Contents

- Speech Service Documentation
- Overview
  - What is the Speech Service?
- Quickstarts
  - Try the Speech service for free
  - Recognize speech
    - Using C# (.NET on Windows)
    - Using C# (.NET Core on Windows)
    - Using C# (Universal Windows Platform)
    - Using C++ (Windows)
    - Using C++ (Linux)
    - Using Java (Android)
    - Using Java (Windows or Linux)
    - Using JavaScript (Browser)
    - Using Objective-C (iOS)
  - Get started with Speech Devices SDK
- Tutorials
  - Recognize speech
    - C#
    - C++
    - Java
    - REST
  - Recognize intents from speech
    - C#
    - C++
    - Java
  - Translate speech
    - C#
    - C++

- Java
- How-to guides
  - Synthesize speech (REST)
  - Batch transcription (REST)
  - Get the Speech Devices SDK
  - Customize
    - Customize acoustic models
    - Customize language models
    - Customize pronunciation
    - Customize voice fonts (bot brand voice)
    - Customize device wake word (Devices SDK)
  - Record voice samples
  - Prepare transcripts
  - Use custom audio streams
  - Migrate
    - From Bing Speech
    - From the Custom Speech service
    - From the Translator Speech API
- Concepts
  - Speech to Text
  - Text to Speech
  - Speech Translation
- Reference
  - PowerShell
  - Azure CLI
  - Language support
  - Regions
  - REST APIs
  - Swagger Reference
  - Speech SDK API
  - Speech synthesis markup
- Resources
  - FAQ

- Speech to Text FAQ
- Text to Speech FAQ
- Translator FAQ

Speech SDK

- Download
- Samples
- Shipping your application
- Release notes
- License
- Third-party notices

Speech Devices SDK

- Download
- Samples
- Release notes
- License
- Third-party notices

Troubleshoot the Speech SDK

Support

Pricing and limits

Azure Roadmap

Privacy & cookies

What is the Speech Service?

10/19/2018 • 4 minutes to read • Edit Online

Like the other Azure speech services, the Speech service is powered by the speech technologies used in products like Cortana and Microsoft Office.

The Speech service unites the Azure speech features previously available via the Bing Speech API, Translator Speech, Custom Speech, and Custom Voice services. Now, one subscription provides access to all of these capabilities.

Main Speech service functions

The primary functions of the Speech service are Speech to Text (also called speech recognition or transcription), Text to Speech (speech synthesis), and Speech Translation.

FUNCTION	FEATURES
Speech to Text	• Transcribes continuous real-time speech into text. • Can batch-transcribe speech from audio recordings. • Supports intermediate results, end-of-speech detection, automatic text formatting, and profanity masking. • Can call on Language Understanding (LUIS) to derive user intent from transcribed speech.*
Text to Speech	• Converts text to natural-sounding speech. • Offers multiple genders and/or dialects for many supported languages. • Supports plain text input or Speech Synthesis Markup Language (SSML).
Speech Translation	• Translates streaming audio in near-real-time. • Can also process recorded speech. • Provides results as text or synthesized speech.

Customize speech features

You can use your own data to train the models that underlie the Speech service's Speech-to-Text and Text-to-Speech features.

FEATURE	MODEL	PURPOSE
Speech to Text	Acoustic model	Helps transcribe particular speakers and environments, such as cars or factories.
	Language model	Helps transcribe field-specific vocabulary and grammar, such as medical or IT jargon.
	Pronunciation model	Helps transcribe abbreviations and acronyms, such as "IOU" for "I owe you."

FEATURE	MODEL	PURPOSE
Text to Speech	Voice font	Gives your app a voice of its own by training the model on samples of human speech.

You can use your custom models anywhere you use the standard models in your app's Speech-to-Text or Text-to-Speech functionality.

Use the Speech service

To simplify the development of speech-enabled applications, Microsoft provides the Speech SDK for use with the Speech service. The Speech SDK provides consistent native Speech-to-Text and Speech Translation APIs for C#, C++, and Java. If you develop with one of these languages, the Speech SDK makes development easier by handling the network details for you.

The Speech service also has a REST API that works with any programming language that can make HTTP requests. The REST interface does not offer the streaming, real-time functionality of the SDK.

METHOD	SPEECH TO TEXT	TEXT TO SPEECH	SPEECH TRANSLATION	DESCRIPTION
Speech SDK	Yes	No	Yes	Native APIs for C#, C++, and Java to simplify development.
REST	Yes	Yes	No	A simple HTTP-based API that makes it easy to add speech to your applications.

WebSockets

The Speech service also has WebSocket protocols for streaming Speech to Text and Speech Translation. The Speech SDKs use these protocols to communicate with the Speech service. Use the Speech SDK instead of trying to implement your own WebSocket communication with the Speech service.

If you already have code that uses Bing Speech or Translator Speech via WebSockets, you can update it to use the Speech service. The WebSocket protocols are compatible, only the endpoints are different.

Speech Devices SDK

The Speech Devices SDK is an integrated hardware and software platform for developers of speech-enabled devices. Our hardware partner provides reference designs and development units. Microsoft provides a device-optimized SDK that takes full advantage of the hardware's capabilities.

Speech scenarios

Use cases for the Speech service include:

- Create voice-triggered apps
- Transcribe call center recordings
- Implement voice bots

Voice user interface

Voice input is a great way to make your app flexible, hands-free, and quick to use. With a voice-enabled app, users can just ask for the information they want.

If your app is intended for use by the general public, you can use the default speech recognition models. They recognize a wide variety of speakers in common environments.

If your app is used in a specific domain, for example, medicine or IT, you can create a language model. You can use this model to teach the Speech service about the special terminology used by your app.

If your app is used in a noisy environment, such as a factory, you can create a custom acoustic model. This model helps the Speech service to distinguish speech from noise.

Call center transcription

Often, call center recordings are consulted only if an issue arises with a call. With the Speech service, it's easy to transcribe every recording to text. You can easily index the text for full-text search or apply Text Analytics to detect sentiment, language, and key phrases.

If your call center recordings involve specialized terminology, such as product names or IT jargon, you can create a language model to teach the Speech service the vocabulary. A custom acoustic model can help the Speech service understand less-than-optimal phone connections.

For more information about this scenario, read more about batch transcription with the Speech service.

Voice bots

Bots are a popular way to connect users with the information they want and customers with businesses they like. When you add a conversational user interface to your website or app, the functionality is easier to find and quicker to access. With the Speech service, this conversation takes on a new dimension of fluency by responding to spoken queries in kind.

To add a unique personality to your voice-enabled bot, you can give it a voice of its own. Creating a custom voice is a two-step process. First, make recordings of the voice you want to use. Then submit those recordings along with a text transcript to the Speech service's voice customization portal, which does the rest. After you create your custom voice, the steps to use it in your app are straightforward.

Next steps

Get a subscription key for the Speech service.

Try the Speech service for free

Try the Speech service for free

10/19/2018 • 4 minutes to read • Edit Online

Getting started with the Speech service is easy and affordable. A 30-day free trial lets you discover what the service can do and decide whether it's right for your application's needs.

If you need more time, sign up for a Microsoft Azure account—it comes with $200 in service credit that you can apply toward a paid Speech service subscription for up to 30 days.

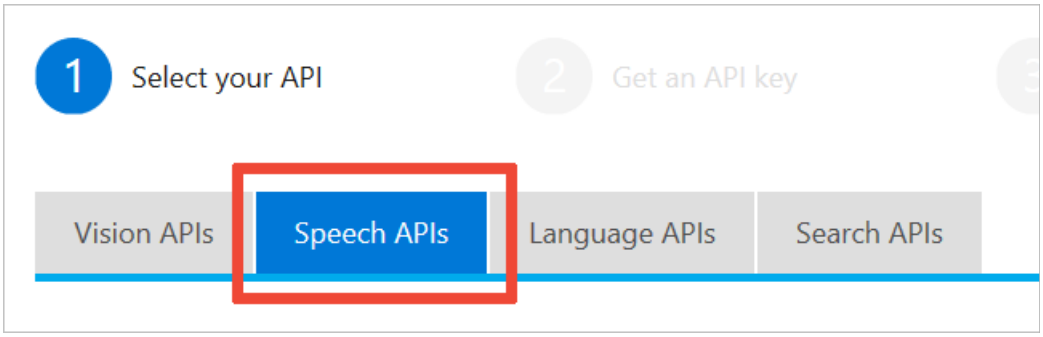
Finally, the Speech service offers a free, low-volume tier that's suitable for developing applications. You can keep this free subscription even after your service credit expires.

