image registry.	`containerpreview.azurecr.io`
`image.repository`	The **text-to-speech** docker image repository.	`microsoft/cognitive-services-text-to-speech`
`image.tag`	The **text-to-speech** docker image tag.	`latest`
`image.pullSecrets`	The image secrets for pulling the **text-to-speech** docker image.	
`image.pullByHash`	Whether the docker image is pulled by hash. If `true`, `image.hash` is required.	`false`
`image.hash`	The **text-to-speech** docker image hash. Only used when `image.pullByHash: true`.	
`image.args.eula` (required)	Indicates you've accepted the license. The only valid value is `accept`	
`image.args.billing` (required)	The billing endpoint URI value is available on the Azure portal's Speech Overview page.	
`image.args.apikey` (required)	Used to track billing information.	
`service.type`	The Kubernetes service type of the **text-to-speech** service. See the Kubernetes service types instructions for more details and verify cloud provider support.	`LoadBalancer`
`service.port`	The port of the **text-to-speech** service.	`80`

`service.annotations`	The **text-to-speech** annotations for the service metadata. Annotations are key value pairs.
	`annotations:`
	`some/annotation1: value1`
	`some/annotation2: value2`	

`service.autoScaler.enabled`	Whether the Horizontal Pod Autoscaler is enabled. If `true`, the `text-to-speech-autoscaler` will be deployed in the Kubernetes cluster.	`true`

PARAMETER	DESCRIPTION	DEFAULT
`service.podDisruption.enabled`	Whether the Pod Disruption Budget is enabled. If `true`, the `text-to-speech-poddisruptionbudget` will be deployed in the Kubernetes cluster.	`true`

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

11/23/2021 • 7 minutes to read • Edit Online

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
- CLI

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 `cognitive-services`.
Container name	Enter a name such as `cognitive-container-instance`. 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 `Public`. If accessing your container image requires credentials, choose `Private`. Refer to container repositories and images for details on whether or not the container image is `Public` or `Private` ("Public Preview").
Image name	Enter the Cognitive Services container location. The location is what's used as an argument to the `docker pull` command. Refer to the container repositories and images for the available image names and their corresponding repository. The image name must be fully qualified specifying three parts. First, the container registry, then the repository, finally the image name: `<container-registry>/<repository>/<image-name>`. Here is an example, `mcr.microsoft.com/azure-cognitive-services/keyphrase` would represent the Key Phrase Extraction image in the Microsoft Container Registry under the Azure Cognitive Services repository. Another example is, `containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text` which would represent the Speech to Text image in the Microsoft repository of the Container Preview container registry.
OS type	`Linux`
Size	Change size to the suggested recommendations for your specific Cognitive Service container: 2 CPU cores 4 GB

3. On the **Networking** tab, enter the following details:

SETTING	VALUE

SETTING	VALUE
Ports	Set the TCP port to `5000`. 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
`ApiKey`	Copied from the **Keys and endpoint** page of the resource. It is a 32 alphanumeric-character string with no spaces or dashes, `xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`.
`Billing`	Your endpoint URL copied from the **Keys and endpoint** page of the resource.
`Eula`	`accept`

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
- CLI

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.

Migrate from Bing Speech to the Speech service

11/23/2021 • 3 minutes to read • Edit Online

Use this article to migrate your applications from the Bing Speech API to the Speech service.

This article outlines the differences between the Bing Speech APIs and the Speech service, and suggests strategies for migrating your applications. Your Bing Speech API subscription key won't work with the Speech service; you'll need a new Speech service subscription.

A single Speech service subscription key grants access to the following features. Each is metered separately, so you're charged only for the features you use.

- Speech-to-text
- Custom speech-to-text
- Text-to-speech
- Custom text-to-speech voices
- Speech translation (does not include Text translation)

The Speech SDK is a functional replacement for the Bing Speech client libraries, but uses a different API.

Comparison of features

The Speech service is largely similar to Bing Speech, with the following differences.

FEATURE	BING SPEECH	SPEECH SERVICE	DETAILS
C# SDK	✔	✔	Speech service supports Windows 10, Universal Windows Platform (UWP), and .NET Standard 2.0.
C++ SDK	—	✔	Speech service supports Windows and Linux.
Java SDK	✔	✔	Speech service supports Android and Speech Devices.
Continuous speech recognition	10 minutes	Unlimited	The Speech SDK supports unlimited continuous recognition, and automatically reconnects upon timeout or disconnect.
Partial or interim results	✔	✔	Supported with the Speech SDK.
Custom speech models	✔	✔	Bing Speech requires a separate Custom Speech subscription.

FEATURE	BING SPEECH	SPEECH SERVICE	DETAILS
Custom voice fonts	✔	✔	Bing Speech requires a separate Custom Voice subscription.
24-kHz voices	—	✔	
Speech intent recognition	Requires separate LUIS API call	Integrated (with SDK)	You can use a LUIS key with the Speech service.
Simple intent recognition	—	✔	
Batch transcription of long audio files	—	✔	
Recognition mode	Manual via endpoint URI	Automatic	Recognition mode is not available in the Speech service.
Endpoint locality	Global	Regional	Regional endpoints improve latency.
REST APIs	✔	✔	The Speech service REST APIs are compatible with Bing Speech (different endpoint). REST APIs support text-to-speech and limited speech-to-text functionality.
WebSockets protocols	✔	—	The Speech SDK abstracts web socket connections for functionality that requires a constant connection to the service, so there is no longer support to subscribe to them manually.
Service-to-service API calls	✔	—	Provided in Bing Speech via the C# Service Library.
Open-source SDK	✔	—	

The Speech service uses a time-based pricing model (rather than a transaction-based model). See Speech service pricing for details.

Migration strategies

If you or your organization have applications in development or production that use a Bing Speech API, you should update them to use the Speech service as soon as possible. See the Speech service documentation for available SDKs, code samples, and tutorials.

The Speech service REST APIs are compatible with the Bing Speech APIs. If you're currently using the Bing Speech REST APIs, you need only change the REST endpoint, and switch to a Speech service subscription key.

If you're using a Bing Speech client library for a specific programming language, migrating to the Speech SDK

requires changes to your application, because the API is different. The Speech SDK can make your code simpler, while also giving you access to new features. The Speech SDK is available in a wide variety of programming languages. APIs on all platforms are similar, easing multi-platform development.

The Speech service doesn't offer a global endpoint. Determine if your application functions efficiently when it uses a single regional endpoint for all of its traffic. If not, use geolocation to determine the most efficient endpoint. You need a separate Speech service subscription in each region you use.

To get started with the Speech SDK:

1. Download the Speech SDK.
2. Work through the Speech service quickstart guides and tutorials. Also look at the code samples to get experience with the new APIs.
3. Update your application to use the Speech service.

Support

Bing Speech customers should contact customer support by opening a support ticket. You can also contact us if your support need requires a technical support plan.

For Speech service, SDK, and API support, visit the Speech service support page.

Next steps

- Try out Speech service for free
- Get started with speech-to-text
- Get started with text-to-speech

See also

- Speech service release notes
- What is the Speech service
- Speech service and Speech SDK documentation

Migrate from the Translator Speech API to the Speech service

11/23/2021 • 2 minutes to read • Edit Online

Use this article to migrate your applications from the Microsoft Translator Speech API to the Speech service. This guide outlines the differences between the Translator Speech API and Speech service, and suggests strategies for migrating your applications.

> **NOTE**
> Your Translator Speech API subscription key won't be accepted by the Speech service. You'll need to create a new Speech service subscription.

Comparison of features

FEATURE	TRANSLATOR SPEECH API	SPEECH SERVICE	DETAILS
Translation to text	✔	✔	
Translation to speech	✔	✔	
Global endpoint	✔	—	The Speech service doesn't offer a global endpoint. A global endpoint can automatically direct traffic to the nearest regional endpoint, decreasing latency in your application.
Regional endpoints	—	✔	
Connection time limit	90 minutes	Unlimited with the SDK. 10 minutes with a WebSockets connection.	
Auth key in header	✔	✔	
Multiple languages translated in a single request	—	✔	
SDKs available	—	✔	See the Speech service documentation for available SDKs.
WebSockets connections	✔	✔	

FEATURE	TRANSLATOR SPEECH API	SPEECH SERVICE	DETAILS
Languages API	✔	—	The Speech service supports the same range of languages described in the Translator languages reference article.
Profanity Filter and Marker	—	✔	
.WAV/PCM as input	✔	✔	
Other file types as input	—	—	
Partial results	✔	✔	
Timing info	✔	—	
Correlation ID	✔	—	
Custom speech models	—	✔	The Speech service offers custom speech models that enable you to customize speech recognition to your organization's unique vocabulary.
Custom translation models	—	✔	Subscribing to the Microsoft Text Translation API enables you to use Custom Translator to use your own data for more accurate translations.

Migration strategies

If you or your organization have applications in development or production that use the Translator Speech API, you should update them to use the Speech service. See the Speech service documentation for available SDKs, code samples, and tutorials. Consider the following when you are migrating:

- The Speech service doesn't offer a global endpoint. Determine if your application functions efficiently when it uses a single regional endpoint for all of its traffic. If not, use geolocation to determine the most efficient endpoint.
- If your application uses long-lived connections and can't use the available SDKs, you can use a WebSockets connection. Manage the 10-minute timeout limit by reconnecting at the appropriate times.
- If your application uses the Translator service and Translator Speech API to enable custom translation models, you can add Category IDs directly by using the Speech service.
- Unlike the Translator Speech API, the Speech service can complete translations into multiple languages in a single request.

Next steps

- Try out Speech service for free

- Quickstart: Recognize speech in a UWP app using the Speech SDK

See also

- What is the Speech service
- Speech service and Speech SDK documentation

Configure Azure Cognitive Services virtual networks

11/23/2021 • 16 minutes to read • Edit Online

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**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. 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. Currently multi-service resource does not support VNET. 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 or Speech Services, the **CognitiveServicesManagement** tag only enables you use the service using the SDK or REST API. To access and use LUIS portal and/or Speech 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 allow 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
- PowerShell
- Azure CLI

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**.

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
- PowerShell
- Azure CLI

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**.

5. Select the **Virtual networks** and **Subnets** options, and then select **Enable**.

6. To create a new virtual network and grant it access, select **Add new virtual network**.

7. Provide the information necessary to create the new virtual network, and then select **Create**.

> **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**.

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
- PowerShell
- Azure CLI

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.

5. To remove an IP network rule, select the trash can icon next to the address range.

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 service encryption of data at rest

11/23/2021 • 2 minutes to read • Edit Online

Speech Service automatically encrypts your data when it is persisted it 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 your 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 theSpeech 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.

Next steps

- Speech service - bring your own storage (BYOS) request form
- What are managed identities.

Back up and recover speech customer resources

11/23/2021 • 5 minutes to read • Edit Online

The Speech service is available in various regions. Service subscription 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 in the event of a region outage.

How to monitor service availability

If you use our default endpoints, you should configure your client code to monitor for errors, and if errors persist, be prepared to re-direct to another region of your choice where you have a service subscription.

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. Note that 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 subscription key and the region code

5. Configure your code to monitor for connectivity errors (typically connection timeouts and service unavailability errors). Here is 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 very 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 cannot 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 does not 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 Model Copy API 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 Train and 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 do not require real-time failover, you can still follow these steps to prepare for a manual failover.

Offline failover

If you do not 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 is nonetheless advisable to create keys for a primary and secondary region for production models with real-time requirements.

Custom Voice

Custom Voice does not 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**.
   - Note: additional endpoint is subjective to additional 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 failover operations. Speaker enrollments and voice signatures are backed up regularly to prevent data loss and to be used in case of an outage.

During an outage, Speaker Recognition service will automatically failover to a paired region and use the backed up data to continue processing requests until the main region is back online.

Audio processing

11/23/2021 • 2 minutes to read • Edit Online

Audio processing refers to enhancements applied to a stream of audio with a goal of improving the audio quality. A set of enhancements combined are often called an audio processing stack. The goal of improving audio quality can be further segmented into different scenarios like speech processing and telecommunications. Examples of common enhancements include automatic gain control (AGC), noise suppression, and acoustic echo cancellation (AEC).

Different scenarios/use-cases 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.

Microsoft Audio Stack

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.

The Microsoft Audio Stack powers a wide range of Microsoft's 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.

Pricing

There is no cost to using the Microsoft Audio Stack with the Speech SDK.

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 (i.e., 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 downsampling 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

- Learn more about the Speech SDK integration of Microsoft Audio Stack.

Audio processing with Speech SDK

11/23/2021 • 7 minutes to read • Edit Online

The Speech SDK integrates Microsoft Audio Stack (MAS), allowing any application or product to use its audio processing capabilities on input audio. The minimum requirements from Minimum requirements to use Microsoft Audio Stack apply.

Key features made available via Speech SDK APIs 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.

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.

> **NOTE**
>
> The Speech Devices SDK is now deprecated. Archived versions of the Speech Devices SDK are available here, with corresponding sample code available on GitHub. All users of the Speech Devices SDK are advised to migrate to using Speech SDK v1.19 or newer.

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) will need to be configured to provide the reference audio stream as the last channel for the audio input device that will be used. This is in addition to providing the `SpeakerReferenceChannel::LastChannel` option when creating `AudioProcessingOptions`.

Language and platform support

LANGUAGE	PLATFORM(S)	REFERENCE DOCS
C++	Windows, Linux	C++ docs

LANGUAGE	PLATFORM(S)	REFERENCE DOCS
C#	Windows, Linux	C# docs
Java	Windows, Linux	Java docs

Sample code

Using Microsoft Audio Stack with all default options

This sample shows how to use MAS with all default enhancement options on input from the device's default microphone.

- C#
- C++
- Java

```
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);
```

Using Microsoft Audio Stack with a 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 will be 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++
- Java

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

var audioProcessingOptions =
AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCESSING_ENABLE_DEFAULT,
PresetMicrophoneArrayGeometry.Linear2);
var audioInput = AudioConfig.FromMicrophoneInput("hw:0,1", audioProcessingOptions);

var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Using Microsoft Audio Stack with 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 by specifying

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 to be captured from an audio input device corresponding to the custom geometry specified.