Free trial

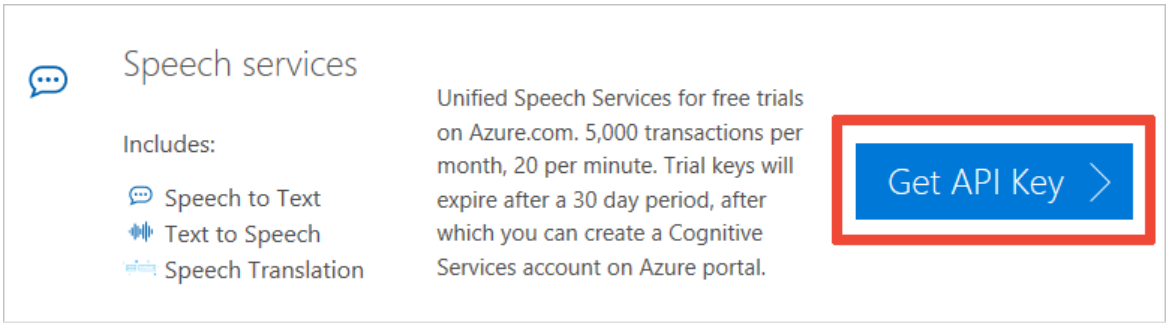
The 30-day free trial gives you access to the standard pricing tier for a limited time.

To sign up for a 30-day free trial:

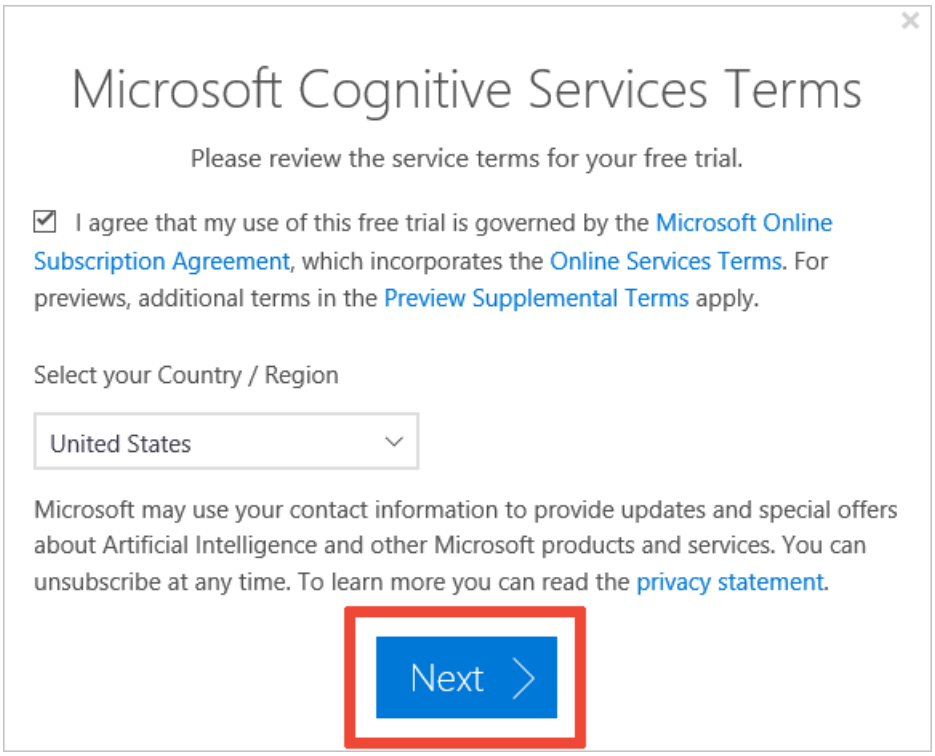
1. Go to Try Cognitive Services.
2. Select the **Speech APIs** tab.



3. Under **Speech services**, select the **Get API Key** button.



4. Agree to the terms and select your locale from the drop-down menu.



5. Sign in by using your Microsoft, Facebook, LinkedIn, or GitHub account.

   You can sign up for a free Microsoft account at the Microsoft account portal. To get started, click **Sign in with Microsoft** and then, when asked to sign in, click **Create one.** Follow the steps to create and verify your new Microsoft account.

After you sign in to Try Cognitive Services, your free trial begins. The displayed webpage lists all the Azure Cognitive Services services for which you currently have trial subscriptions. Two subscription keys are listed beside **Speech services**. You can use either key in your applications.

> **NOTE**
> All free trial subscriptions are in the West US region. When you make requests, be sure to use the `westus` endpoint.

New Azure account

New Azure accounts receive a $200 service credit that is available for up to 30 days. You can use this credit to further explore the Speech service or to begin application development.

To sign up for a new Azure account, go to the Azure sign-up page, click **Start Free,** and create a new Azure account using your Microsoft account.

You can sign up for a free Microsoft account at the Microsoft account portal. To get started, click **Sign in with Microsoft** and then, when asked to sign in, click **Create one.** Follow the steps to create and verify your new Microsoft account.

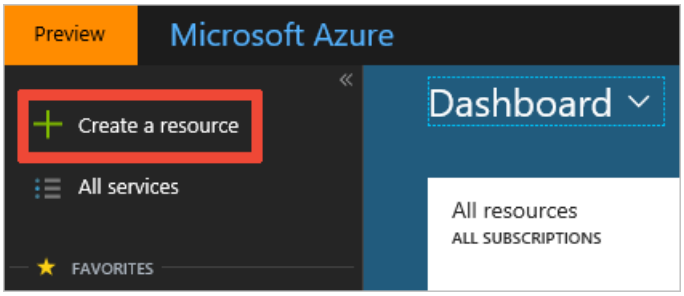
After creating your Azure account, follow the steps in the next section to start a subscription to the Speech service.

Create a Speech resource in Azure

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

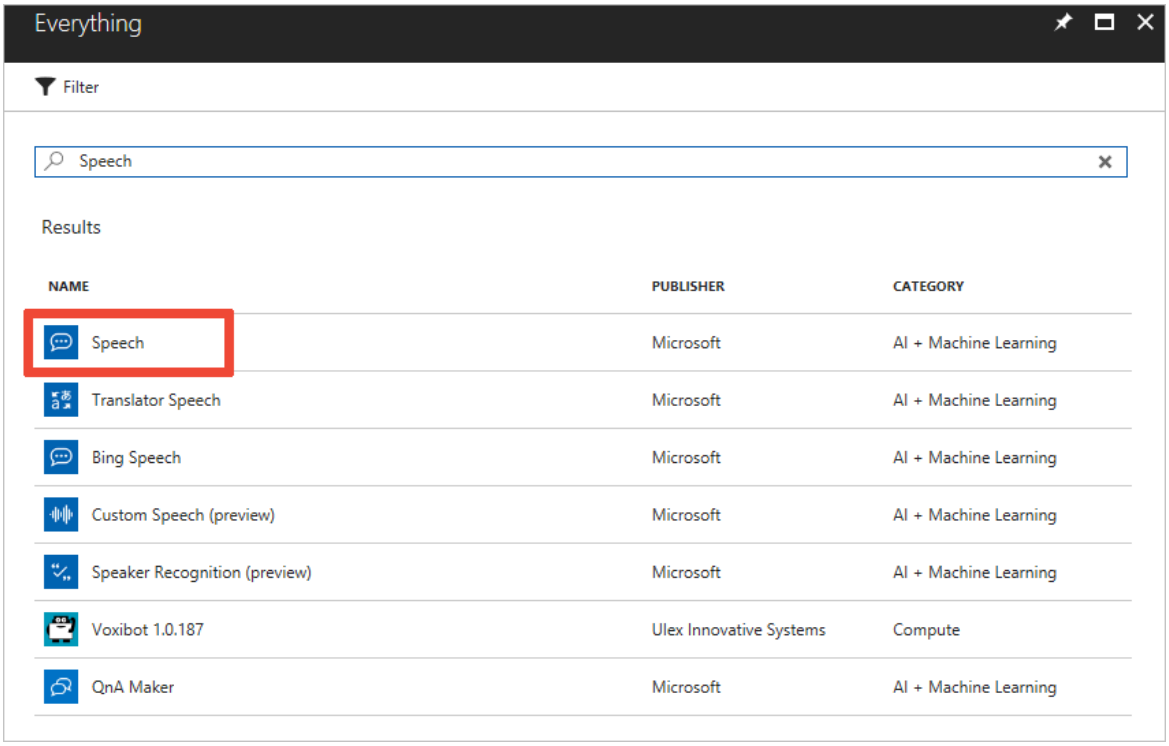
1. Sign in to the Azure portal using your Microsoft account.

2. Select **Create a resource** at the top left of the portal.

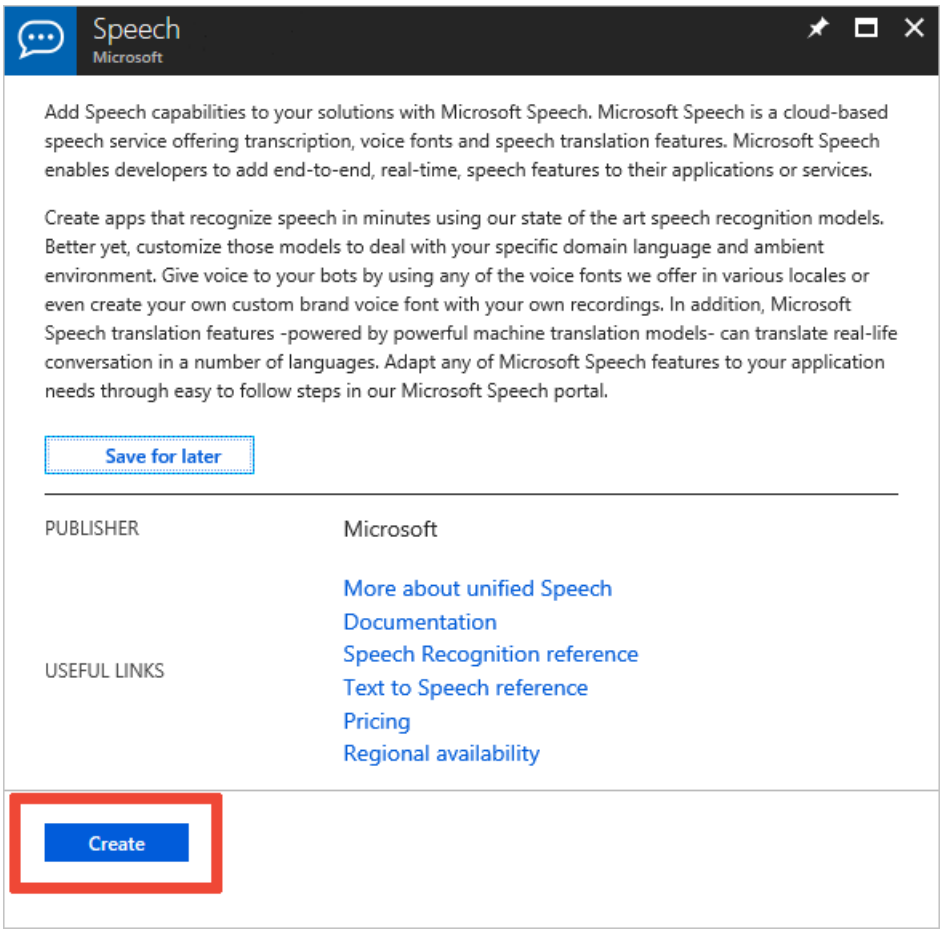


3. In the **New** window, search for **speech**.

4. In the search results, select **Speech**.

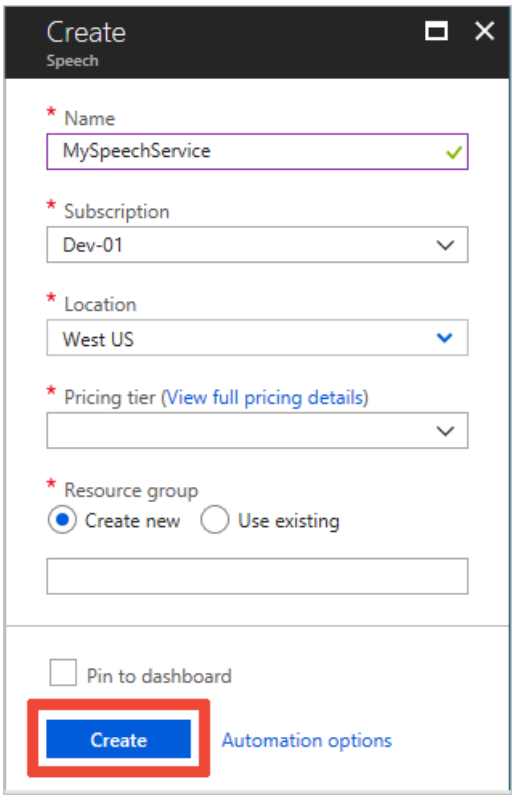


5. Under **Speech**, select the **Create** button.

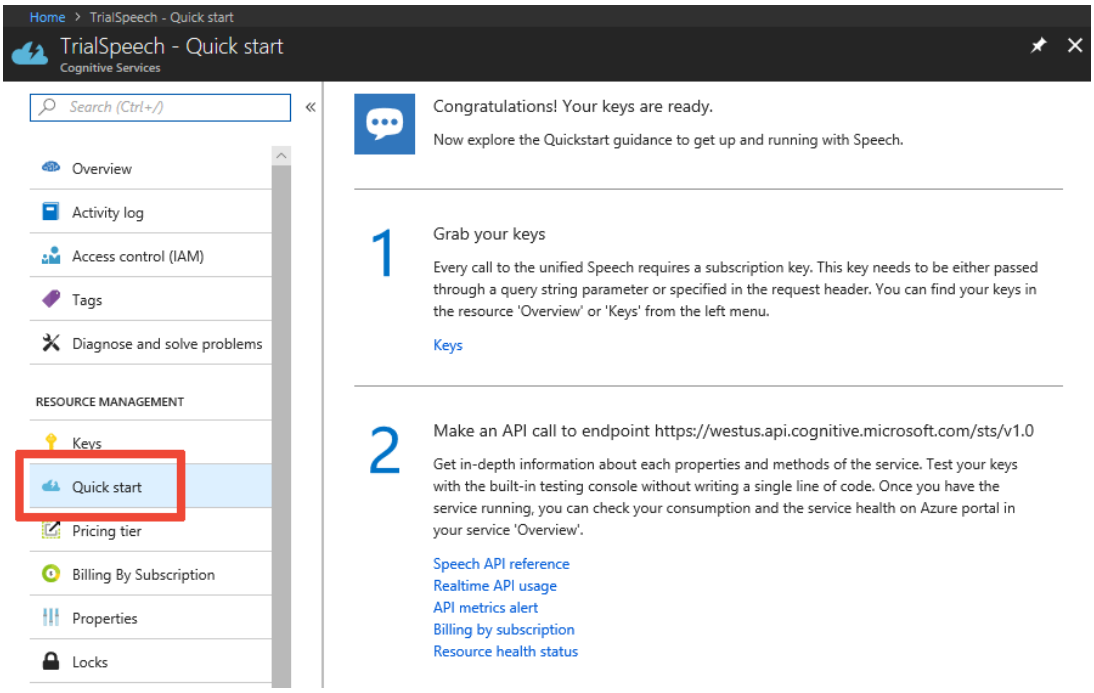


6. Under **Create**, enter:

   - A name for the new resource. The name helps you distinguish among multiple subscriptions to the same service.
   - Choose the Azure subscription that the new resource is associated with to determine how the fees are billed.
   - Choose the region where the resource will be used. Currently, the Speech service is available in East Asia, North Europe, and West US regions.
   - Choose either a free or paid pricing tier. Click **View full pricing details** for complete information about pricing and usage quotas for each tier.
   - 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.
   - For convenient access to your subscription in the future, select the **Pin to dashboard** check box.
   - Select **Create.**



It takes a moment to create and deploy your new Speech resource. Select **Quickstart** to see information about your new resource.



7. Under **Quickstart**, click the **Keys** link under step 1 to display your subscription keys. Each subscription has two keys; you can use either key in your application. Select the button next to each key to copy it to the clipboard for pasting into your code.

> **NOTE**
>
> You can create an unlimited number of standard-tier subscriptions in one or multiple regions. However, you can create only one free-tier subscription. Model deployments on the free tier that remain unused for 7 days will be decomissioned automatically.

Switch to a new subscription

To switch from one subscription to another, for example when your free trial expires or when you publish your application, replace the region and subscription key in your code with the region and subscription key of the new Azure resource.

> **NOTE**
> Free trial keys are created in the West US (`westus`) region. A subscription created via the Azure dashboard may be in some other region if you so choose.

- If your application uses a Speech SDK, you provide the region code, such as `westus`, when creating a speech configuration.
- If your application uses one of the Speech service's REST APIs, the region is part of the endpoint URI you use when making requests.

Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

Next steps

Complete one of our 10-minute quickstarts or check out our SDK samples:

Quickstart: Recognize speech in C# Speech SDK samples

Quickstart: Recognize speech in C# under .NET Framework on Windows by using the Speech SDK

10/19/2018 • 3 minutes to read • Edit Online

In this article, you create a C# console application for .NET Framework on Windows by using the Speech SDK. You transcribe speech to text in real time from your PC's microphone. The application is built with the Speech SDK NuGet package and Microsoft Visual Studio 2017 (any edition).

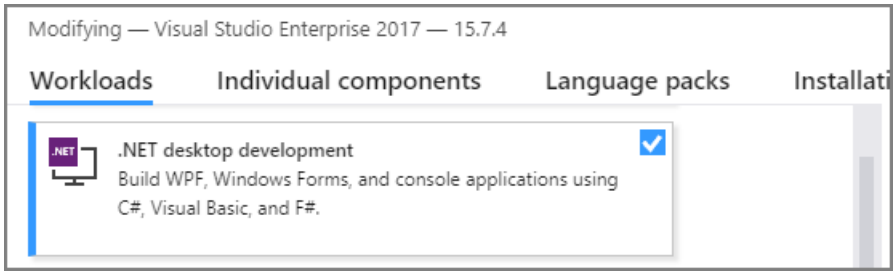
Prerequisites

You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech service for free for details.

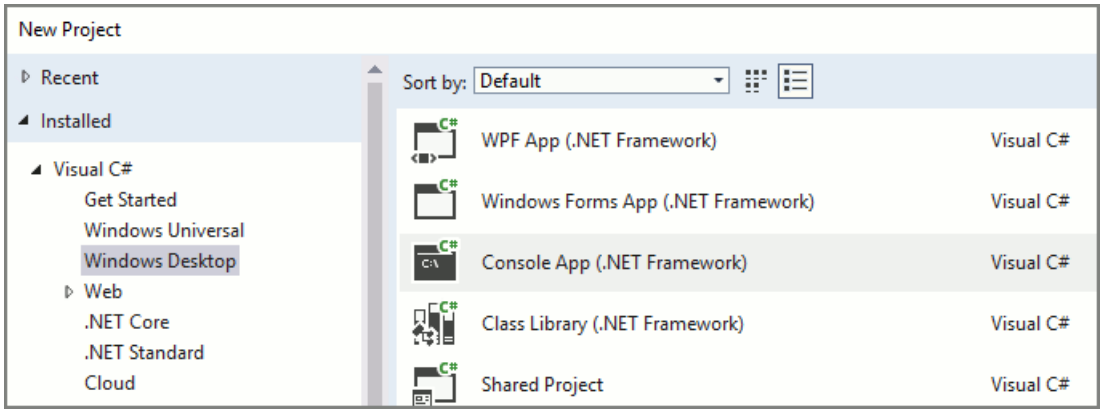
Create a Visual Studio project

1. Start Visual Studio 2017.
2. Make sure the **.NET desktop environment** workload is available. Choose **Tools** > **Get Tools and Features** from the Visual Studio menu bar to open the Visual Studio installer. If this workload is already enabled, close the dialog.

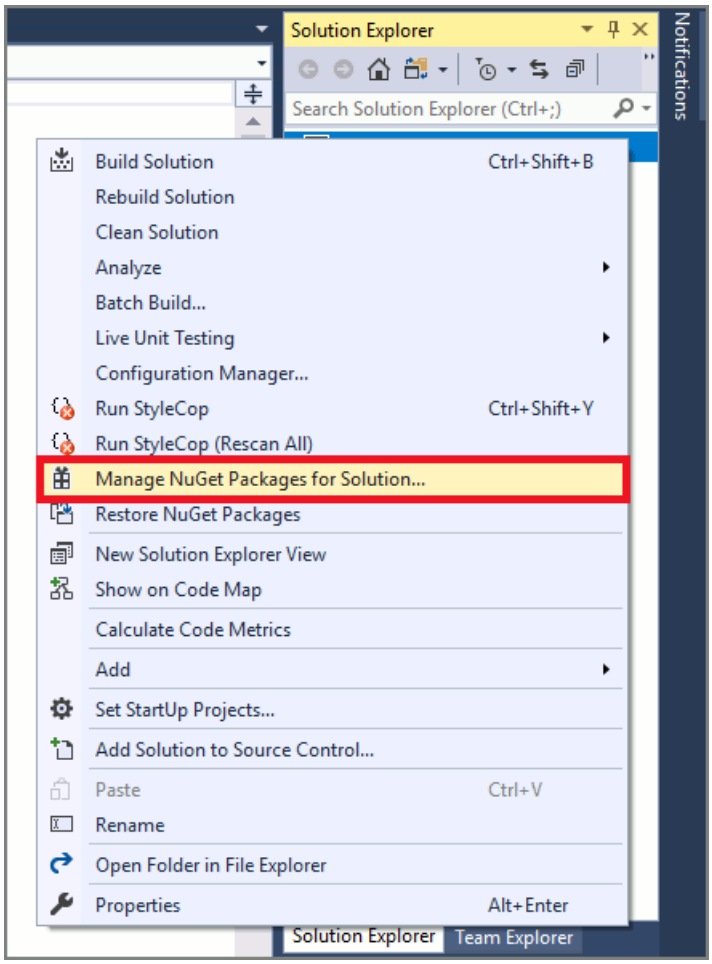
   Otherwise, mark the checkbox next to **.NET desktop development,** then click the **Modify** button at the lower right corner of the dialog. Installation of the new feature will take a moment.



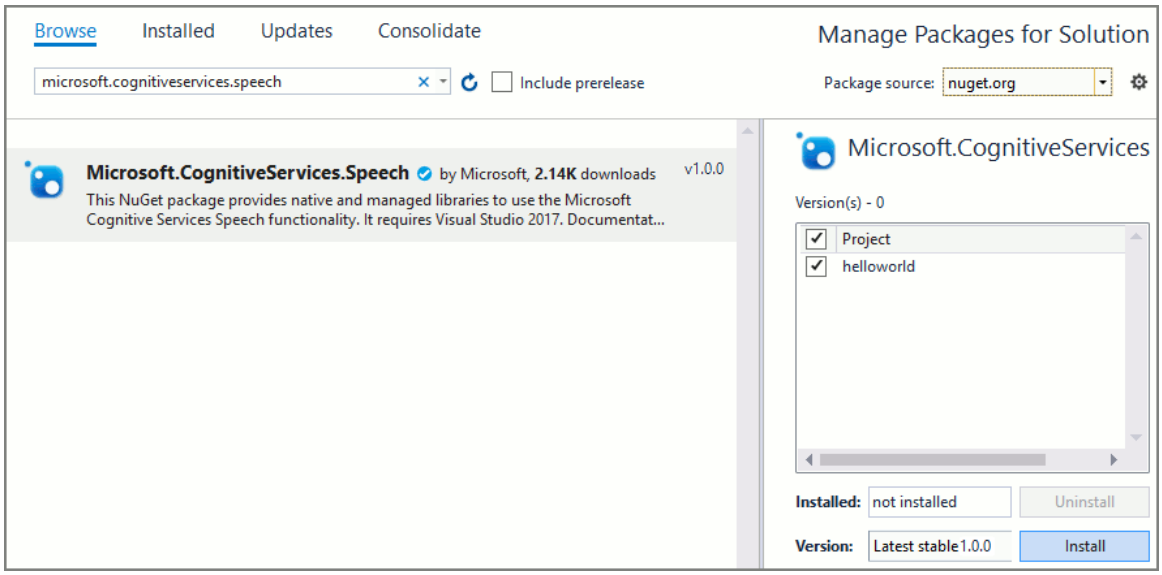
3. Create a new Visual C# Console App. In the **New Project** dialog box, from the left pane, expand **Installed** > **Visual C#** > **Windows Desktop** and then choose **Console App (.NET Framework)**. For the project name, enter *helloworld*.