- C#
- C++
- Java

```
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(20, 35, 0)
};
var microphoneArrayGeometry = new MicrophoneArrayGeometry(MicrophoneArrayType.Planar,
microphoneCoordinates);
var audioProcessingOptions =
AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCESSING_ENABLE_DEFAULT,
microphoneArrayGeometry, SpeakerReferenceChannel.LastChannel);
var audioInput = AudioConfig.FromWavFileInput("katiesteve.wav", audioProcessingOptions);

var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Using Microsoft Audio Stack with 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++
- Java

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

var audioProcessingOptions =
AudioProcessingOptions.Create(AudioProcessingConstants.AUDIO_INPUT_PROCESSING_DISABLE_ECHO_CANCELLATION |
AudioProcessingConstants.AUDIO_INPUT_PROCESSING_DISABLE_NOISE_SUPPRESSION |
AudioProcessingConstants.AUDIO_INPUT_PROCESSING_ENABLE_DEFAULT);
var audioInput = AudioConfig.FromDefaultMicrophoneInput(audioProcessingOptions);

var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Using Microsoft Audio Stack to 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 to be captured from an audio input device corresponding to the custom geometry specified.

- C#
- C++
- Java

```
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_PROCESSING_ENABLE_DEFAULT,
microphoneArrayGeometry, SpeakerReferenceChannel.LastChannel);
var pushStream = AudioInputStream.CreatePushStream();
var audioInput = AudioConfig.FromStreamInput(pushStream, audioProcessingOptions);

var recognizer = new SpeechRecognizer(speechConfig, audioInput);
```

Speech SDK Microphone array recommendations

11/23/2021 • 4 minutes to read • Edit Online

In this article, you learn how to design a microphone array for the Speech SDK.

The Speech SDK works best with a microphone array that has been designed according to the following guidelines, including the microphone geometry and component selection. Guidance is also given on integration and electrical considerations.

Microphone geometry

The following array geometries are recommended for use with the Microsoft Audio Stack. Location of sound sources and rejection of ambient noise is improved with greater number of microphones with dependencies on specific applications, user scenarios, and the device form factor.

MICS & GEOMETRY	CIRCULAR ARRAY	CIRCULAR ARRAY	LINEAR ARRAY	LINEAR ARRAY
# Mics	7	4	4	2
Geometry	6 Outer, 1 Center, Radius = 42.5 mm, Evenly Spaced	3 Outer, 1 Center, Radius = 42.5 mm, Evenly Spaced	Length = 120 mm, Spacing = 40 mm	Spacing = 40 mm

Microphone channels should be ordered according to the numbering depicted for each above array, increasing from 0. The Microsoft Audio Stack will require an additional reference stream of audio playback to perform echo cancellation.

Component selection

Microphone components should be selected to accurately reproduce a signal free of noise and distortion.

The recommended properties when selecting microphones are:

PARAMETER	RECOMMENDED
SNR	>= 65 dB (1 kHz signal 94 dBSPL, A-weighted noise)
Amplitude Matching	± 1 dB @ 1 kHz
Phase Matching	± 2° @ 1 kHz
Acoustic Overload Point (AOP)	>= 120 dBSPL (THD = 10%)
Bit Rate	Minimum 24-bit

PARAMETER	RECOMMENDED
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

PARAMETER	RECOMMENDED
THD%	3rd Octave Bands minimum 5th Order, 70 dBA Playback @ 0.8 m ≤ 6.3%, 315-500 Hz ≤ 5%, 630-5000 Hz
Echo Coupling to Microphones	> -10 dB TCLw using ITU-T G.122 Annex B.4 method, normalized to mic level TCLw = TCLwmeasured + (Measured Level - Target Output Sensitivity) TCLw = TCLwmeasured + (Measured Level - (-26))

Integration design architecture

The following guidelines for architecture are necessary when integrating microphones into a device:

PARAMETER	RECOMMENDATION
Mic Port Similarity	All microphone ports are same length in array
Mic Port Dimensions	Port size Ø0.8-1.0 mm. Port Length / Port Diameter < 2
Mic Sealing	Sealing gaskets uniformly implemented in stack-up. 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) and interfaces to Speech services or other applications.

Hardware components such as PDM-to-TDM conversion should ensure that the dynamic range and SNR of the microphones is preserved within re-samplers.

High-speed USB Audio Class 2.0 should be supported within any audio MCUs in order to provide the necessary bandwidth for up to seven channels at higher sample rates and bit depths.

Next steps

Learn more about audio processing

What is the Speech Devices SDK?

11/23/2021 • 2 minutes to read • Edit Online

The Speech service works with a wide variety of devices and audio sources. Now, you can take your speech applications to the next level with matched hardware and software. The Speech Devices SDK is a pre-tuned library that's paired with purpose-built, microphone array development kits.

The Speech Devices SDK can help you:

- Rapidly test new voice scenarios.
- More easily integrate the cloud-based Speech service into your device.
- Create an exceptional user experience for your customers.

The Speech Devices SDK consumes the Speech SDK. Using our advanced audio processing algorithms with the device's microphone array to send the audio to the Speech service. It provides accurate far-field speech recognition via noise suppression, echo cancellation, beamforming, and dereverberation.

You can also use the Speech Devices SDK to build ambient devices that have your own customized keyword. A Custom Keyword provides a cue that starts a user interaction which is unique to your brand.

The Speech Devices SDK enables a variety of voice-enabled scenarios, such as voice assistants, drive-thru ordering systems, conversation transcription, and smart speakers. You can respond to users with text, speak back to them in a default or custom voice, provide search results, translate to other languages, and more. We look forward to seeing what you build!

Get the Speech Devices SDK

Android

The Speech Devices SDK for Android supports the Roobo v1 and equivalent devices, for these download the latest version of the Android Speech Devices SDK.

If you have a different Android device, like a phone or mobile, start with the Android Speech SDK

Windows

For Windows, the sample application is provided as a cross-platform Java application. Download the latest version of the JRE Speech Devices SDK. The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Windows. It runs on a 64-bit Java 8 runtime environment (JRE).

Linux

For Linux, the sample application is provided as a cross-platform Java application. Download the latest version of the JRE Speech Devices SDK. The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Linux (Ubuntu Ubuntu 18.04, Debian 9, RHEL 8, CentOS 8). It runs on a 64-bit Java 8 runtime environment (JRE).

Additional binaries are provided to support upcoming devices, Roobo v2 DDK, Urbetter DDK, GGEC Speaker, Linux ARM32, and Linux ARM64.

Next steps

Choose your speech device

Get a Speech service subscription key for free

Get the Cognitive Services Speech Devices SDK

11/23/2021 • 2 minutes to read • Edit Online

The Speech Devices SDK is a pretuned library designed to work with purpose-built development kits, and varying microphone array configurations.

Choose a Development Kit

DEVICES	SPECIFICATION	DESCRIPTION	SCENARIOS
Azure Percept Audio DK Setup / Quickstart 	4 mic linear array with XMOS Codec. Linux	An accessory device that adds speech artificial intelligence (AI) capabilities to your edge device. It contains a pre-configured audio processor and a four-microphone linear array that enable the use of voice commands, keyword spotting, and far field speech with the help of Azure Cognitive Services. This device ships with Azure Percept DK, Azure Percept Studio, and other Azure edge management services to smoothly integrate with our most powerful and compact all-in-one speech devices SDK.	Conversation Transcription, Robotics, Smart Building, Manufacturing, Agriculture
Azure Kinect DK Setup / Quickstart 	7 Mic Array RGB and Depth cameras. Windows/Linux	A spatial computing developer kit with advanced artificial intelligence (AI) sensors for building sophisticated computer vision and speech models. It combines a best-in-class spatial microphone array and depth camera with a video camera and orientation sensor—all in one small device with multiple modes, options, and SDKs that seamlessly integrate with Azure Cognitive Services.	Conversation Transcription, Robotics, Smart Building
Urbetter Dev Kit 	7 Mic Array, ARM SOC, WIFI, Ethernet, HDMI, USB Camera. Linux	An industry level Speech Devices SDK that adapts Microsoft Mic array and supports extended I/O such as HDMI/Ethernet and more USB peripherals Contact Urbetter	Conversation Transcription, Education, Hospital, Robots, OTT Box, Voice Agent, Drive Thru

DEVICES	SPECIFICATION	DESCRIPTION	SCENARIOS
Roobo Smart Audio Dev Kit Setup / Quickstart 	7 Mic Array, ARM SOC, WIFI, Audio Out, IO. Android	The first Speech Devices SDK to adapt Microsoft Mic Array and front processing SDK, for developing high-quality transcription and speech scenarios	Conversation Transcription, Smart Speaker, Voice Agent, Wearable
Roobo Smart Audio Dev Kit 2 Setup 	7 Mic Array, ARM SOC, WIFI, Bluetooth, IO. Linux	The 2nd generation Speech Devices SDK that provides alternative OS and more features in a cost effective reference design.	Conversation Transcription, Smart Speaker, Voice Agent, Wearable

Download the Speech Devices SDK

Download the Speech Devices SDK.

Next steps

Get started with the Speech Devices SDK

Quickstart: Run the Speech Devices SDK sample app on Windows, Linux or Android

11/23/2021 • 14 minutes to read • Edit Online

In this quickstart, you'll learn how to use the Speech Devices SDK for Windows to build a speech-enabled product or use it as a Conversation Transcription device. For Conversation Transcription only the Azure Kinect DK is supported. For other speech use linear mic arrays that provide a microphone array geometry are supported.

The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Windows. It runs on a 64-bit Java 8 runtime environment (JRE).

This guide requires an Azure Cognitive Services account with a Speech service resource.

The source code for the sample application is included with the Speech Devices SDK. It's also available on GitHub.

Prerequisites

This quickstart requires:

- Operating System: 64-bit Windows
- A microphone array such as Azure Kinect DK
- Eclipse Java IDE
- Java 8 or JDK 8 only.
- Microsoft Visual C++ Redistributable
- An Azure subscription key for the Speech service. Get one for free.
- Download the latest version of the Speech Devices SDK for Java, and extract the .zip to your working directory.

> **NOTE**
>
> This quickstart assumes that the app is extracted to C:\SDSDK\JRE-Sample-Release

If you plan to use the intents you'll need a Language Understanding Service (LUIS) subscription. To learn more about LUIS and intent recognition, see Recognize speech intents with LUIS, C#. A sample LUIS model is available for this app.

Create and configure the project

1. Start Eclipse.

2. In the **Eclipse IDE Launcher**, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.

3. In a moment, the main window of the Eclipse IDE appears. Close the Welcome screen if one is present.

4. From the Eclipse menu bar, create a new project by choosing **File** > **New** > **Java Project**. If not available choose **Project** and then **Java Project**.

5. The **New Java Project** wizard starts. **Browse** for the location of the sample project. Select **Finish**.

6. In the **Package explorer**, right-click your project. Choose **Configure** > **Convert to Maven Project** from the context menu. Select **Finish**.

7. Open the pom.xml file and edit it.

At the end of the file, before the closing tag `</project>`, create `repositories` and `dependencies` elements, as shown here, and ensure the `version` matches your current version:

```
<repositories>
     <repository>
         <id>maven-cognitiveservices-speech</id>
         <name>Microsoft Cognitive Services Speech Maven Repository</name>
         <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
     </repository>
</repositories>

<dependencies>
    <dependency>
         <groupId>com.microsoft.cognitiveservices.speech</groupId>
         <artifactId>client-sdk</artifactId>
         <version>1.15.0</version>
    </dependency>
</dependencies>
```

8. Copy the contents of **Windows-x64** to the Java Project location, eg **C:\SDSDK\JRE-Sample-Release**
9. Copy `kws.table`, `participants.properties` and `Microsoft.CognitiveServices.Speech.extension.pma.dll` into the project folder **target\classes**

Configure the sample application

1. Add your speech subscription key to the source code. If you want to try intent recognition, also add your Language Understanding service subscription key and application ID.

   For speech and LUIS, your information goes into `FunctionsList.java`:

```
 // Subscription
 private static String SpeechSubscriptionKey = "<enter your subscription info here>";
 private static String SpeechRegion = "westus"; // You can change this if your speech region is
different.
 private static String LuisSubscriptionKey = "<enter your subscription info here>";
 private static String LuisRegion = "westus2"; // you can change this, if you want to test the
intent, and your LUIS region is different.
 private static String LuisAppId = "<enter your LUIS AppId>";
```

   If you are using conversation transcription, your speech key and region information are also needed in `Cts.java`:

```
 private static final String CTSKey = "<Conversation Transcription Service Key>";
 private static final String CTSRegion="<Conversation Transcription Service Region>";// Region may be
"centralus" or "eastasia"
```

2. The default keyword (keyword) is "Computer". You can also try one of the other provided keywords, like "Machine" or "Assistant". The resource files for these alternate keywords are in the Speech Devices SDK, in the keyword folder. For example, `C:\SDSDK\JRE-Sample-Release\keyword\Computer` contains the files used for the keyword "Computer".

   > **TIP**
   >
   > You can also create a custom keyword.

   To use a new keyword, update the following line in `FunctionsList.java`, and copy the keyword to your

app. For example, to use the keyword 'Machine' from the keyword package `machine.zip`:

- Copy the `kws.table` file from the zip package into the project folder **target/classes**.
- Update the `FunctionsList.java` with the keyword name:

```
private static final String Keyword = "Machine";
```

Run the sample application from Eclipse

1. From the Eclipse menu bar, **Run** > **Run As** > **Java Application**. Then select **FunctionsList** and **OK**.

2. The Speech Devices SDK example application starts and displays the following options:

3. Try the new **Conversation Transcription** demo. Start transcribing with **Session** > **Start**. By default everyone is a guest. However, if you have participant's voice signatures they can be put into a file `participants.properties` in the project folder **target/classes**. To generate the voice signature, look at Transcribe conversations (SDK).

Create and run a standalone application

1. In the **Package explorer**, right-click your project. Choose **Export**.

2. The **Export** window appears. Expand **Java** and select **Runnable JAR file** and then select **Next**.

3. The **Runnable JAR File Export** window appears. Choose an **Export destination** for the application, and then select **Finish**.

4. Please put `kws.table`, `participants.properties`, `unimic_runtime.dll`, `pma.dll` and `Microsoft.CognitiveServices.Speech.extension.pma.dll` in the destination folder chosen above as these files are needed by the application.

5. To run the standalone application

```
java -jar SpeechDemo.jar
```

In this quickstart, you'll learn how to use the Speech Devices SDK for Linux to build a speech-enabled product or use it as a Conversation Transcription device. Currently only the Azure Kinect DK is supported.

The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Linux (Ubuntu Ubuntu 18.04, Debian 9, RHEL 7/8, CentOS 7/8). It runs on a 64-bit Java 8 runtime environment (JRE).

This guide requires an Azure Cognitive Services account with a Speech service resource.

The source code for the sample application is included with the Speech Devices SDK. It's also available on GitHub.

Prerequisites

This quickstart requires:

- Operating System: 64-bit Linux (Ubuntu Ubuntu 18.04, Debian 9, RHEL 7/8, CentOS 7/8)
- Azure Kinect DK
- Eclipse Java IDE
- Java 8 or JDK 8 only.
- An Azure subscription key for the Speech service. Get one for free.
- Download the latest version of the Speech Devices SDK for Java, and extract the .zip to your working directory.

  > **NOTE**
  >
  > This quickstart assumes that the app is extracted to /home/wcaltest/JRE-Sample-Release

Make sure these dependencies are installed before starting Eclipse.

- On Ubuntu:

  ```
  sudo apt-get update
  sudo apt-get install libssl1.0.0 libasound2
  ```

- On Debian 9:

  ```
  sudo apt-get update
  sudo apt-get install libssl1.0.2 libasound2
  ```

- On RHEL/CentOS:

  ```
  sudo yum update
  sudo yum install alsa-lib openssl
  ```

  > **NOTE**
  >
  > - On RHEL/CentOS 7, follow the instructions on how to configure RHEL/CentOS 7 for Speech SDK.

- On RHEL/CentOS 8, follow the instructions on how to configure OpenSSL for Linux.

Conversation Transcription is currently only available for "en-US" and "zh-CN", in the "centralus" and "eastasia" regions. You must have a speech key in one of those regions to use Conversation Transcription.

If you plan to use the intents you'll need a Language Understanding Service (LUIS) subscription. To learn more about LUIS and intent recognition, see Recognize speech intents with LUIS, C#. A sample LUIS model is available for this app.

Create and configure the project

1. Start Eclipse.
2. In the **Eclipse IDE Launcher**, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.

3. In a moment, the main window of the Eclipse IDE appears. Close the Welcome screen if one is present.
4. From the Eclipse menu bar, create a new project by choosing **File** > **New** > **Java Project**. If not available choose **Project** and then **Java Project**.
5. The **New Java Project** wizard starts. **Browse** for the location of the sample project. Select **Finish**.

6. In the **Package explorer**, right-click your project. Choose **Configure** > **Convert to Maven Project** from the context menu. Select **Finish**.

7. Open the pom.xml file and edit it.

   At the end of the file, before the closing tag `</project>`, create `repositories` and `dependencies` elements, as shown here, and ensure the `version` matches your current version:

```
<repositories>
     <repository>
         <id>maven-cognitiveservices-speech</id>
         <name>Microsoft Cognitive Services Speech Maven Repository</name>
         <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
     </repository>
</repositories>

<dependencies>
    <dependency>
         <groupId>com.microsoft.cognitiveservices.speech</groupId>
         <artifactId>client-sdk</artifactId>
         <version>1.15.0</version>
    </dependency>
</dependencies>
```

8. In the **Package explorer**, right-click your project. Choose **Properties**, then **Run/Debug Settings** > **New...** > **Java Application**.

9. The **Edit Configuration** window appears. In the **Name** field enter **Main**, and use **Search** for the **Main Class** to find and select **com.microsoft.cognitiveservices.speech.samples.FunctionsList**.

10. Copy the audio binaries for your target architecture, from either **Linux-arm** or **Linux-x64**, to the Java Project location, eg **/home/wcaltest/JRE-Sample-Release**

11. Also from the **Edit Configuration** window select the **Environment** page and **New**. The **New Environment Variable** window appears. In the **Name** field enter **LD_LIBRARY_PATH** and in the **value** field enter the folder containing the *.so files, for example **/home/wcaltest/JRE-Sample-Release**

12. Copy `kws.table` and `participants.properties` into the project folder **target/classes**

Configure the sample application

1. Add your speech subscription key to the source code. If you want to try intent recognition, also add your Language Understanding service subscription key and application ID.

   For speech and LUIS, your information goes into `FunctionsList.java`:

```
 // Subscription
 private static String SpeechSubscriptionKey = "<enter your subscription info here>";
 private static String SpeechRegion = "westus"; // You can change this if your speech region is
different.
 private static String LuisSubscriptionKey = "<enter your subscription info here>";
 private static String LuisRegion = "westus2"; // you can change this, if you want to test the
intent, and your LUIS region is different.
 private static String LuisAppId = "<enter your LUIS AppId>";
```

   If you are using conversation transcription, your speech key and region information are also needed in `Cts.java`:

```
 private static final String CTSKey = "<Conversation Transcription Service Key>";
 private static final String CTSRegion="<Conversation Transcription Service Region>";// Region may be
"centralus" or "eastasia"
```

2. The default keyword (keyword) is "Computer". You can also try one of the other provided keywords, like "Machine" or "Assistant". The resource files for these alternate keywords are in the Speech Devices SDK,

in the keyword folder. For example, `/home/wcaltest/JRE-Sample-Release/keyword/Computer` contains the files used for the keyword "Computer".

> **TIP**
>
> You can also create a custom keyword.

To use a new keyword, update the following line in `FunctionsList.java`, and copy the keyword to your app. For example, to use the keyword 'Machine' from the keyword package `machine.zip`:

- Copy the `kws.table` file from the zip package into the project folder **target/classes**.
- Update the `FunctionsList.java` with the keyword name:

```
private static final String Keyword = "Machine";
```

Run the sample application from Eclipse

1. From the Eclipse menu bar, **Run** > **Run**
2. The Speech Devices SDK example application starts and displays the following options:

3. Try the new **Conversation Transcription** demo. Start transcribing with **Session** > **Start**. By default everyone is a guest. However, if you have participant's voice signatures they can be put into `participants.properties` in the project folder **target/classes**. To generate the voice signature, look at Transcribe conversations (SDK).

Create and run standalone the application

1. In the **Package explorer**, right-click your project. Choose **Export**.

2. The **Export** window appears. Expand **Java** and select **Runnable JAR file** and then select **Next**.

3. The **Runnable JAR File Export** window appears. Choose an **Export destination** for the application, and then select **Finish**.

4. Please put `kws.table` and `participants.properties` in the destination folder chosen above as these files are needed by the application.

5. Set the LD_LIBRARY_LIB to the folder containing the *.so files

```
export LD_LIBRARY_PATH=/home/wcaltest/JRE-Sample-Release
```

6. To run the standalone application

```
java -jar SpeechDemo.jar
```

In this quickstart, you'll learn how to use the Speech Devices SDK for Android to build a speech-enabled product or use it as a Conversation Transcription device.

This guide requires an Azure Cognitive Services account with a Speech service resource.

The source code for the sample application is included with the Speech Devices SDK. It's also available on GitHub.

Prerequisites

Before you start using the Speech Devices SDK, you'll need to:

- Follow the instructions provided with your development kit to power on the device.
- Download the latest version of the Speech Devices SDK, and extract the .zip to your working directory.

> **NOTE**
>
> This quickstart assumes that the app is extracted to C:\SDSDK\Android-Sample-Release

- To get an Azure subscription key for Speech service
- If you plan to use the Conversation Transcription you must use a circular microphone device and this feature is currently only available for "en-US" and "zh-CN" in regions, "centralus" and "eastasia". You must have a speech key in one of those regions to use Conversation Transcription.
- If you plan to use the Speech service to identify intents (or actions) from user utterances, you'll need a Language Understanding Service (LUIS) subscription. To learn more about LUIS and intent recognition, see Recognize speech intents with LUIS, C#.

  You can create a simple LUIS model or use the sample LUIS model, LUIS-example.json. The sample LUIS model is available from the Speech Devices SDK download site. To upload your model's JSON file to the LUIS portal, select **Import new app**, and then select the JSON file.
- Install Android Studio and Vysor on your PC.

Set up the device

1. Start Vysor on your computer.

2. Your device should be listed under **Choose a device**. Select the **View** button next to the device.

3. Connect to your wireless network by selecting the folder icon, and then select **Settings** > **WLAN**.

P818
P818
4:11 PM
Settings
Wireless & networks
WLAN
Bluetooth
Data usage
More
Device
Home
Display

NOTE

If your company has policies about connecting devices to its Wi-Fi system, you need to obtain the MAC address and contact your IT department about how to connect it to your company's Wi-Fi.

To find the MAC address of the dev kit, select the file folder icon on the desktop of the dev kit.

Select **Settings**. Search for "mac address", and then select **Mac address** > **Advanced WLAN**. Write down the MAC address that appears near the bottom of the dialog box.

Some companies might have a time limit on how long a device can be connected to their Wi-Fi system. You might need to extend the dev kit's registration with your Wi-Fi system after a specific number of days.

Run the sample application

To validate your development kit setup, build and install the sample application:

1. Start Android Studio.

2. Select **Open an existing Android Studio project**.

3. Go to C:\SDSDK\Android-Sample-Release\example. Select **OK** to open the example project.

4. Configure gradle to reference the Speech SDK. The following files can be found under **Gradle Scripts** in Android Studio.

   Update the **build.gradle(Project:example)**, the allprojects block should match below, by adding the maven lines.

```
allprojects {
    repositories {
        google()
        jcenter()
        mavenCentral()
        maven {
            url 'https://csspeechstorage.blob.core.windows.net/maven/'
        }
    }
}
```

   Update the **build.gradle(Module:app)** by adding this line to the dependencies section.

```
implementation'com.microsoft.cognitiveservices.speech:client-sdk:1.19.0'
```

5. Add your speech subscription key to the source code. If you want to try intent recognition, also add your Language Understanding service subscription key and application ID.

   For speech and LUIS, your information goes into MainActivity.java:

```
 // Subscription
 private static String SpeechSubscriptionKey = "<enter your subscription info here>";
 private static String SpeechRegion = "westus"; // You can change this if your speech region is
different.
 private static String LuisSubscriptionKey = "<enter your subscription info here>";
 private static String LuisRegion = "westus2"; // you can change this, if you want to test the
intent, and your LUIS region is different.
 private static String LuisAppId = "<enter your LUIS AppId>";
```

If you are using conversation transcription, your speech key and region information are also needed in conversation.java:

```
 private static final String CTSKey = "<Conversation Transcription Service Key>";
 private static final String CTSRegion="<Conversation Transcription Service Region>";// Region may be
"centralus" or "eastasia"
```

6. The default keyword is "Computer". You can also try one of the other provided keywords, like "Machine" or "Assistant". The resource files for these alternate keywords are in the Speech Devices SDK, in the keyword folder. For example, C:\SDSDK\Android-Sample-Release\keyword\Computer contains the files used for the keyword "Computer".

> **TIP**
>
> You can also create a custom keyword.

To use a new keyword, update the following two lines in `MainActivity.java`, and copy the keyword package to your app. For example, to use the keyword 'Machine' from the keyword package kws-machine.zip:

- Copy the keyword package into the folder "C:\SDSDK\Android-Sample-Release\example\app\src\main\assets".
- Update the `MainActivity.java` with the keyword and the package name:

  ```
  private static final String Keyword = "Machine";
  private static final String KeywordModel = "kws-machine.zip" // set your own keyword package
  name.
  ```

7. Update the following lines, which contain the microphone array geometry settings:

```
private static final String DeviceGeometry = "Circular6+1";
private static final String SelectedGeometry = "Circular6+1";
```

This table lists supported values:

VARIABLE	MEANING	AVAILABLE VALUES
`DeviceGeometry`	Physical mic configuration	For a circular dev kit: `Circular6+1`
		For a linear dev kit: `Linear4`
`SelectedGeometry`	Software mic configuration	For a circular dev kit that uses all mics: `Circular6+1`

VARIABLE	MEANING	AVAILABLE VALUES
		For a circular dev kit that uses four mics: `Circular3+1`
		For a linear dev kit that uses all mics: `Linear4`
		For a linear dev kit that uses two mics: `Linear2`

8. To build the application, on the **Run** menu, select **Run 'app'**. The **Select Deployment Target** dialog box appears.

9. Select your device, and then select **OK** to deploy the application to the device.

10. The Speech Devices SDK example application starts and displays the following options:

11. Try the new Conversation Transcription demo. Start transcribing with 'Start Session'. By default everyone is a guest. However, if you have participant's voice signatures they can be put into a file `/video/participants.properties` on the device. To generate the voice signature, look at Transcribe conversations (SDK).

12. Experiment!

Troubleshooting

If you cannot connect to the Speech Device. Type the following command in a Command Prompt window. It will return a list of devices:

```
adb devices
```

NOTE

This command uses the Android Debug Bridge, `adb.exe`, which is part of the Android Studio installation. This tool is located in C:\Users[user name]\AppData\Local\Android\Sdk\platform-tools. You can add this directory to your path to make it more convenient to invoke `adb`. Otherwise, you must specify the full path to your installation of adb.exe in every command that invokes `adb`.

If you see an error `no devices/emulators found` then check your USB cable is connected and ensure a high quality cable is used.

Next steps

Review the release notes

Troubleshoot the Speech Devices SDK

11/23/2021 • 2 minutes to read • Edit Online

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

Certificate failures

If you get certificate failures when using the Speech service, make sure that your device has the correct date and time:

1. Go to **Settings**. Under **System**, select **Date & time**.

2. Keep the **Automatic date & time** option selected. Under **Select time zone**, select your current time zone.

When you see that the dev kit's time matches the time on your PC, the dev kit is connected to the internet.

Next steps

- Review the release notes

Release notes: Speech Devices SDK

11/23/2021 • 3 minutes to read • Edit Online

The following sections list changes in the most recent releases.

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. For more information, see its release notes.

Speech Devices SDK 0.2.12733: 2018-May release

The first public preview release of the Cognitive Services Speech Devices SDK.

How to configure a Microphone Array

11/23/2021 • 2 minutes to read • Edit Online

In this article, you learn how to configure a microphone array. It includes setting the working angle, and how to select which microphone is used for the Speech Devices SDK.

The Speech Devices SDK works best with a microphone array that has been designed according to our guidelines. The microphone array configuration can be provided by the Operating System or supplied through one of the following methods.

The Speech Devices SDK initially supported microphone arrays by selecting from a fixed set of configurations.

```
private static String DeviceGeometry = "Circular6+1"; // "Circular6+1", "Linear4",
private static String SelectedGeometry = "Circular6+1"; // "Circular6+1", "Circular3+1", "Linear4",
"Linear2"
```

On Windows, the microphone array configuration is supplied by the audio driver.

From v1.11.0, the Speech Devices SDK also supports configuration from a JSON file.

Windows

On Windows, the microphone array geometry information is automatically obtained from the audio driver. So, the properties `DeviceGeometry`, `SelectedGeometry`, and `MicArrayGeometryConfigFile` are optional. We use the JSON file provided using `MicArrayGeometryConfigFile` for only getting the beamforming range.

If a microphone array is specified using `AudioConfig::FromMicrophoneInput`, then we use the specified microphone. If a microphone is not specified or `AudioConfig::FromDefaultMicrophoneInput` is called, then we use the default microphone, which is specified in Sound settings on Windows. The Microsoft Audio Stack in the Speech Devices SDK only supports down sampling for sample rates that are integral multiples of 16 KHz.

Linux

On Linux, the microphone geometry information must be provided. The use of `DeviceGeometry` and `SelectedGeometry` remains supported. It can also be provided via the JSON file using the `MicArrayGeometryConfigFile` property. Similar to Windows, the beamforming range can be provided by the JSON file.

If a microphone array is specified using `AudioConfig::FromMicrophoneInput`, then we use the specified microphone. If a microphone is not specified or `AudioConfig::FromDefaultMicrophoneInput` is called, then we record from ALSA device named *default*. By default, *default* always points to card 0 device 0, but users can change it in the `asound.conf` file.

The Microsoft Audio Stack in the Speech Devices SDK only supports downsampling for sample rates that are integral multiples of 16 KHz. 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.

Android

Currently only Roobo v1 is supported by the Speech Devices SDK. Behavior is same as previous releases, except now `MicArrayGeometryConfigFile` property can be used to specify JSON file containing beamforming range.

Microphone Array Configuration JSON

The JSON file for microphone array geometry configuration will follow the JSON schema. Following are some examples that follow the schema.