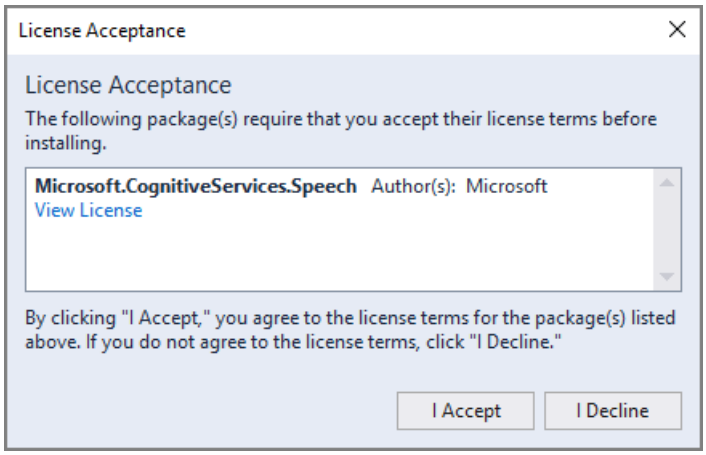
4. Install and reference the Speech SDK NuGet package. In the Solution Explorer, right-click the solution and select **Manage NuGet Packages for Solution**.



5. In the upper-right corner, in the **Package Source** field, select **nuget.org**. Search for the `Microsoft.CognitiveServices.Speech` package and install it into the **helloworld** project.

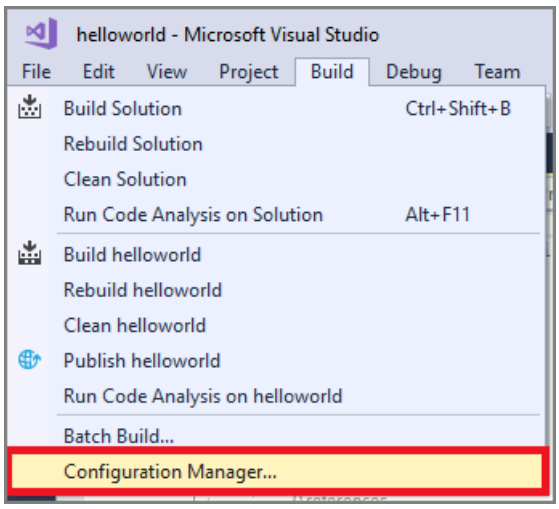


6. Accept the displayed license to begin installation of the NuGet package.

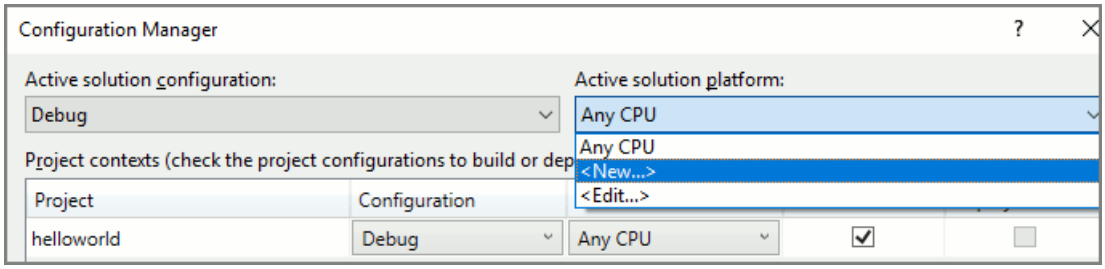


After the package is installed, a confirmation appears in the Package Manager console.

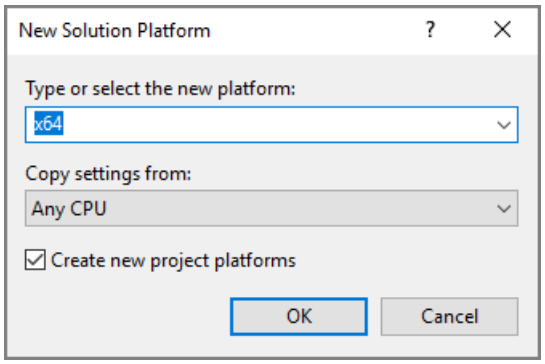
7. Create a platform configuration matching your PC architecture via the Configuration Manager. Select **Build** > **Configuration Manager**.



8. In the **Configuration Manager** dialog box, add a new platform. From the **Active solution platform** drop-down list, select **New**.



9. If you are running 64-bit Windows, create a new platform configuration named `x64`. If you are running 32-bit Windows, create a new platform configuration named `x86`.



Add sample code

1. Open `Program.cs`, and replace all the code in it with the following.

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

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeSpeechAsync()
        {
            // Creates an instance of a speech config with specified subscription key and service
region.
            // Replace with your own subscription key and service region (e.g., "westus").
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            // Creates a speech recognizer.
            using (var recognizer = new SpeechRecognizer(config))
            {
                Console.WriteLine("Say something...");

                // Performs recognition. RecognizeOnceAsync() returns when the first utterance has
been recognized,
                // so it is suitable only for single shot recognition like command or query. For
long-running
                // recognition, use StartContinuousRecognitionAsync() instead.
                var result = await recognizer.RecognizeOnceAsync();

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

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

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

2. In the same file, replace the string `YourSubscriptionKey` with your Speech service subscription key.

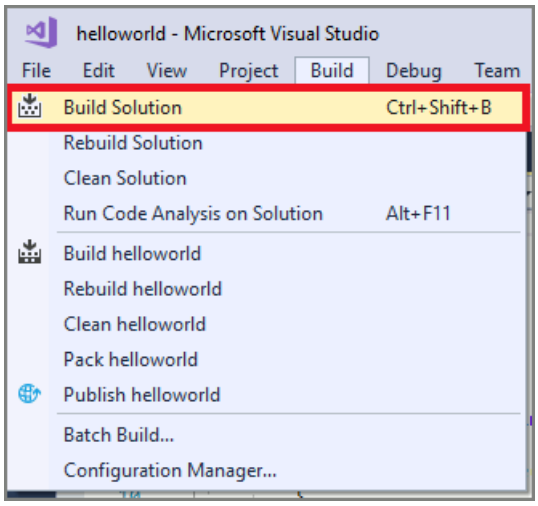
3. Also replace the string `YourServiceRegion` with the region associated with your subscription (for

example, `westus` for the free trial subscription).

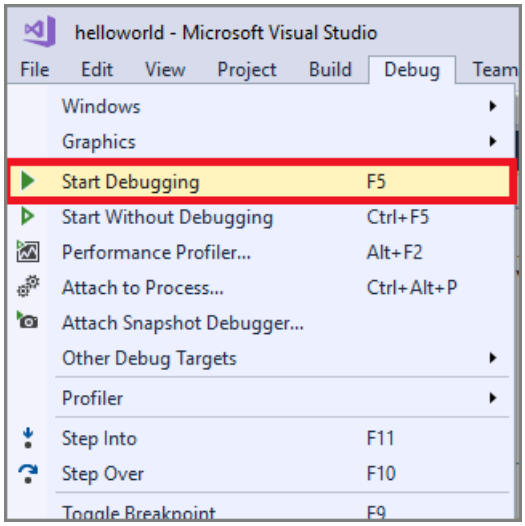
4. Save changes to the project.

Build and run the app

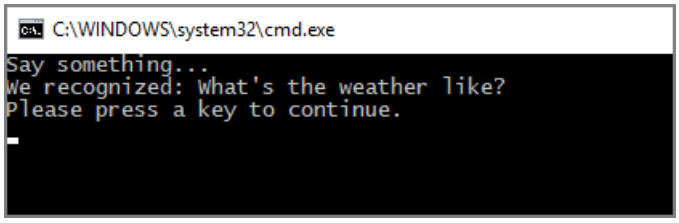
1. Build the application. From the menu bar, select **Build** > **Build Solution**. The code should compile without errors now.



2. Start the application. From the menu bar, select **Debug** > **Start Debugging**, or press **F5**.



3. A console window appears, prompting you to say something. Speak an English phrase or sentence. Your speech is transmitted to the Speech service and transcribed to text, which appears in the same window.



Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/csharp-dotnet-windows` folder.

Next steps

Recognize intents from speech by using the Speech SDK for C#

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in C# under .NET Core on Windows by using the Speech SDK

10/19/2018 • 3 minutes to read • Edit Online

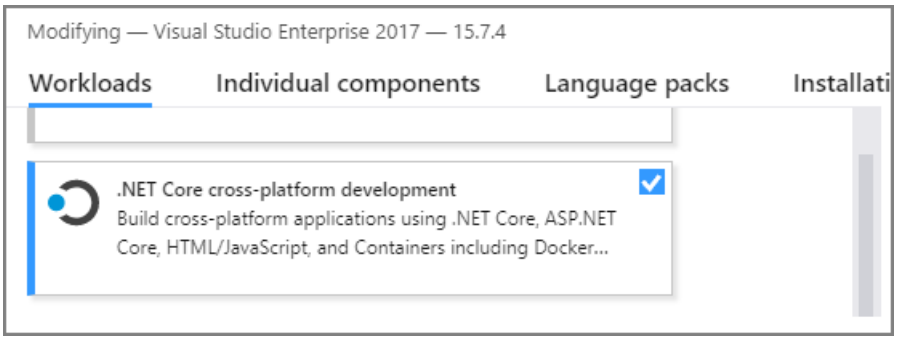
In this article, you create a C# console application for .NET Core on Windows by using the Cognitive Services Speech SDK. You transcribe speech to text in real time from your PC's microphone. The application is built with the Speech SDK NuGet package and Microsoft Visual Studio 2017 (any edition).