```
{
    "micArrayType": "Linear",
    "geometry": "Linear4"
}
```

Or

```
{
    "micArrayType": "Planar",
    "horizontalAngleBegin": 0,
    "horizontalAngleEnd": 360,
    "numberOfMicrophones": 4,
    "micCoord": [
        {
            "xCoord": 0,
            "yCoord": 0,
            "zCoord": 0
        },
        {
            "xCoord": 40,
            "yCoord": 0,
            "zCoord": 0
        },
        {
            "xCoord": -20,
            "yCoord": -35,
            "zCoord": 0
        },
        {
            "xCoord": -20,
            "yCoord": 35,
            "zCoord": 0
        }
    ]
}
```

Or

```
{
    "micArrayType": "Linear",
    "horizontalAngleBegin": 70,
    "horizontalAngleEnd": 110
}
```

Next steps

Choose your speech device

Device: Roobo Smart Audio Dev Kit

11/23/2021 • 2 minutes to read • Edit Online

This article provides device specific information for the Roobo Smart Audio Dev Kit.

Set up the development kit

1. The development kit has two micro USB connectors. The left connector is to power the development kit and is highlighted as Power in the image below. The right one is to control it, and is marked Debug in the image.

2. Power the development kit by using a micro USB cable to connect the power port to a PC or power adapter. A green power indicator will light up under the top board.
3. To control the development kit, connect the debug port to a computer by using a second micro USB cable. It is essential to use a high quality cable to ensure reliable communications.
4. Orient your development kit for either the circular or linear configuration.

DEVELOPMENT KIT CONFIGURATION	ORIENTATION
Circular	Upright, with microphones facing the ceiling
Linear	On its side, with microphones facing you (shown in the following image)

5. Install the certificates and set the permissions of the sound device. Type the following commands in a Command Prompt window:

```
adb push C:\SDSDK\Android-Sample-Release\scripts\roobo_setup.sh /data/
adb shell
cd /data/
chmod 777 roobo_setup.sh
./roobo_setup.sh
exit
```

> **NOTE**
>
> These commands use the Android Debug Bridge, `adb.exe`, which is part of the Android Studio installation. This tool is located in C:\Users[user name]\AppData\Local\Android\Sdk\platform-tools. You can add this directory to your path to make it more convenient to invoke `adb`. Otherwise, you must specify the full path to your installation of adb.exe in every command that invokes `adb`.
>
> If you see an error `no devices/emulators found` then check your USB cable is connected and is a high quality cable. You can use `adb devices` to check that your computer can talk to the development kit as it will return a list of devices.

> **TIP**
>
> Mute your PC's microphone and speaker to be sure you are working with the development kit's microphones. This way, you won't accidentally trigger the device with audio from the PC.

6. If you want to attach a speaker to the dev kit, you can connect it to the audio line out. You should choose a good-quality speaker with a 3.5mm analog plug.

Development information

For more development information, see the Roobo development guide.

Audio

Roobo provides a tool that captures all audio to flash memory. It might help you troubleshoot audio issues. A version of the tool is provided for each development kit configuration. On the Roobo site, select your device, and then select the **Roobo Tools** link at the bottom of the page.

Next steps

- Run the Android sample app

Device: Roobo Smart Audio Dev Kit v2

11/23/2021 • 2 minutes to read • Edit Online

This article provides device-specific information for the Roobo Smart Audio Dev Kit2.

Set up the development kit

1. The development kit has two micro USB connectors. The left connector is to power the development kit and is highlighted as Power in the image below. The right one is to control it, and is marked Debug in the image.

2. Power the development kit by using a micro USB cable to connect the power port to a PC or power adapter. A green power indicator will light up under the top board.
3. To control the development kit, connect the debug port to a computer by using a second micro USB cable. It is essential to use a high-quality cable to ensure reliable communications.
4. Orient your development kit Circularly - Upright, with microphones facing the ceiling as shown above

Development information

For more development information, see the Roobo development guide.

Audio Record/Play

DDK2 audio operations can be performed in the following ways:

- Use ALSA Open-source libraries and their applications.

- Use appmainprog interface to do application development. DDK2 audio - related software framework uses standard ALSA framework, you can use libasound. So to develop software directly. So you can use ALSA's arecord and aplay directly to record and play audio.

About the Speech SDK

11/23/2021 • 17 minutes to read • Edit Online

The Speech software development kit (SDK) exposes many of the Speech service capabilities, to empower you to develop speech-enabled applications. The Speech SDK is available in many programming languages and across all platforms.

PROGRAMMING LANGUAGE	PLATFORM	SDK REFERENCE
C# [1]	Windows, Linux, macOS, Mono, Xamarin.iOS, Xamarin.Mac, Xamarin.Android, UWP, Unity	.NET SDK
C++	Windows, Linux, macOS	C++ SDK
Go	Linux	Go SDK
Java [2]	Android, Windows, Linux, macOS	Java SDK
JavaScript	Browser, Node.js	JavaScript SDK
Objective-C / Swift	iOS, macOS	Objective-C SDK
Python	Windows, Linux, macOS	Python SDK

1 The .NET Speech SDK is based on .NET Standard 2.0, thus it supports many platforms. For more information, see .NET implementation support.

2 The Java Speech SDK is also available as part of the Speech Devices SDK.

> **IMPORTANT**
>
> C is not a supported programming language for the Speech SDK. Several supported programming languages, e.g. C++, include C headers that are part of a common Application Binary Interface (ABI) layer. These ABI headers are **not** intended for direct use and are subject to change across versions.

Scenario capabilities

The Speech SDK exposes many features from the Speech service, but not all of them. The capabilities of the Speech SDK are often associated with scenarios. The Speech SDK is ideal for both real-time and non-real-time scenarios, using local devices, files, Azure blob storage, and even input and output streams. When a scenario is not achievable with the Speech SDK, look for a REST API alternative.

Speech-to-text

Speech-to-text (also known as *speech recognition*) transcribes audio streams to text that your applications, tools, or devices can consume or display. Use speech-to-text with Language Understanding (LUIS) to derive user intents from transcribed speech and act on voice commands. Use Speech Translation to translate speech input to a different language with a single call. For more information, see Speech-to-text basics.

Speech-Recognition (SR), Phrase List, Intent, Translation, and On-premises containers are available on the following platforms:

- C++/Windows & Linux & macOS
- C# (Framework & .NET Core)/Windows & UWP & Unity & Xamarin & Linux & macOS
- Java (Jre and Android)
- JavaScript (Browser and NodeJS)
- Python
- Swift
- Objective-C
- Go (SR only)

Text-to-speech

Text-to-speech (also known as *speech synthesis*) converts text into human-like synthesized speech. The input text is either string literals or using the Speech Synthesis Markup Language (SSML). For more information on standard or neural voices, see Text-to-speech language and voice support.

Text-to-speech (TTS) is available on the following platforms:

- C++/Windows & Linux & macOS
- C# (Framework & .NET Core)/Windows & UWP & Unity & Xamarin & Linux & macOS
- Java (Jre and Android)
- JavaScript (Browser and NodeJS)
- Python
- Swift
- Objective-C
- Go
- TTS REST API can be used in every other situation.

Voice assistants

Voice assistants using the Speech SDK enable you to create natural, human-like conversational interfaces for your applications and experiences. The Speech SDK provides fast, reliable interaction that includes speech-to-text, text-to-speech, and conversational data on a single connection. Your implementation can use the Bot Framework's Direct Line Speech channel or the integrated Custom Commands service for task completion. Additionally, voice assistants can use custom voices created in the Custom Voice Portal to add a unique voice output experience.

Voice assistant support is available on the following platforms:

- C++/Windows & Linux & macOS
- C#/Windows
- Java/Windows & Linux & macOS & Android (Speech Devices SDK)
- Go

Keyword recognition

The concept of keyword recognition is supported in the Speech SDK. Keyword recognition is the act of identifying a keyword in speech, followed by an action upon hearing the keyword. For example, "Hey Cortana" would activate the Cortana assistant.

Keyword recognition is available on the following platforms:

- C++/Windows & Linux
- C#/Windows & Linux
- Python/Windows & Linux
- Java/Windows & Linux & Android

Meeting scenarios

The Speech SDK is perfect for transcribing meeting scenarios, whether from a single device or multi-device conversation.

Conversation Transcription

Conversation Transcription enables real-time (and asynchronous) speech recognition, speaker identification, and sentence attribution to each speaker (also known as *diarization*). It's perfect for transcribing in-person meetings with the ability to distinguish speakers.

Conversation Transcription is available on the following platforms:

- C++/Windows & Linux
- C# (Framework & .NET Core)/Windows & UWP & Linux
- Java/Windows & Linux & Android (Speech Devices SDK)

Multi-device Conversation

With Multi-device Conversation, connect multiple devices or clients in a conversation to send speech-based or text-based messages, with easy support for transcription and translation.

Multi-device Conversation is available on the following platforms:

- C++/Windows
- C# (Framework & .NET Core)/Windows

Custom / agent scenarios

The Speech SDK can be used for transcribing call center scenarios, where telephony data is generated.

Call Center Transcription

Call Center Transcription is common scenario for speech-to-text for transcribing large volumes of telephony data that may come from various systems, such as Interactive Voice Response (IVR). The latest speech recognition models from the Speech service excel at transcribing this telephony data, even in cases when the data is difficult for a human to understand.

Call Center Transcription is available through the Batch Speech Service via its REST API and can be used in any situation.

Codec compressed audio input

Several of the Speech SDK programming languages support codec compressed audio input streams. For more information, see use compressed audio input formats .

Codec compressed audio input is available on the following platforms:

- C++/Linux
- C#/Linux
- Java/Linux, Android, and iOS

REST API

While the Speech SDK covers many feature capabilities of the Speech Service, for some scenarios you might want to use the REST API.

Batch transcription

Batch transcription enables asynchronous speech-to-text transcription of large volumes of data. Batch transcription is only possible from the REST API. In addition to converting speech audio to text, batch speech-to-text also allows for diarization and sentiment-analysis.

Customization

The Speech Service delivers great functionality with its default models across speech-to-text, text-to-speech, and speech-translation. Sometimes you may want to increase the baseline performance to work even better with your unique use case. The Speech Service has a variety of no-code customization tools that make it easy, and allow you to create a competitive advantage with custom models based on your own data. These models will only be available to you and your organization.

Custom Speech-to-text

When using speech-to-text for recognition and transcription in a unique environment, you can create and train custom acoustic, language, and pronunciation models to address ambient noise or industry-specific vocabulary. The creation and management of no-code Custom Speech models is available through the Custom Speech Portal. Once the Custom Speech model is published, it can be consumed by the Speech SDK.

Custom Text-to-speech

Custom text-to-speech, also known as Custom Voice is a set of online tools that allow you to create a recognizable, one-of-a-kind voice for your brand. The creation and management of no-code Custom Voice models is available through the Custom Voice Portal. Once the Custom Voice model is published, it can be consumed by the Speech SDK.

Get the Speech SDK

- Windows
- Linux
- iOS
- macOS
- Android
- Node.js
- Browser

The Speech SDK supports Windows 10 and Windows Server 2016, or later versions. Earlier versions are **not** officially supported. It is possible to use parts of the Speech SDK with earlier versions of Windows, although it's not advised.

System requirements

The Speech SDK on Windows requires the Microsoft Visual C++ Redistributable for Visual Studio 2019 on the system.

- Install for x86
- Install for x64
- Install for ARMx64

C#

The .NET Speech SDK is available as a NuGet package and implements .NET Standard 2.0, for more information, see Microsoft.CognitiveServices.Speech .

C#

C# NuGet Package

The .NET Speech SDK can be installed from the **.NET Core CLI** with the following `dotnet add` command.

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

The .NET Speech SDK can be installed from the **Package Manager** with the following `Install-Package` command.

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

Additional resources

- .NET Core, .NET Framework, Unity, UWP, and Xamarin quickstart C# source code

For microphone input, the Media Foundation libraries must be installed. These libraries are part of Windows 10 and Windows Server 2016. It's possible to use the Speech SDK without these libraries, as long as a microphone isn't used as the audio input device.

The required Speech SDK files can be deployed in the same directory as your application. This way your application can directly access the libraries. Make sure you select the correct version (x86/x64) that matches your application.

NAME	FUNCTION
`Microsoft.CognitiveServices.Speech.core.dll`	Core SDK, required for native and managed deployment
`Microsoft.CognitiveServices.Speech.csharp.dll`	Required for managed deployment

> **NOTE**
> Starting with the release 1.3.0 the file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated in the core SDK.

> **IMPORTANT**
> For the Windows Forms App (.NET Framework) C# project, make sure the libraries are included in your project's deployment settings. You can check this under `Properties -> Publish Section`. Click the `Application Files` button and find corresponding libraries from the scroll down list. Make sure the value is set to `Included`. Visual Studio will include the file when project is published/deployed.

C++

The C++ Speech SDK is available as a NuGet package on Windows, Linux, and macOS. For more information, see Microsoft.CognitiveServices.Speech. The C++ Speech SDK is also available as tar package from https://aka.ms/csspeech/linuxbinary.

C++

C++ NuGet package

The C++ Speech SDK can be installed from the **Package Manager** with the following `Install-Package` command.

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

Additional resources

- Windows, Linux, and macOS quickstart C++ source code

Python

The Python Speech SDK is available as a Python Package Index (PyPI) module, for more information, see azure-cognitiveservices-speech . The Python Speech SDK is compatible with Windows, Linux, and macOS.

Python

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

> **TIP**
> If you are on macOS, you may need to run the following command to get the pip command above to work:
>
> ```
> python3 -m pip install --upgrade pip
> ```

Additional resources

- Python Speech SDK quickstart source code

Java

The Java SDK for Android is packaged as an AAR (Android Library) , which includes the necessary libraries and required Android permissions. It's hosted in a Maven repository at `https://csspeechstorage.blob.core.windows.net/maven/` as package `com.microsoft.cognitiveservices.speech:client-sdk:1.19.0`. (Make sure 1.19.0 is the latest version by searching our GitHub repo.)

Java

To consume the package from your Android Studio project, make the following changes:

1. In the project-level *build.gradle* file, add the following to the `repositories` section:

```
maven { url 'https://csspeechstorage.blob.core.windows.net/maven/' }
```

2. In the module-level *build.gradle* file, add the following to the `dependencies` section:

```
implementation 'com.microsoft.cognitiveservices.speech:client-sdk:1.19.0'
```

The Java SDK is also part of the Speech Devices SDK.

Additional resources

- Android specific Java quickstart source code
- Java Runtime (JRE) quickstart source code

> **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

Sample source code

The Speech SDK team actively maintains a large set of examples in an open-source repository. For the sample source code repository, visit the Microsoft Cognitive Services Speech SDK on GitHub . There are samples for C#, C++, Java, Python, Objective-C, Swift, JavaScript, UWP, Unity, and Xamarin.

GitHub

Next steps

- Create a free Azure account
- See how to recognize speech in C#

Quickstart: Setup development environment

11/23/2021 • 30 minutes to read • Edit Online

Choose your target environment

- .NET
- .NET Core
- Unity
- UWP
- Xamarin

This guide shows how to install the Speech SDK for .NET Framework (Windows). If you just want the package name to get started on your own, run `Install-Package Microsoft.CognitiveServices.Speech` in the NuGet console.

> **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

Prerequisites

This quickstart requires:

- On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
- Visual Studio 2019

Create a Visual Studio project and install the Speech SDK

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

Create helloworld project

1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.
3. In the **Create a new project** window, choose **Console App (.NET Framework)**, and then select **Next**.
4. In the **Configure your new project** window, enter *helloworld* in **Project name**, choose or create the directory path in **Location**, and then select **Create**.
5. From the Visual Studio menu bar, select **Tools** > **Get Tools and Features**, which opens Visual Studio Installer and displays the **Modifying** dialog box.
6. Check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, select the check box next to it, and then select **Modify** to start the installation. It may take a few minutes to download and install.

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

7. Close Visual Studio Installer.

Use NuGet Package Manager to install the Speech SDK

1. In the Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show the NuGet Package Manager.

2. In the upper-right corner, find the **Package Source** drop-down box, and make sure that `nuget.org` is selected.

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

4. In the search box, type *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 matching your computer's architecture.

1. From the menu bar, select **Build** > **Configuration Manager**. The **Configuration Manager** dialog box appears.

2. In the **Active solution platform** drop-down box, select **New**. The **New Solution Platform** dialog box appears.

3. In the **Type or select the new platform** drop-down box:

   - If you're running 64-bit Windows, select **x64**.
   - If you're running 32-bit Windows, select **x86**.

4. Select **OK** and then **Close**.

Next steps

- Quickstart: Recognize speech from a microphone
- Quickstart: Recognize speech from a file
- Quickstart: Recognize speech from an Azure Blob
- Quickstart: Translate speech-to-text
- Quickstart: Synthesize text to an audio device
- Quickstart: Synthesize text to a file
- Quickstart: Recognize Intents

Choose your target environment

- Linux
- macOS
- Windows

This guide shows how to install the Speech SDK for Linux

> **IMPORTANT**
>
> By downloading any of the 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

System requirements

Linux; see the list of supported Linux distributions and target architectures.

Prerequisites

See system requirements and setup instructions on Linux.

Install Speech SDK

The Speech SDK for Linux can be used to build both 64-bit and 32-bit applications. The required libraries and header files can be downloaded as a tar file from https://aka.ms/csspeech/linuxbinary.

Download and install the SDK as follows:

1. Choose a directory to which the Speech SDK files should be extracted, and set the `SPEECHSDK_ROOT` environment variable to point to that directory. This variable makes it easy to refer to the directory in future commands. For example, if you want to use the directory `speechsdk` in your home directory, use a command like the following:

   ```
   export SPEECHSDK_ROOT="$HOME/speechsdk"
   ```

2. Create the directory if it doesn't exist yet.

   ```
   mkdir -p "$SPEECHSDK_ROOT"
   ```

3. Download and extract the `.tar.gz` archive containing the Speech SDK binaries:

```
wget -O SpeechSDK-Linux.tar.gz https://aka.ms/csspeech/linuxbinary
tar --strip 1 -xzf SpeechSDK-Linux.tar.gz -C "$SPEECHSDK_ROOT"
```

4. Validate the contents of the top-level directory of the extracted package:

```
ls -l "$SPEECHSDK_ROOT"
```

The directory listing should contain the third-party notice and license files, as well as an `include` directory containing header (`.h`) files and a `lib` directory containing libraries.

PATH	DESCRIPTION
`license.md`	License
`ThirdPartyNotices.md`	Third-party notices.
`REDIST.txt`	Redistribution notice.
`include`	The required header files for C and C++
`lib/arm32`	Native library for ARM32 required to link your application
`lib/arm64`	Native library for ARM64 required to link your application
`lib/x64`	Native library for x64 required to link your application
`lib/x86`	Native library for x86 required to link your application

Next steps

- Quickstart: Recognize speech from a microphone
- Quickstart: Recognize speech from a file
- Quickstart: Recognize speech from an Azure Blob
- Quickstart: Translate speech-to-text
- Quickstart: Synthesize text to an audio device
- Quickstart: Synthesize text to a file
- Quickstart: Recognize Intents

Choose your target environment

- Java Runtime
- Android

This guide shows how to install the Speech SDK for 64-bit Java 8 JRE. If you just want the package name to get started on your own, the Java SDK is not available in the Maven central repository. Whether you're using Gradle or a `pom.xml` dependency file, you need to add a custom repository pointing to `https://csspeechstorage.blob.core.windows.net/maven/` (see below for package name).

> **NOTE**
> For the Speech Devices SDK and the Roobo device, see Speech Devices SDK.

> **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

Supported operating systems

- The Java Speech SDK package is available for these operating systems:
  - Windows: 64-bit only
  - Mac: macOS X version 10.14 or later
  - Linux; see the list of supported Linux distributions and target architectures.

Prerequisites

- Java 8 or JDK 8
- Eclipse Java IDE (requires Java already installed)
- On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
- On Linux, see the system requirements and setup instructions.

Gradle config

Gradle configs require both a custom repository, and an explicit reference to the dependency extension `.jar`.

```
// build.gradle

repositories {
    maven {
        url "https://csspeechstorage.blob.core.windows.net/maven/"
    }
}

dependencies {
    implementation group: 'com.microsoft.cognitiveservices.speech', name: 'client-sdk', version: "1.18.0",
ext: "jar"
}
```

Create an Eclipse project and install the Speech SDK

1. Start Eclipse.
2. In the Eclipse Launcher, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.

3. In a moment, the main window of the Eclipse IDE appears. Close the **Welcome** screen if one is present.

4. From the Eclipse menu bar, create a new project by choosing **File** > **New** > **Project**.

5. The **New Project** dialog box appears. Select **Java Project**, and select **Next**.

6. The **New Java Project** wizard starts. In the **Project name** field, enter **quickstart**, and choose **JavaSE-1.8** as the execution environment. Select **Finish**.

7. If the **Open Associated Perspective?** window appears, select **Open Perspective**.

8. In the **Package explorer**, right-click the **quickstart** project. Choose **Configure** > **Convert to Maven Project** from the context menu.

9. The **Create new POM** window appears. In the **Group Id** field, enter *com.microsoft.cognitiveservices.speech.samples*, and in the **Artifact Id** field, enter *quickstart*. Then select **Finish**.

10. Open the *pom.xml* file and edit it.

    - At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK, as shown here:

    ```
    <repositories>
      <repository>
        <id>maven-cognitiveservices-speech</id>
        <name>Microsoft Cognitive Services Speech Maven Repository</name>
        <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
      </repository>
    </repositories>
    ```

    - Also add a `dependencies` element, with the Speech SDK version 1.18.0 as a dependency:

    ```
    <dependencies>
      <dependency>
        <groupId>com.microsoft.cognitiveservices.speech</groupId>
        <artifactId>client-sdk</artifactId>
        <version>1.19.0</version>
      </dependency>
    </dependencies>
    ```

    - Save the changes.

Next steps

- Quickstart: Recognize speech from a microphone
- Quickstart: Recognize speech from a file
- Quickstart: Recognize speech from an Azure Blob
- Quickstart: Translate speech-to-text
- Quickstart: Synthesize text to an audio device
- Quickstart: Synthesize text to a file
- Quickstart: Recognize Intents

This guide shows how to install the Speech SDK for Python. If you just want the package name to get started on your own, run `pip install azure-cognitiveservices-speech`.

> **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

Supported operating systems

- The Python Speech SDK package is available for these operating systems:
  - Windows: x64 and x86
  - Mac: macOS X version 10.14 or later (the minimum Python version on macOS is 3.7)
  - Linux; see the list of supported Linux distributions and target architectures.

Prerequisites

- On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
- On Linux, see the system requirements and setup instructions.
- And finally, you'll need Python 3.6 to 3.9. To check your installation, open a command prompt and type the command `python --version` and check the result. If it's installed properly, you'll get a response like "Python 3.8.8".

Install the Speech SDK from PyPI

If you're using your own environment or build tools, run the following command to install the Speech SDK from PyPI. For users of Visual Studio Code, skip to the next sub-section for guided installation.

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

If you are on macOS, you may need to run the following command to get the `pip` command above to work:

```
python3 -m pip install --upgrade pip
```

Once you've successfully used `pip` to install `azure-cognitiveservices-speech`, you can use the Speech SDK by importing the namespace into your Python projects.

```
import azure.cognitiveservices.speech as speechsdk
```

Install the Speech SDK using Visual Studio Code

1. Download and install the latest supported version of Python for your platform, 3.6 to 3.9.
   - Windows users make sure to select "Add Python to your PATH" during the installation process.
2. Download and install Visual Studio Code.
3. Open Visual Studio Code and install the Python extension. Select **File** > **Preferences** > **Extensions** from the menu. Search for **Python** and click **Install**.

4. Also from within Visual Studio Code, install the Speech SDK Python package from the integrated command line:
   a. Open a terminal (from the drop-down menus, **Terminal** > **New Terminal**)
   b. In the terminal that opens, enter the command `python -m pip install azure-cognitiveservices-speech`

If you are new to Visual Studio Code, refer to the more extensive Visual Studio Code Documentation. For more information about Visual Studio Code and Python, see Visual Studio Code Python tutorial.

Support and updates

Updates to the Speech SDK Python package are distributed via PyPI and announced in the Release notes. If a new version is available, you can update to it with the command `pip install --upgrade azure-cognitiveservices-speech`. Check which version is currently installed by inspecting the `azure.cognitiveservices.speech.__version__` variable.

If you have a problem, or you're missing a feature, see Support and help options.

Next steps

- Quickstart: Recognize speech from a microphone
- Quickstart: Recognize speech from a file
- Quickstart: Recognize speech from an Azure Blob
- Quickstart: Translate speech-to-text
- Quickstart: Synthesize text to an audio device
- Quickstart: Synthesize text to a file
- Quickstart: Recognize Intents

This guide shows how to install the Speech SDK for GO on Linux.

> **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

System requirements

Linux; see the list of supported Linux distributions and target architectures.

Prerequisites

To complete this quickstart, you'll need:

- Go binary (1.13 or later)
- Linux environment set up as per system requirements and setup instructions.

Install Speech SDK

The Speech SDK for Linux can be used to build both 64-bit and 32-bit applications. The required libraries and header files can be downloaded as a tar file from https://aka.ms/csspeech/linuxbinary.

Download and install the SDK as follows:

1. Choose a directory to which the Speech SDK files should be extracted, and set the `SPEECHSDK_ROOT` environment variable to point to that directory. This variable makes it easy to refer to the directory in future commands. For example, if you want to use the directory `speechsdk` in your home directory, use a command like the following:

   ```
   export SPEECHSDK_ROOT="$HOME/speechsdk"
   ```

2. Create the directory if it doesn't exist yet.

   ```
   mkdir -p "$SPEECHSDK_ROOT"
   ```

3. Download and extract the `.tar.gz` archive containing the Speech SDK binaries:

   ```
   wget -O SpeechSDK-Linux.tar.gz https://aka.ms/csspeech/linuxbinary
   tar --strip 1 -xzf SpeechSDK-Linux.tar.gz -C "$SPEECHSDK_ROOT"
   ```

4. Validate the contents of the top-level directory of the extracted package:

   ```
   ls -l "$SPEECHSDK_ROOT"
   ```

   The directory listing should contain the third-party notice and license files, as well as an `include` directory containing header (`.h`) files and a `lib` directory containing libraries.

PATH	DESCRIPTION
`license.md`	License

PATH	DESCRIPTION
`ThirdPartyNotices.md`	Third-party notices.
`REDIST.txt`	Redistribution notice.
`include`	The required header files for C and C++
`lib/arm32`	Native library for ARM32 required to link your application
`lib/arm64`	Native library for ARM64 required to link your application
`lib/x64`	Native library for x64 required to link your application
`lib/x86`	Native library for x86 required to link your application

Configure Go environment

Perform the following steps to set up your Go environment to find the Speech SDK. In both steps, replace `<architecture>` with the processor architecture of your CPU. This will be `x86`, `x64`, `arm32`, or `arm64`.

1. Since the bindings rely on `cgo`, you need to set the environment variables so Go can find the SDK:

```
export CGO_CFLAGS="-I$SPEECHSDK_ROOT/include/c_api"
export CGO_LDFLAGS="-L$SPEECHSDK_ROOT/lib/<architecture> -lMicrosoft.CognitiveServices.Speech.core"
```

2. To run applications including the SDK, we need to tell the OS where to find the libs:

```
export LD_LIBRARY_PATH="$SPEECHSDK_ROOT/lib/<architecture>:$LD_LIBRARY_PATH"
```

Next steps

- Quickstart: Recognize speech from a microphone
- Quickstart: Recognize speech from a file
- Quickstart: Create a custom voice assistant

Choose your target environment

- Browser-based
- NodeJS

This guide shows how to install the Speech SDK for JavaScript for use with a web page.

> **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

Create a new Website folder

Create a new, empty folder. In case you want to host the sample on a web server, make sure that the web server can access the folder.

Unpack the Speech SDK for JavaScript into that folder

Download the Speech SDK as a .zip package and unpack it into the newly created folder. This results in five files being unpacked:

- `microsoft.cognitiveservices.speech.sdk.bundle.js` A human readable version of the Speech SDK.
- `microsoft.cognitiveservices.speech.sdk.bundle.js.map` A map file used for debugging SDK code.
- `microsoft.cognitiveservices.speech.sdk.bundle.d.ts` Object definitions for use with TypeScript
- `microsoft.cognitiveservices.speech.sdk.bundle-min.js` A minified version of the Speech SDK.
- `speech-processor.js` Code to improve performance on some browsers.

Create an index.html page

Create a new file in the folder, named `index.html` and open this file with a text editor.

Next steps

- Quickstart: Recognize speech from a microphone
- Quickstart: Recognize speech from a file
- Quickstart: Recognize speech from an Azure Blob
- Quickstart: Translate speech-to-text
- Quickstart: Synthesize text to an audio device
- Quickstart: Synthesize text to a file
- Quickstart: Recognize Intents

Troubleshoot the Speech SDK

11/23/2021 • 2 minutes to read • Edit Online

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

Error: WebSocket Upgrade failed with an authentication error (403)

You might have the wrong endpoint for your region or service. Check the URI to make sure it's correct.

Also, there might be a problem with your subscription key or authorization token. For more information, see the next section.

Error: HTTP 403 Forbidden or HTTP 401 Unauthorized

This error often is caused by authentication issues. Connection requests without a valid `Ocp-Apim-Subscription-Key` or `Authorization` header are rejected with a status of 403 or 401.

- If you're using a subscription key for authentication, you might see the error because:
  - The subscription key is missing or invalid
  - You have exceeded your subscription's usage quota
- If you're using an authorization token for authentication, you might see the error because:
  - The authorization token is invalid
  - The authorization token is expired

Validate your subscription key

You can verify that you have a valid subscription key by running one of the following commands.

> **NOTE**
>
> Replace `YOUR_SUBSCRIPTION_KEY` and `YOUR_REGION` with your own subscription key and associated region.

- 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://YOUR_REGION.api.cognitive.microsoft.com/sts/v1.0/issueToken -Headers $FetchTokenHeader
$OAuthToken
```

- cURL

```
curl -v -X POST "https://YOUR_REGION.api.cognitive.microsoft.com/sts/v1.0/issueToken" -H "Ocp-Apim-
Subscription-Key: YOUR_SUBSCRIPTION_KEY" -H "Content-type: application/x-www-form-urlencoded" -H
"Content-Length: 0"
```

If you entered a valid subscription key, the command returns an authorization token, otherwise an error is

returned.

Validate an authorization token

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.