> **NOTE**
> .NET Core is an open-source, cross-platform .NET platform that implements the .NET Standard specification.

You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech service for free for details.

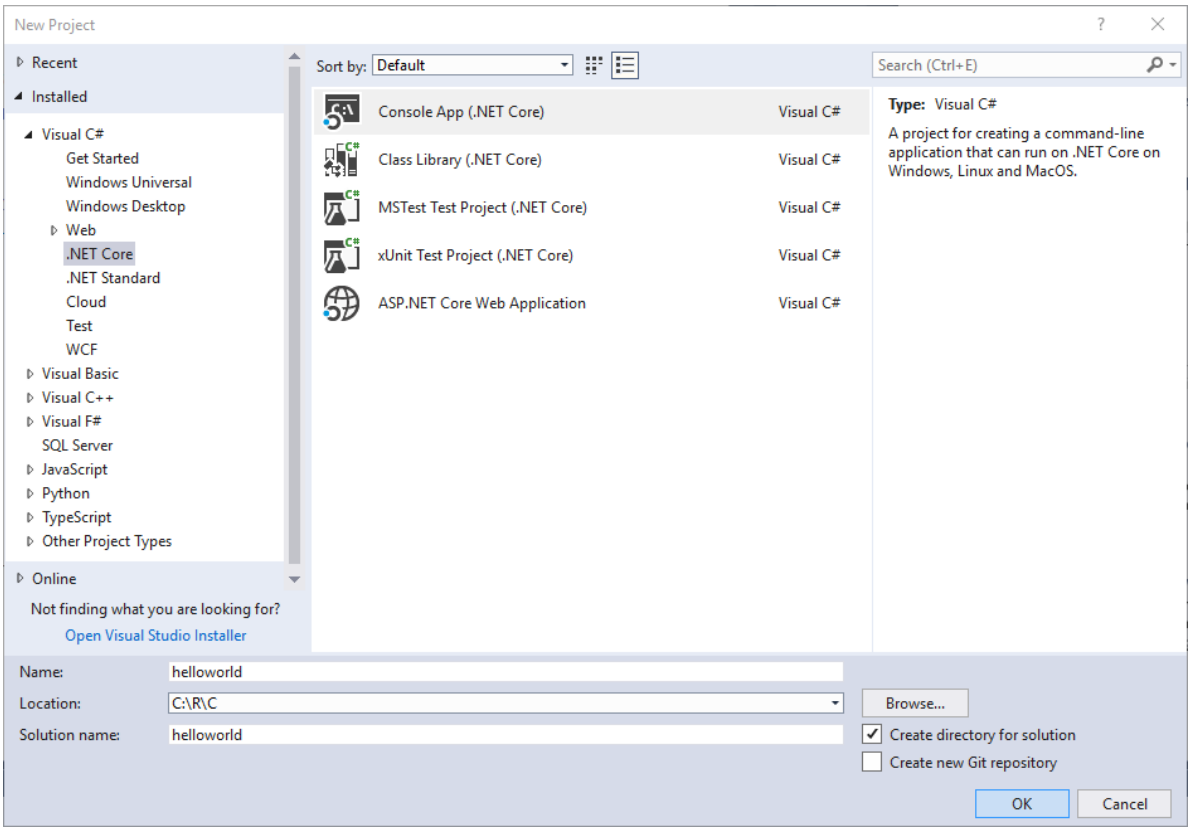
Create a Visual Studio project

1. Start Visual Studio 2017.
2. Make sure the **.NET cross-platform development** workload is available. Choose **Tools** > **Get Tools and Features** from the Visual Studio menu bar to open the Visual Studio installer. If this workload is already enabled, close the dialog box.

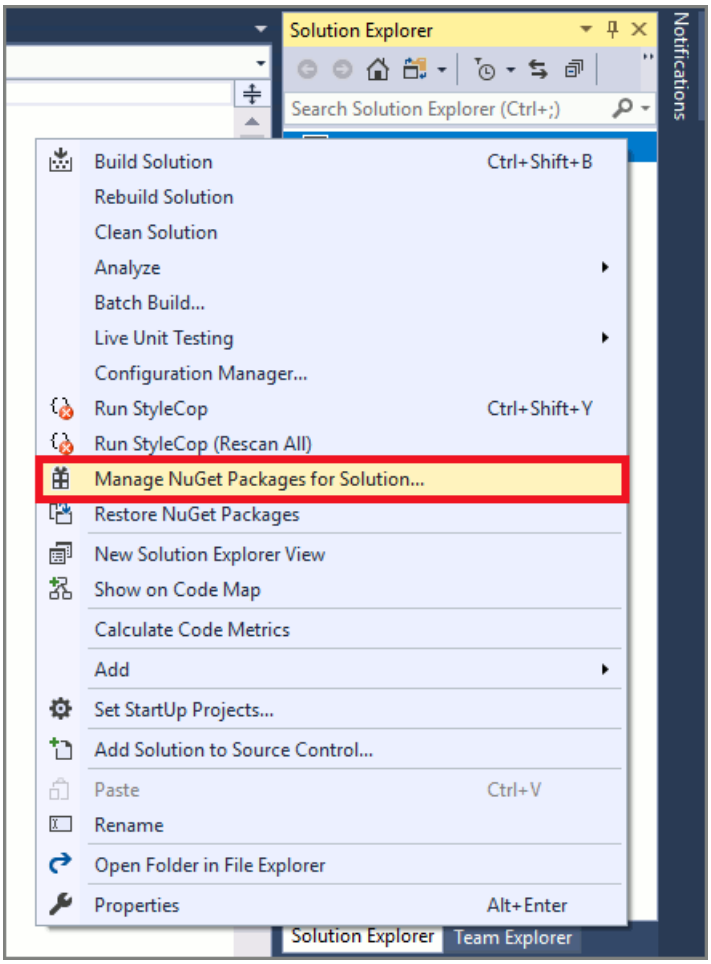


   Otherwise, select the box next to **.NET Core cross-platform development,** and select **Modify** at the lower right corner of the dialog box. Installation of the new feature will take a moment.

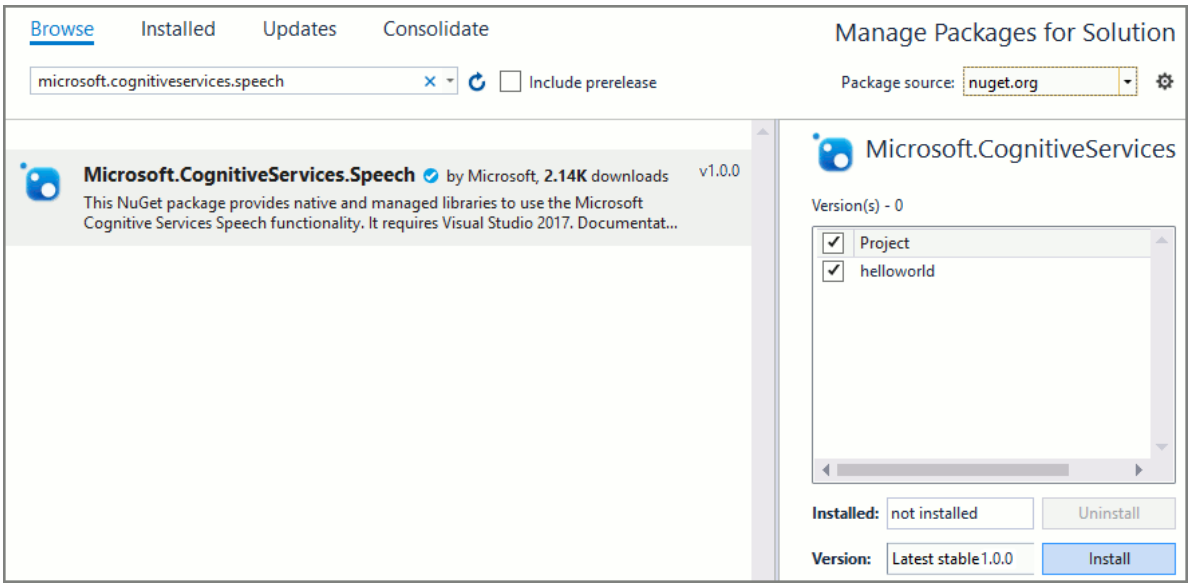
3. Create a new Visual C# .NET Core Console App. In the **New Project** dialog box, from the left pane, expand **Installed** > **Visual C#** > **.NET Core**. Then select **Console App (.NET Core)**. For the project name, enter *helloworld*.



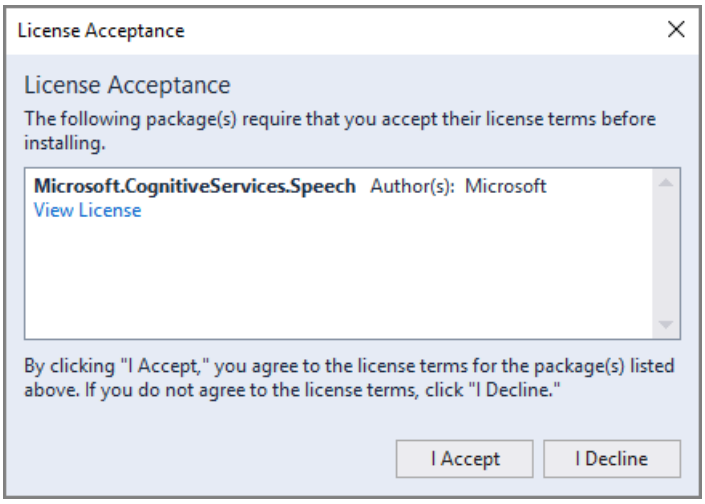
4. Install and reference the Speech SDK NuGet package. In Solution Explorer, right-click the solution and select **Manage NuGet Packages for Solution**.



5. In the upper-right corner, in the **Package Source** field, select **nuget.org**. Search for the `Microsoft.CognitiveServices.Speech` package, and install it into the **helloworld** project.



6. Accept the displayed license to begin installation of the NuGet package.



After the package is installed, a confirmation appears in the Package Manager console.

Add sample code

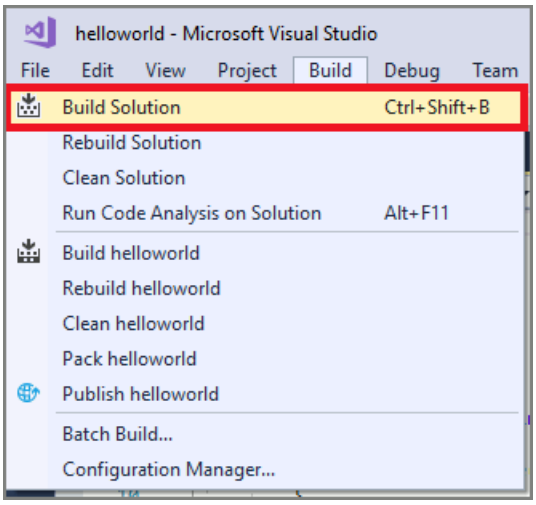
1. Open `Program.cs`, and replace all the code in it with the following.

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

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeSpeechAsync()
        {
            // Creates an instance of a speech config with specified subscription key and service
region.
            // Replace with your own subscription key // and service region (e.g., "westus").
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            // Creates a speech recognizer.
            using (var recognizer = new SpeechRecognizer(config))
            {
                Console.WriteLine("Say something...");

                // Performs recognition. RecognizeOnceAsync() returns when the first utterance has been
recognized,
                // so it is suitable only for single shot recognition like command or query. For long-
running
                // recognition, use StartContinuousRecognitionAsync() instead.
                var result = await recognizer.RecognizeOnceAsync();

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

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

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

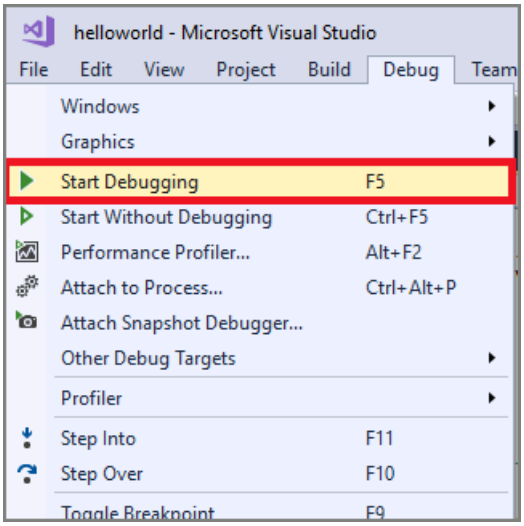
2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Also replace the string `YourServiceRegion` with the region associated with your subscription (for example, `westus` for the free trial subscription).
4. Save changes to the project.

Build and run the app

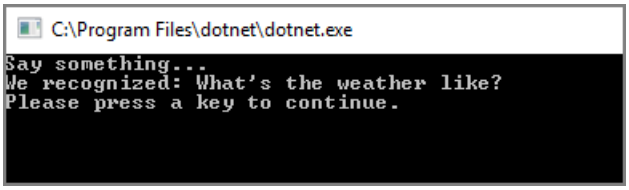
1. Build the application. From the menu bar, choose **Build** > **Build Solution**. The code should compile without errors.



2. Start the application. From the menu bar, choose **Debug** > **Start Debugging**, or press **F5**.



3. A console window appears, prompting you to say something. Speak an English phrase or sentence. Your speech is transmitted to the Speech service and transcribed to text, which appears in the same window.



Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/csharp-dotnetcore-windows` folder.

Next steps

Recognize intents from speech by using the Speech SDK for C#

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in a UWP app by using the Speech SDK

10/19/2018 • 5 minutes to read • Edit Online

In this article, you create a C# Universal Windows Platform (UWP) application by using the Cognitive Services Speech SDK. You transcribe speech to text in real time from your device's microphone. The application is built with the Speech SDK NuGet Package and Microsoft Visual Studio 2017 (any edition).

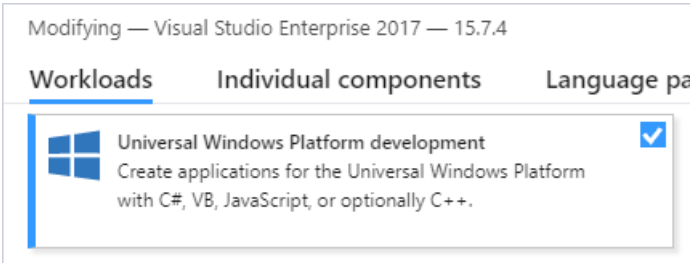
> **NOTE**
>
> The Universal Windows Platform lets you develop apps that run on any device that supports Windows 10, including PCs, Xbox, Surface Hub, and other devices.

Prerequisites

You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech service for free for details.

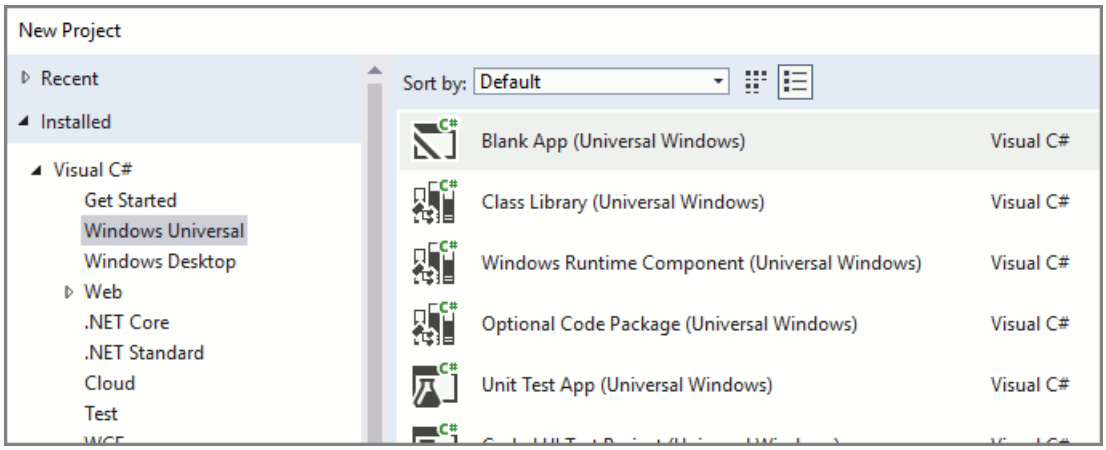
Create a Visual Studio project

1. Start Visual Studio 2017.
2. Make sure the **Universal Windows Platform development** workload is available. Choose **Tools** > **Get Tools and Features** from the Visual Studio menu bar to open the Visual Studio installer. If this workload is already enabled, close the dialog box.