- PowerShell

```
$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
```

- cURL

```
curl -v -X POST
"https://YOUR_REGION.stt.speech.microsoft.com/speech/recognition/interactive/cognitiveservices/v1?
language=en-US" -H "Authorization: Bearer YOUR_ACCESS_TOKEN" -H "Transfer-Encoding: chunked" -H
"Content-type: audio/wav; codec=audio/pcm; samplerate=16000" --data-binary @YOUR_AUDIO_FILE
```

If you entered a valid authorization token, the command returns the transcription for your audio file, otherwise an error is returned.

Error: 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`.

Error: 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.

"RecognitionStatus" in the response is "InitialSilenceTimeout"

This issue usually is caused by audio data. You might see this error because:

- There's a long stretch of silence at the beginning of the audio. 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.

Next steps

- Review the release notes

Speech Service release notes

11/23/2021 • 63 minutes to read • Edit Online

Speech SDK 1.19.0: 2021-Nov release

Note: Get started with the Speech SDK here. The Speech SDK on Windows depends on the shared Microsoft Visual C++ Redistributable for Visual Studio. Download it here.

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. See the documentation for more details.
- We have 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. Please migrate 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.
- **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.

Text-to-speech 2021-October release

New languages and voices addes for Neural TTS

- **49 new languages and variants introduced** - 98 new voices in 49 new locales are added into the neural TTS languages list: Adri in `af-ZA` Afrikaans (South Africa), Willem in `af-ZA` Afrikaans (South Africa), Mekdes in `am-ET` Amharic (Ethiopia), Ameha in `am-ET` Amharic (Ethiopia), Fatima in `ar-AE` Arabic (United Arab Emirates), Hamdan in `ar-AE` Arabic (United Arab Emirates), Laila in `ar-BH` Arabic (Bahrain), Ali in `ar-BH` Arabic (Bahrain), Amina in `ar-DZ` Arabic (Algeria), Ismael in `ar-DZ` Arabic (Algeria), Rana in `ar-IQ` Arabic (Iraq), Bassel in `ar-IQ` Arabic (Iraq), Sana in `ar-JO` Arabic (Jordan), Taim in `ar-JO` Arabic (Jordan), Noura in `ar-KW` Arabic (Kuwait), Fahed in `ar-KW` Arabic (Kuwait), Iman in `ar-LY` Arabic (Libya), Omar in `ar-LY` Arabic (Libya), Mouna in `ar-MA` Arabic (Morocco), Jamal in `ar-MA` Arabic (Morocco), Amal in `ar-QA` Arabic (Qatar), Moaz in `ar-QA` Arabic (Qatar), Amany in `ar-SY` Arabic (Syria), Laith in `ar-SY` Arabic (Syria), Reem in `ar-TN` Arabic (Tunisia), Hedi in `ar-TN` Arabic (Tunisia), Maryam in `ar-YE` Arabic (Yemen), Saleh in `ar-YE` Arabic (Yemen), Nabanita in `bn-BD` Bangla (Bangladesh), Pradeep in `bn-BD` Bangla (Bangladesh), Asilia in `en-KE` English (Kenya), Chilemba in `en-KE` English (Kenya), Ezinne in `en-NG` English (Nigeria), Abeo in `en-NG` English (Nigeria), Imani in `en-TZ` English (Tanzania), Elimu in `en-TZ` English (Tanzania), Sofia in `es-BO` Spanish (Bolivia), Marcelo in `es-BO` Spanish (Bolivia), Catalina in `es-CL` Spanish (Chile), Lorenzo in `es-CL` Spanish (Chile), Maria in `es-CR` Spanish (Costa Rica), Juan in `es-CR` Spanish (Costa Rica), Belkys in `es-CU` Spanish (Cuba), Manuel in `es-CU` Spanish (Cuba), Ramona in `es-DO` Spanish (Dominican Republic), Emilio in `es-DO` Spanish (Dominican Republic), Andrea in `es-EC` Spanish (Ecuador), Luis in `es-EC` Spanish (Ecuador), Teresa in `es-GQ` Spanish (Equatorial Guinea), Javier in `es-GQ` Spanish (Equatorial Guinea), Marta in `es-GT` Spanish (Guatemala), Andres in `es-GT` Spanish (Guatemala), Karla in `es-HN` Spanish (Honduras), Carlos in `es-HN` Spanish (Honduras), Yolanda in `es-NI` Spanish (Nicaragua), Federico in `es-NI` Spanish (Nicaragua), Margarita in `es-PA` Spanish (Panama), Roberto in `es-PA` Spanish (Panama), Camila in `es-PE` Spanish (Peru), Alex in `es-PE` Spanish (Peru), Karina in `es-PR` Spanish (Puerto Rico), Victor in `es-PR` Spanish (Puerto Rico), Tania in `es-PY` Spanish (Paraguay), Mario in `es-PY` Spanish (Paraguay), Lorena in `es-SV` Spanish (El Salvador), Rodrigo in `es-SV` Spanish (El Salvador), Valentina in `es-UY` Spanish (Uruguay), Mateo in `es-UY` Spanish (Uruguay), Paola in `es-VE` Spanish (Venezuela), Sebastian in `es-VE` Spanish (Venezuela), Dilara in `fa-IR` Persian (Iran), Farid in `fa-IR` Persian (Iran), Blessica in `fil-PH` Filipino (Philippines), Angelo in `fil-PH` Filipino (Philippines), Sabela in `gl-ES` Galician (Spain), Roi in `gl-ES` Galician (Spain), Siti in `jv-ID` Javanese (Indonesia), Dimas in `jv-ID` Javanese (Indonesia), Sreymom in `km-KH` Khmer (Cambodia), Piseth in `km-KH` Khmer (Cambodia), Nilar in `my-MM` Burmese (Myanmar), Thiha in `my-MM` Burmese (Myanmar), Ubax in `so-SO` Somali (Somalia), Muuse in `so-SO` Somali (Somalia), Tuti in `su-ID` Sundanese (Indonesia), Jajang in `su-ID` Sundanese (Indonesia), Rehema in `sw-TZ` Swahili (Tanzania), Daudi in `sw-TZ` Swahili (Tanzania), Saranya in `ta-LK` Tamil (Sri Lanka), Kumar in `ta-LK` Tamil (Sri Lanka), Venba in `ta-SG` Tamil (Singapore), Anbu in `ta-SG` Tamil (Singapore), Gul in `ur-IN` Urdu (India), Salman in `ur-IN` Urdu (India), Madina in `uz-UZ` Uzbek (Uzbekistan), Sardor in `uz-UZ` Uzbek (Uzbekistan), Thando in `zu-ZA` Zulu (South Africa), Themba in `zu-ZA` Zulu (South Africa).

Text-to-speech 2021-September release

- **New chatbot voice in `en-US` English (US)**: Sara, represents a young female adult that talks more casually and fits best for the chatbot scenarios.
- **New styles added for `ja-JP` Japanese voice Nanami**: Three new styles are now available with Nanami: chat, customer service, and cheerful.
- **Overal pronunciation improvement**: Ardi in `id-ID`, Premwadee in `th-TH`, Christel in `da-DK`, HoaiMy

and NamMinh in `vi-VN`.

- **Two new voices in `zh-CN` Chinese (Mandarin, China) in preview**: Xiaochen & Xiaoyan, optimized for spontaneous speech and customer service scenarios.

Text-to-speech 2021-July release

Neural TTS updates

- Reduced pronunciation errors in Hebrew by 20%.

Speech Studio updates

- **Custom Neural Voice**: Updated the training pipeline to UniTTSv3 with which the model quality is improved while training time is reduced by 50% for acoustic models.
- **Audio Content Creation**: Fixed the "Export" performance issue and the bug on custom voice selection.

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. In conjunction with Azure DevOps and GitHub, we will drop support for 16.04 in September 2021. Please 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.

Text-to-speech 2021-June release

Speech Studio updates

- **Custom Neural Voice**: Custom Neural Voice training extended to support South East Asia. New features released to suport data uploading status checking.
- **Audio Content Creation**: Released a new feature to support custom lexicon. With this feature, users can easily create their lexicon files and define the customized pronunciation for their audio output.

Text-to-speech 2021-May release

New languages and voices added for neural TTS

- **Ten new languages introduced** - 20 new voices in 10 new locales are added into the neural TTS language list: Yan in `en-HK` English (Hongkong), Sam in `en-HK` English (Hongkong), Molly in `en-NZ` English (New Zealand), Mitchell in `en-NZ` English (New Zealand), Luna in `en-SG` English (Singapore), Wayne in `en-SG` English (Singapore), Leah in `en-ZA` English (South Africa), Luke in `en-ZA` English (South Africa), Dhwani in `gu-IN` Gujarati (India), Niranjan in `gu-IN` Gujarati (India), Aarohi in `mr-IN` Marathi (India), Manohar in `mr-IN` Marathi (India), Elena in `es-AR` Spanish (Argentina), Tomas in `es-AR` Spanish (Argentina), Salome in `es-CO` Spanish (Colombia), Gonzalo in `es-CO` Spanish (Colombia), Paloma in `es-US` Spanish (US), Alonso in `es-US` Spanish (US), Zuri in `sw-KE` Swahili (Kenya), Rafiki in `sw-KE` Swahili (Kenya).

- **Eleven new en-US voices in preview** - 11 new en-US voices in preview are added to American English, they are Ashley, Amber, Ana, Brandon, Christopher, Cora, Elizabeth, Eric, Michelle, Monica, Jacob.

- **Five `zh-CN` Chinese (Mandarin, Simplified) voices are generally available** - 5 Chinese (Mandarin, Simplified) voices are changed from preview to generally available. They are Yunxi, Xiaomo, Xiaoman, Xiaoxuan, Xiaorui. Now, these voices are available in all regions. Yunxi is added with a new 'assistant' style, which is suitable for chat bot and voice agent. Xiaomo's voice styles are refined to be more natural and featured.

Speech SDK 1.17.0: 2021-May release

> **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 Single-Shot/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 are 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 very 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 Java Speech SDK 1.16 crashes when using DialogServiceConnector without a network connection or an invalid subscription key.
- Fixed a crash when abruptly stopping speech recognition (e.g. using CTRL+C on console app).
- **Java**: Added a fix to delete temporary files on Windows when using Java Speech SDK.
- **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 is not 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 is not 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 CLI (also known as SPX): 2021-May release

> **NOTE**
>
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- SPX now supports Profile, Speaker ID and Speaker verification - Try `spx profile` and `spx speaker` from the SPX command line.
- We also added Dialog support - Try `spx dialog` from the SPX command line.
- SPX help improvements. Please give us feedback about how this works for you by opening a GitHub issue.
- We've decreased the size of the SPX .NET tool install.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-April release

Neural TTS is available across 21 regions

- **Twelve new regions added** - Neural TTS is now available in these new 12 regions: `Japan East`, `Japan West`, `Korea Central`, `North Central US`, `North Europe`, `South Central US`, `Southeast Asia`, `UK South`, `west Central US`, `West Europe`, `West US`, `West US 2`. Check here for full list of 21 supported regions.

Text-to-speech 2021-March release

New languages and voices added for neural TTS

- **Six new languages introduced** - 12 new voices in 6 new locales are added into the neural TTS language list: Nia in `cy-GB` Welsh (United Kingdom), Aled in `cy-GB` Welsh (United Kingdom), Rosa in `en-PH` English (Philippines), James in `en-PH` English (Philippines), Charline in `fr-BE` French (Belgium), Gerard in `fr-BE` French (Belgium), Dena in `nl-BE` Dutch (Belgium), Arnaud in `nl-BE` Dutch (Belgium), Polina in `uk-UA` Ukrainian (Ukraine), Ostap in `uk-UA` Ukrainian (Ukraine), Uzma in `ur-PK` Urdu (Pakistan), Asad in `ur-PK` Urdu (Pakistan).
- **Five languages from preview to GA** - 10 voices in 5 locales introduced in 2020-November now are GA: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).
- **New male voice added for French (Canada)** - A new voice Antoine is available for `fr-CA` French (Canada).
- **Quality improvement** - Pronunciation error rate reduction on `hu-HU` Hungarian - 48.17%, `nb-NO` Norwegian - 52.76%, `nl-NL` Dutch (Netherlands) - 22.11%.

With this release, we now support a total of 142 neural voices across 60 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Get facial pose events to animate characters

Neural Text-to-speech now includes the viseme event. Viseme events allow users to get a sequence of facial poses along with synthesized speech. Visemes can be used to control the movement of 2D and 3D avatar models, matching mouth movements to synthesized speech. Viseme events are only available for `en-US-AriaNeural` voice at this time.

Add the bookmark element in Speech Synthesis Markup Language (SSML)

The bookmark element allows you to insert custom markers in SSML to get the offset of each marker in the audio stream. It can be used to reference a specific location in the text or tag sequence.

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 to use the SDK in environments that do not have required audio library dependencies.
- **Objective-C/Swift**: Added support for module framework with umbrella header. This allows 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` cannot 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 CLI (also known as SPX): 2021-March release

> **NOTE**
> Get started with the Azure Speech service command line interface (CLI) here. The CLI enables you to use the Azure Speech service without writing any code.

New features

- Added `spx intent` command for intent recognition, replacing `spx recognize intent`.
- Recognize and intent can now use Azure functions to calculate word error rate using `spx recognize --wer url <URL>`.
- Recognize can now output results as VTT files using `spx recognize --output vtt file <FILENAME>`.
- Sensitive key info now obscured in debug/verbose output.
- Added URL checking and error message for content field in batch transcription create.

COVID-19 abridged testing:

As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2021-February release

Custom Neural Voice GA

Custom Neural Voice is GA in February in 13 languages: Chinese (Mandarin, Simplified), English (Australia), English (India), English (United Kingdom), English (United States), French (Canada), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Mexico), and Spanish (Spain). Learn more about what is Custom Neural Voice and how to use it responsibly. Custom Neural Voice feature requires registration and Microsoft may limit access based on Microsoft's eligibility criteria. Learn more about the limited access.

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.
- Your voice assistant or bot 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 have 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 are not yet seeing a size improvements.
  - Linux libraries are 20-25% smaller.
  - Android libraries are 3-5% smaller.

New features

- **All**: New 48KHz 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 public 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, e.g. 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 (e.g. 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 programing 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 pronunciation 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` did not 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 re-usage.
- **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 has not 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 CLI (also known as SPX): 2021-January release

New features

- Speech CLI is now available as a NuGet package and can be installed via .NET CLI as a .NET global tool you can call from the shell/command line.
- The Custom Speech DevOps Template repo has been updated to use Speech CLI for its Custom Speech workflows.

COVID-19 abridged testing: As the ongoing pandemic continues to require our engineers to work from home, pre-pandemic manual verification scripts have been significantly reduced. We test on fewer devices with fewer configurations, and the likelihood of environment-specific bugs slipping through may be increased. We

still rigorously validate with a large set of automation. In the unlikely event that we missed something, please let us know on GitHub.
Stay healthy!

Text-to-speech 2020-December release

New neural voices in GA and preview

Released 51 new voices for a total of 129 neural voices across 54 languages/locales:

- **46 new voices in GA locales**: Shakir in `ar-EG` Arabic (Egypt), Hamed in `ar-SA` Arabic (Saudi Arabia), Borislav in `bg-BG` Bulgarian (Bulgaria), Joana in `ca-ES` Catalan (Spain), Antonin in `cs-CZ` Czech (Czech Republic), Jeppe in `da-DK` Danish (Denmark), Jonas in `de-AT` German (Austria), Jan in `de-CH` German (Switzerland), Nestoras in `el-GR` Greek (Greece), Liam in `en-CA` English (Canada), Connor in `en-IE` English (Ireland), Madhur in `en-IN` Hindi (India), Mohan in `en-IN` Telugu (India), Prabhat in `en-IN` English (India), Valluvar in `en-IN` Tamil (India), Enric in `es-ES` Catalan (Spain), Kert in `et-EE` Estonian (Estonia), Harri in `fi-FI` Finnish (Finland), Selma in `fi-FI` Finnish (Finland), Fabrice in `fr-CH` French (Switzerland), Colm in `ga-IE` Irish (Ireland), Avri in `he-IL` Hebrew (Israel), Srecko in `hr-HR` Croatian (Croatia), Tamas in `hu-HU` Hungarian (Hungary), Gadis in `id-ID` Indonesian (Indonesia), Leonas in `lt-LT` Lithuanian (Lithuania), Nils in `lv-LV` Latvian (Latvia), Osman in `ms-MY` Malay (Malaysia), Joseph in `mt-MT` Maltese (Malta), Finn in `nb-NO` Norwegian, Bokmål (Norway), Pernille in `nb-NO` Norwegian, Bokmål (Norway), Fenna in `nl-NL` Dutch (Netherlands), Maarten in `nl-NL` Dutch (Netherlands), Agnieszka in `pl-PL` Polish (Poland), Marek in `pl-PL` Polish (Poland), Duarte in `pt-BR` Portuguese (Brazil), Raquel in `pt-PT` Portuguese (Potugal), Emil in `ro-RO` Romanian (Romania), Dmitry in `ru-RU` Russian (Russia), Svetlana in `ru-RU` Russian (Russia), Lukas in `sk-SK` Slovak (Slovakia), Rok in `sl-SI` Slovenian (Slovenia), Mattias in `sv-SE` Swedish (Sweden), Sofie in `sv-SE` Swedish (Sweden), Niwat in `th-TH` Thai (Thailand), Ahmet in `tr-TR` Turkish (Turkey), NamMinh in `vi-VN` Vietnamese (Vietnam), HsiaoChen in `zh-TW` Taiwanese Mandarin (Taiwan), YunJhe in `zh-TW` Taiwanese Mandarin (Taiwan), HiuMaan in `zh-HK` Chinese Cantonese (Hong Kong), WanLung in `zh-HK` Chinese Cantonese (Hong Kong).
- **5 new voices in preview locales**: Kert in `et-EE` Estonian (Estonia), Colm in `ga-IE` Irish (Ireland), Nils in `lv-LV` Latvian (Latvia), Leonas in `lt-LT` Lithuanian (Lithuania), Joseph in `mt-MT` Maltese (Malta).

With this release, we now support a total of 129 neural voices across 54 languages/locales. In addition, over 70 standard voices are available in 49 languages/locales. Visit Language support for the full list.

Updates for Audio Content Creation

- Improved voice selection UI with voice categories and detailed voice descriptions.
- Enabled intonation tuning for all neural voices across different languages.
- Automated the UI localization based on the language of the browser.
- Enabled `StyleDegree` controls for all `zh-CN` Neural voices. Visit the Audio Content Creation tool to check out the new features.

Updates for zh-CN voices

- Updated all `zh-CN` neural voices to support English speaking.
- Enabled all `zh-CN` neural voices to support intonation adjustment. SSML or Audio Content Creation tool can be used to adjust for the best intonation.
- Updated all `zh-CN` multi-style neural voices to support `StyleDegree` control. Emotion intensity (soft or strong) is adjustable.
- Updated `zh-CN-YunyeNeural` to support multiple styles which can perform different emotions.

Text-to-speech 2020-November release

New locales and voices in preview

- **Five new voices and languages** are introduced to the Neural TTS portfolio. They are: Grace in Maltese (Malta), Ona in Lithuanian (Lithuania), Anu in Estonian (Estonia), Orla in Irish (Ireland) and Everita in Latvian (Latvia).
- **Five new `zh-CN` voices with multiple styles and roles support**: Xiaohan, Xiaomo, Xiaorui, Xiaoxuan and Yunxi.

> These voices are available in public preview in three Azure regions: EastUS, SouthEastAsia and WestEurope.

Neural TTS Container GA

- With Neural TTS Container, developers can run speech synthesis with the most natural digital voices in their own environment for specific security and data governance requirements. Check how to install Speech Containers.

New features

- **Custom Voice**: enabled users to copy a voice model from one region to another; supported endpoint suspension and resuming. Go to the portal here.
- SSML silence tag support.
- General TTS voice quality improvements: Improved word-level pronunciation accuracy in nb-NO. Reduced 53% pronunciation error.

> Read more at this tech blog.

Text-to-speech 2020-October release

New features

- Jenny supports a new `newscast` style. See how to use the speaking styles in SSML.
- **Neural voices upgraded to HiFiNet vocoder, with higher audio fidelity and faster synthesis speed**. This benefits customers whose scenario relies on hi-fi audio or long interactions, including video dubbing, audio books, or online education materials. Read more about the story and hear the voice samples on our tech community blog
- **Custom Voice & Audio Content Creation Studio localized to 17 locales**. Users can easily switch the UI to a local language for a more friendly experience.
- **Audio Content Creation**: Added style degree control for XiaoxiaoNeural; Refined the customized break feature to include incremental breaks of 50ms.

General TTS voice quality improvements

- Improved word-level pronunciation accuracy in `pl-PL` (error rate reduction: 51%) and `fi-FI` (error rate reduction: 58%)
- Improved `ja-JP` single word reading for the dictionary scenario. Reduced pronunciation error by 80%.
- `zh-CN-XiaoxiaoNeural`: Improved sentiment/CustomerService/Newscast/Cheerful/Angry style voice quality.
- `zh-CN`: Improved Erhua pronunciation and light tone and refined space prosody, which greatly improves intelligibility.

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 co-existence 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 is not 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 CLI (also known as SPX): 2020-October release

SPX is the command line interface to use the Azure Speech service without writing code. Download the latest version here.

New features

- `spx csr dataset upload --kind audio|language|acoustic` – create datasets from local data, not just from URLs.
- `spx csr evaluation create|status|list|update|delete` – compare new models against baseline truth/other models.
- `spx * list` – supports non-paged experience (doesn't require --top X --skip X).
- `spx * --http header A=B` – support custom headers (added for Office for custom authentication).
- `spx help` – improved text and back-tick text color coded (blue).

Text-to-speech 2020-September release

New features

- **Neural TTS**
  - **Extended to support 18 new languages/locales.** They are Bulgarian, Czech, German (Austria), German (Switzerland), Greek, English (Ireland), French (Switzerland), Hebrew, Croatian, Hungarian, Indonesian, Malay, Romanian, Slovak, Slovenian, Tamil, Telugu and Vietnamese.
  - **Released 14 new voices to enrich the variety in the existing languages.** See full language and voice list.
  - **New speaking styles for `en-US` and `zh-CN` voices.** Jenny, the new voice in English (US), supports chatbot, customer service, and assistant styles. 10 new speaking styles are available with our zh-CN voice, XiaoXiao. In addition, the XiaoXiao neural voice supports `StyleDegree` tuning. See how to use the speaking styles in SSML.
- **Containers: Neural TTS Container released in public preview with 16 voices available in 14 languages.** Learn more on how to deploy Speech Containers for Neural TTS

Read the full announcement of the TTS updates for Ignite 2020

Text-to-speech 2020-August release

New features

- **Neural TTS: new speaking style for `en-US` Aria voice.** AriaNeural can sound like a news caster when reading news. The 'newscast-formal' style sounds more serious, while the 'newscast-casual' style is more relaxed and informal. See how to use the speaking styles in SSML.
- **Custom Voice: a new feature is released to automatically check training data quality.** When you upload your data, the system will examine various aspects of your audio and transcript data, and automatically fix or filter issues to improve the quality of the voice model. This covers the volume of your audio, the noise level, the pronunciation accuracy of speech, the alignment of speech with the normalized text, silence in the audio, in addition to the audio and script format.
- **Audio Content Creation: a set of new features to enable more powerful voice tuning and audio management capabilities.**
  - Pronunciation: the pronunciation tuning feature is updated to the latest phoneme set. You can pick the right phoneme element from the library and refine the pronunciation of the words you have selected.
  - Download: The audio "Download"/"Export" feature is enhanced to support generating audio by paragraph. You can edit content in the same file/SSML, while generating multiple audio outputs. The file structure of "Download" is refined as well. Now, you can easily get all audio files in one folder.
  - Task status : The multi-file export experience is improved. When you export multiple files in the past, if one of the files has failed, the entire task will fail. But now, all other files will be successfully exported. The task report is enriched with more detailed and structured information. You can check the logs for all failed files and sentences now with the report.
  - SSML documentation: linked to SSML document to help you check the rules for how to use all tuning features.
- **The Voice List API is updated to include a user-friendly display name and the speaking styles supported for neural voices.**

General TTS voice quality improvements

- Reduced word-level pronunciation error % for `ru-RU` (errors reduced by 56%) and `sv-SE` (errors reduced by 49%)
- Improved polyphony word reading on `en-US` neural voices by 40%. Examples of polyphony words include "read", "live", "content", "record", "object", etc.
- Improved the naturalness of the question tone in `fr-FR`. MOS (Mean Opinion Score) gain: +0.28
- Updated the vocoders for the following voices, with fidelity improvements and overall performance speed-up by 40%.

LOCALE	VOICE
`en-GB`	Mia
`es-MX`	Dalia
`fr-CA`	Sylvie
`fr-FR`	Denise

LOCALE	VOICE
`ja-JP`	Nanami
`ko-KR`	Sun-Hi

Bug fixes

- Fixed a number of bugs with the Audio Content Creation tool
  - Fixed issue with auto refreshing.
  - Fixed issues with voice styles in zh-CN in the South East Asia region.
  - Fixed stability issue, including an export error with the 'break' tag, and errors in punctuation.

New speech-to-text locales: 2020-August Release

Speech-to-text released 26 new locales in August: 2 European languages `cs-CZ` and `hu-HU`, 5 English locales and 19 Spanish locales that cover most South American countries. Below is a list of the new locales. See the complete language list here.

LOCALE	LANGUAGE
`cs-CZ`	Czech (Czech Republic)
`en-HK`	English (Hong Kong)
`en-IE`	English (Ireland)
`en-PH`	English (Philippines)
`en-SG`	English (Singapore)
`en-ZA`	English (South Africa)
`es-AR`	Spanish (Argentina)
`es-BO`	Spanish (Bolivia)
`es-CL`	Spanish (Chile)
`es-CO`	Spanish (Colombia)
`es-CR`	Spanish (Costa Rica)
`es-CU`	Spanish (Cuba)
`es-DO`	Spanish (Dominican Republic)
`es-EC`	Spanish (Ecuador)
`es-GT`	Spanish (Guatemala)

LOCALE	LANGUAGE
`es-HN`	Spanish (Honduras)
`es-NI`	Spanish (Nicaragua)
`es-PA`	Spanish (Panama)
`es-PE`	Spanish (Peru)
`es-PR`	Spanish (Puerto Rico)
`es-PY`	Spanish (Paraguay)
`es-SV`	Spanish (El Salvador)
`es-US`	Spanish (USA)
`es-UY`	Spanish (Uruguay)
`es-VE`	Spanish (Venezuela)
`hu-HU`	Hungarian (Hungary)

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 does not 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 did not 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!

Text-to-speech 2020-July release

New features

- **Neural TTS, 15 new neural voices**: The new voices added to the Neural TTS portfolio are Salma in `ar-EG` Arabic (Egypt), Zariyah in `ar-SA` Arabic (Saudi Arabia), Alba in `ca-ES` Catalan (Spain), Christel in `da-DK` Danish (Denmark), Neerja in `es-IN` English (India), Noora in `fi-FI` Finnish (Finland), Swara in `hi-IN` Hindi (India), Colette in `nl-NL` Dutch (Netherlands), Zofia in `pl-PL` Polish (Poland), Fernanda in `pt-PT` Portuguese (Portugal), Dariya in `ru-RU` Russian (Russia), Hillevi in `sv-SE` Swedish (Sweden), Achara in `th-TH` Thai (Thailand), HiuGaai in `zh-HK` Chinese (Cantonese, Traditional) and HsiaoYu in `zh-TW` Chinese (Taiwanese Mandarin). Check all supported languages.

- **Custom Voice, streamlined voice testing with the training flow to simplify user experience**: With the new testing feature, each voice will be automatically tested with a predefined test set optimized for each language to cover general and voice assistant scenarios. These test sets are carefully selected and tested to include typical use cases and phonemes in the language. Besides, users can still select to upload their own test scripts when training a model.

- **Audio Content Creation: a set of new features is released to enable more powerful voice tuning and audio management capabilities**
  - `Pitch`, `rate`, and `volume` are enhanced to support tuning with a predefined value, like slow, medium and fast. It's now straightforward for users to pick a 'constant' value for their audio editing.

- Users can now review the `Audio history` for their work file. With this feature, users can easily track all the generated audio related to a working file. They can check the history version and compare the quality while tuning at the same time.

- The `Clear` feature is now more flexible. Users can clear a specific tuning parameter while keeping other parameters available for the selected content.
- A tutorial video was added on the landing page to help users quickly get started with TTS voice tuning and audio management.

General TTS voice quality improvements

- Improved TTS vocoder in for higher fidelity and lower latency.
  - Updated Elsa in `it-IT` to a new vocoder which achieved +0.464 CMOS (Comparative Mean Opinion Score) gain in voice quality, 40% faster in synthesis and 30% reduction on first byte latency.
  - Updated Xiaoxiao in `zh-CN` to the new vocoder with +0148 CMOS gain for the general domain, +0.348 for the newscast style and +0.195 for the lyrical style.
- Updated `de-DE` and `ja-JP` voice models to make the TTS output more natural.
  - Updated Katja in `de-DE` with the latest prosody modeling method, the MOS (Mean Opinion Score) gain is +0.13.
  - Updated Nanami in `ja-JP` with a new pitch accent prosody model, the MOS (Mean Opinion Score) gain is +0.19;
- Improved word-level pronunciation accuracy in five languages.

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`en-GB`	51%
`ko-KR`	17%

LANGUAGE	PRONUNCIATION ERROR REDUCTION
`pt-BR`	39%
`pt-PT`	77%
`id-ID`	46%

Bug fixes

- Currency reading
  - Fixed the issue with currency reading for `es-ES` and `es-MX`

LANGUAGE	INPUT	READOUT AFTER IMPROVEMENT
`es-MX`	$1.58	un peso cincuenta y ocho centavos
`es-ES`	$1.58	un dólar cincuenta y ocho centavos

  - Support for negative currency (like "-325€") in following locales: `en-US`, `en-GB`, `fr-FR`, `it-IT`, `en-AU`, `en-CA`.
- Improved address reading in `pt-PT`.
- Fixed Natasha (`en-AU`) and Libby (`en-UK`) pronunciation issues on the word "for" and "four".
- Fixed bugs on Audio Content Creation tool
  - The additional and unexpected pause after the second paragraph is fixed.
  - 'No break' feature is added back from a regression bug.
  - The random refresh issue of Speech Studio is fixed.

Samples/SDK

- JavaScript: Fixes playback issue in Firefox, and Safari on macOS and iOS.

Speech SDK 1.12.1: 2020-June release

Speech CLI (also known as SPX)

- Added in-CLI help search features:
  - `spx help find --text TEXT`
  - `spx help find --topic NAME`
- Updated to work with newly deployed v3.0 Batch and Custom Speech APIs:
  - `spx help batch examples`
  - `spx help csr examples`

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 was not 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

Speech CLI (Also Know As SPX)

- **SPX** is a new command line tool that allows you to perform recognition, synthesis, translation, batch transcription, and custom speech management from the command line. Use it to test the Speech Service, or to script the Speech Service tasks you need to perform. Download the tool and read the documentation here.

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 was not set properly when recognizing intents from simple phrases.
- Fixed an issue where `SpeechRecognitionEventArgs` result offset was not 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 API's 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 800ms to 100ms 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, please 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 did not 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 could not 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` has not been installed to the `/usr/lib/ssl` directory in the system. Please check our documentation under Speech SDK docs to work around the issue.
- We have 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-prem 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 is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see here.
- Conversation transcription functionality is available through the SDK. See here.
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the text-to-speech overview.
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details here).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.

- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details here).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing audio input device selection. This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the GitHub sample repository. This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our Unity quickstart.
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our sample repository:

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the Batch Transcription REST API.

Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at turn.end, not at speech.end.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see here](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on GitHub.
  - We now support Node.js, more info can be found here.
  - The length restriction for audio sessions has been removed, reconnection will happen automatically

under the cover.

- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; RecognizedIntent was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the sample repository.

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in AddIntent in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the sample repository.

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error 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 do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- SpeechOutputFormat in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in

the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

  When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

What is the Speech CLI?

11/23/2021 • 2 minutes to read • Edit Online

The Speech CLI is a command-line tool for using the Speech service without writing any code. The Speech CLI requires minimal setup, and it's easy to immediately start experimenting with key features of the Speech service to see if your use-cases can be met. Within minutes, you can run simple test workflows like 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 can be scaled up to run larger processes 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. Consider the following guidance to decide when to use the Speech CLI or the Speech SDK.

Use the Speech CLI when:

- You want to experiment with Speech service features with minimal setup and no code
- You have relatively simple requirements for a production application using the Speech service

Use the Speech SDK when:

- You want to integrate Speech service functionality within a specific language or platform (for example, C#, Python, C++)
- You have complex requirements that may require advanced service requests, or 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 using either input from text files, or input directly from the command line. Customize speech output characteristics using SSML configurations, and either standard or neural voices.
- 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 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, and also shows slightly more advanced commands for running batch operations for speech-to-text and text-to-speech. After reading the basics article, you should have enough of an understanding of the syntax to start writing some custom commands, or automating simple Speech service operations.

Next steps

- Get started with the Speech CLI quickstart
- Configure your data store
- Learn how to run batch operations with the Speech CLI

Get started with the Azure Speech CLI

11/23/2021 • 8 minutes to read • Edit Online

In this article, you'll learn how to use the Azure Speech CLI (command-line interface) to access Speech services like speech to text, text to speech, and speech translation without writing code. The Speech CLI is production ready and can be used to automate simple workflows in the Speech service, using `.bat` or shell scripts.

This article assumes that you have working knowledge of the command prompt, 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 Install
- Linux Install
- Docker Install (Windows, Linux, macOS)

Follow these steps to install the Speech CLI on Windows:

1. On Windows, you need the Microsoft Visual C++ Redistributable for Visual Studio 2019 for your platform. Installing this for the first time may require a restart.
2. Install .NET Core 3.1 SDK.
3. Install the Speech CLI using NuGet by entering this command:

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

Type `spx` to see help for the Speech CLI.

> **NOTE**
> As an alternative to NuGet, you can download and extract the Speech CLI for Windows as a zip file.

Font limitations

On Windows, the Speech CLI can only show fonts available to the command prompt on the local computer. Windows Terminal supports all fonts produced interactively by the Speech CLI.

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

Create subscription config

- Terminal
- PowerShell

You need an Azure subscription key and region identifier (ex. `eastus`, `westus`) to get started. See the Speech

service overview documentation for steps to get these credentials.

You run the following commands in a terminal to configure your subscription key and region identifier.

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

The key and region are stored for future Speech CLI commands. Run the following commands to view the current configuration.

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

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

```
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 built in to the tool by running the following command.

```
spx
```

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

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

Run the following command to see options for the recognize command:

```
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)

You run this command to convert speech to text (speech recognition) using your system's default microphone.

```
spx recognize --microphone
```

After entering the command, SPX will begin listening for audio on the current active input device, and stop when you press **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.

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

> **NOTE**
> If you are 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 in the previous step.

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

Text to speech (speech synthesis)

Running the following command will take text as input, and output the synthesized speech to the current active output device (for example, your computer speakers).

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

You can also save the synthesized output to file. In this example, we'll create a file named `my-sample.wav` in the directory that the command is run.

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

These examples presume that you're testing in English. However, we support speech synthesis in many languages. You can pull down a full list of voices with this command, or by visiting the language support page.

```
spx synthesize --voices
```

Here's how you use one of the voices you've discovered.

```
spx synthesize --text "Bienvenue chez moi." --voice fr-CA-Caroline --speakers
```

> **TIP**
> If you get stuck or want to learn more about the Speech CLI's 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 this 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.

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

When translating into multiple languages, separate language codes with `;`.

```
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.

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

> **NOTE**
>
> See the language and locale article for a list of all supported languages with their corresponding locale codes.

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

Next steps

- Install GStreamer to use Speech CLI with MP3 and other formats
- Speech CLI configuration options
- Batch operations with the Speech CLI

Speech CLI configuration options

11/23/2021 • 2 minutes to read • Edit Online

Speech CLI's behavior 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.

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

Create and manage configuration files in the datastore

This section shows how to use a configuration file in the local datastore to store and fetch command settings using `spx config`, and store output from Speech CLI using the `--output` option.

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`.

```
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
```

You can also write dynamic content to a configuration file. 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.

```
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.

```
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
```

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:

```
spx help advanced setup
```

Next steps

- Batch operations with the Speech CLI

Speech CLI batch operations

11/23/2021 • 2 minutes to read • Edit Online

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.

```
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.

```
audio.input.id recognizer.session.started.sessionid recognizer.recognized.result.text
sample_1 07baa2f8d9fd4fbcb9faea451ce05475 A sample wave file.
sample_2 8f9b378f6d0b42f99522f1173492f013 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.

```
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.

```
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:

```
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.

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

Next steps

- Speech CLI overview
- Speech CLI quickstart

What is Speech Studio?

11/23/2021 • 2 minutes to read • Edit Online

Speech Studio is a set of UI-based tools for building and integrating features from Azure Speech service in your applications. You create projects in Speech Studio using a no-code approach, and then reference the assets you create in your applications using the Speech SDK, Speech CLI, or various REST APIs.

Set up your Azure account

You need to have an Azure account and add a Speech service resource before you can use Speech Studio. If you don't have an account and resource, try the Speech service for free.

After you create an Azure account and a Speech service resource:

1. Sign in to the Speech Studio with your Azure account.
2. Select the Speech service resource you need to get started. (You can change the resources anytime in "Settings" in the top menu.)

Speech Studio features

The following Speech service features are available as project types in Speech Studio.

- **Real-time speech-to-text**: Quickly test speech-to-text by dragging and dropping audio files without using any code. This is a demo tool for seeing how speech-to-text works on your audio samples, but see the overview for speech-to-text to explore the full functionality that's available.
- **Custom Speech**: Custom Speech allows you to create speech recognition models that are tailored to specific vocabulary sets and styles of speaking. In contrast to using a base speech recognition model, Custom Speech models become part of your unique competitive advantage because they are not publicly accessible. See the quickstart to get started with uploading sample audio to create a Custom Speech model.
- **Pronunciation Assessment**: Pronunciation assessment evaluates speech pronunciation and gives speakers feedback on the accuracy and fluency of spoken audio. Speech Studio provides a sandbox for testing this feature quickly with no code, but see the how-to article for using the feature with the Speech SDK in your applications.
- **Voice Gallery**: Build apps and services that speak naturally. Choose from more than 170 voices in over 70 languages and variants. Bring your scenarios to life with highly expressive and humanlike neural voices.
- **Custom Voice**: Custom Voice allows you to 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. See the how-to article on creating and using custom voices via endpoints.
- **Audio Content Creation**: Audio Content Creation is an easy-to-use tool that lets you build highly natural audio content for a variety of scenarios, like audiobooks, news broadcasts, video narrations, and chat bots. Speech Studio allows you to export your created audio files to use in your applications.
- **Custom Keyword**: A Custom Keyword is a word or short phrase that allows your product to be voice-activated. 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**: Custom Commands makes it easy to build rich voice commanding apps optimized for voice-first interaction experiences. It provides a code-free authoring experience in Speech Studio, an automatic hosting model, and relatively lower complexity, helping you focus on building the best solution for your voice commanding scenarios. See the how-to guide for building Custom Commands applications, and also see the guide for integrating your Custom Commands application with the Speech SDK.

Next steps

Explore Speech Studio and create a project.

Azure role-based access control in Speech Studio

11/23/2021 • 3 minutes to read • Edit Online

Speech Studio supports Azure role-based access control (Azure RBAC), an authorization system for managing individual access to Azure resources. Using Azure RBAC, you can assign different levels of permissions for your Speech Studio operations to different team members. For more information on Azure RBAC, see the Azure RBAC documentation.

Prerequisites

- You must be signed into Speech Studio with your Azure account and Speech resource. See the Speech Studio overview.

Manage role assignments for Speech resources

To grant access to an Azure speech resource, you add a role assignment through the Azure RBAC tool in the Azure portal.

Within a few minutes, the target will be assigned the selected role at the selected scope. For help with these steps, see Assign Azure roles using the Azure portal.

Supported built-in roles in Speech Studio

A role definition is a collection of permissions. Use the following recommended built-in roles if you don't have any unique custom requirements for permissions:

BUILT-IN ROLE	PERMISSION TO LIST RESOURCE KEYS	PERMISSION FOR CUSTOM SPEECH OPERATIONS	PERMISSION FOR CUSTOM VOICE OPERATIONS	PERMISSION FOR OTHER CAPABILITIES
Owner	Yes	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access
Contributor	Yes	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access
Cognitive Service Contributors	Yes	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access

BUILT-IN ROLE	PERMISSION TO LIST RESOURCE KEYS	PERMISSION FOR CUSTOM SPEECH OPERATIONS	PERMISSION FOR CUSTOM VOICE OPERATIONS	PERMISSION FOR OTHER CAPABILITIES
Cognitive Service Users	Yes	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access
Cognitive Service Speech Contributor	No	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access to the projects, including the permission to create, edit, or delete project / data / model / endpoints	Full access
Cognitive Service Speech User	No	Can view the projects / datasets / models / endpoints; cannot create, edit, delete	Can view the projects / datasets / models / endpoints; cannot create, edit, delete	Full access
Cognitive Services Data Reader (preview)	No	Can view the projects / datasets / models / endpoints; cannot create, edit, delete	Can view the projects / datasets / models / endpoints; cannot create, edit, delete	Full access

Alternatively, you can create your own custom roles. For example, you could create a custom role with the permission to upload custom speech datasets, but without the ability to deploy a custom speech model to an endpoint.

> **NOTE**
>
> Speech Studio supports key-based authentication. Roles that have permission to list resource keys (`Microsoft.CognitiveServices/accounts/listKeys/action`) will firstly be authenticated with a resource key and will have full access to the Speech Studio operations, as long as key authentication is enabled in Azure portal. If key authentication is disabled by the service admin, then those roles will lose all access to the Studio.

> **NOTE**
>
> One resource could be assigned or inherited with multiple roles, and the final level of access to this resource is a combination of all your roles' permissions from the operation level.

Next steps

Learn more about Speech service encryption of data at rest.

Azure Cognitive Services support and help options

11/23/2021 • 2 minutes to read • Edit Online

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

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 (preview)
- Personalizer

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 (preview)
- Personalizer

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 (preview)
- 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?

Swagger documentation

11/23/2021 • 2 minutes to read • Edit Online

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 v3.0 and v2.0 are 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-v3-0`
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.

Reference documents

- Swagger: Speech-to-text REST API v3.0
- Speech-to-text REST API
- Text-to-speech REST API

Next steps

- Speech service samples on GitHub.
- Get a Speech service subscription key for free

Speech Services in sovereign clouds

11/23/2021 • 3 minutes to read • Edit Online

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 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 Government have the following format:

REST API TYPE / OPERATION	ENDPOINT FORMAT
Access token	`https://<REGION_IDENTIFIER>.api.cognitive.microsoft.us/sts/v1.0/issu`
Speech-to-text REST API v3.0	`https://<REGION_IDENTIFIER>.api.cognitive.microsoft.us/<URL_PATH>`
Speech-to-text REST API for short audio	`https://<REGION_IDENTIFIER>.stt.speech.azure.us/<URL_PATH>`
Text-to-speech REST API	`https://<REGION_IDENTIFIER>.tts.speech.azure.us/<URL_PATH>`

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

	REGION IDENTIFIER
US Gov Arizona	`usgovarizona`
US Gov Virginia	`usgovvirginia`

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++
- Java
- Python
- Objective-C

```
var config = SpeechConfig.FromHost(usGovHost, subscriptionKey);
```

Speech CLI should be used like this (note the `--host` option):

```
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	`wss://usgovarizona.stt.speech.azure.us`
Text-to-Speech	`https://usgovarizona.tts.speech.azure.us`
US Gov Virginia	
Speech-to-text	`wss://usgovvirginia.stt.speech.azure.us`
Text-to-Speech	`https://usgovvirginia.tts.speech.azure.us`

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
  - 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	`https://<REGION_IDENTIFIER>.api.cognitive.azure.cn/sts/v1.0/issueTok`
Speech-to-text REST API v3.0	`https://<REGION_IDENTIFIER>.api.cognitive.azure.cn/<URL_PATH>`
Speech-to-text REST API for short audio	`https://<REGION_IDENTIFIER>.stt.speech.azure.cn/<URL_PATH>`
Text-to-speech REST API	`https://<REGION_IDENTIFIER>.tts.speech.azure.cn/<URL_PATH>`

Replace `<REGION_IDENTIFIER>` with the identifier matching the region of your subscription from this table:

	REGION IDENTIFIER
China East 2	`chinaeast2`
China North 2	`chinanorth2`

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++
- Java
- Python
- Objective-C

```
var config = SpeechConfig.FromHost(azCnHost, subscriptionKey);
```

Speech CLI should be used like this (note the `--host` option):

```
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	`wss://chinaeast2.stt.speech.azure.cn`

REGION / SERVICE OFFERING	HOST EXPRESSION
Text-to-Speech	`https://chinaeast2.tts.speech.azure.cn`
China North 2	
Speech-to-text	`wss://chinanorth2.stt.speech.azure.cn`
Text-to-Speech	`https://chinanorth2.tts.speech.azure.cn`