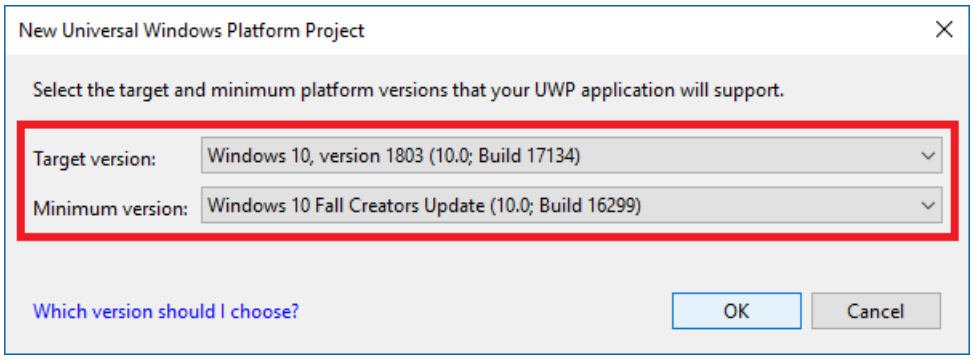


   Otherwise, select the box next to **.NET cross-platform development,** and select **Modify** at the lower right corner of the dialog box. Installation of the new feature takes a moment.

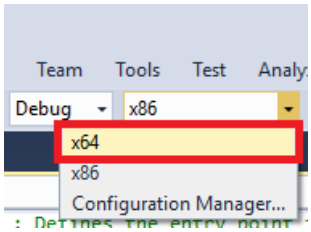
3. Create a blank Visual C# Universal Windows app. First, choose **File** > **New** > **Project** from the menu. In the **New Project** dialog box, expand **Installed** > **Visual C#** > **Windows Universal** in the left pane. Then select **Blank App (Universal Windows)**. For the project name, enter *helloworld*.



4. The Speed SDK requires that your application be built for the Windows 10 Fall Creators Update or later. In the **New Universal Windows Platform Project** window that pops up, choose **Windows 10 Fall Creators Update (10.0; Build 16299)** as **Minimum version**. In the **Target version** box, select this or any later version, and then click **OK**.

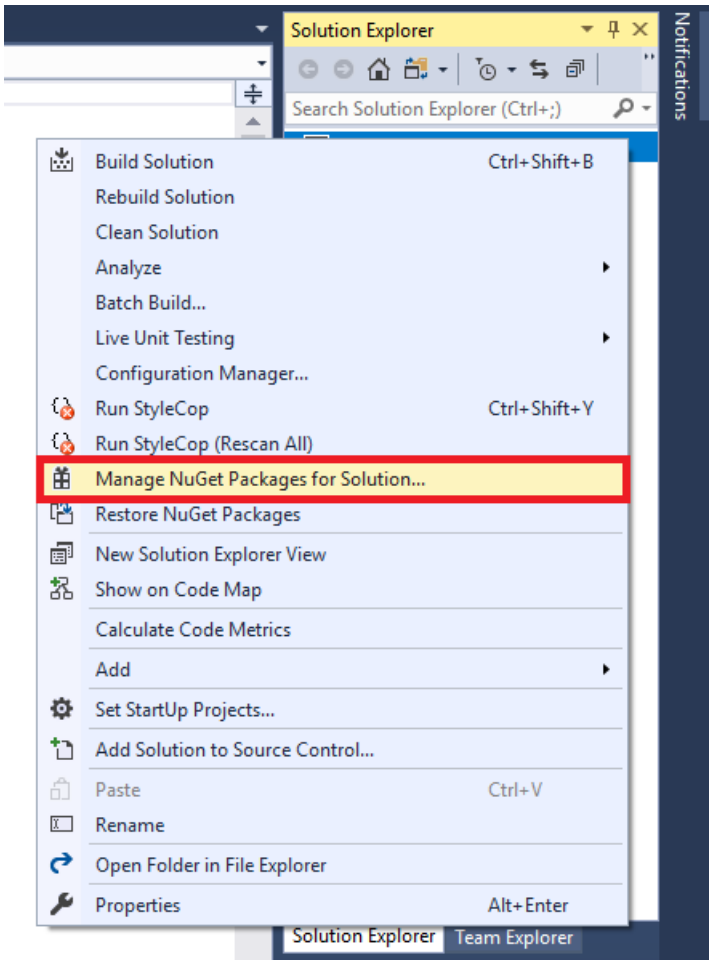


5. If you're running 64-bit Windows, you can switch your build platform to `x64` by using the drop-down menu in the Visual Studio toolbar. (64-bit Windows can run 32-bit applications, so you can leave it set to `x86` if you prefer.)

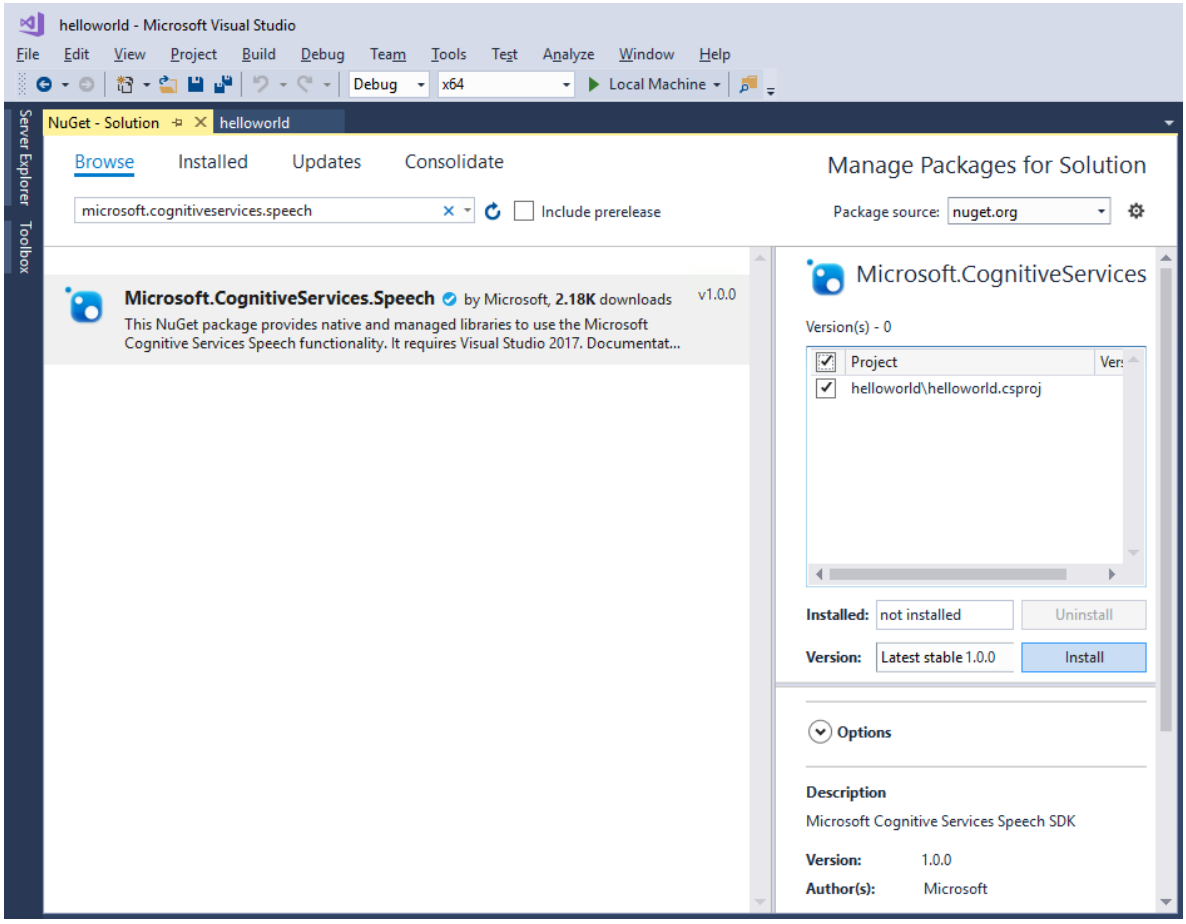


> **NOTE**
> The Speech SDK supports Intel-compatible processors only. ARM is currently not supported.

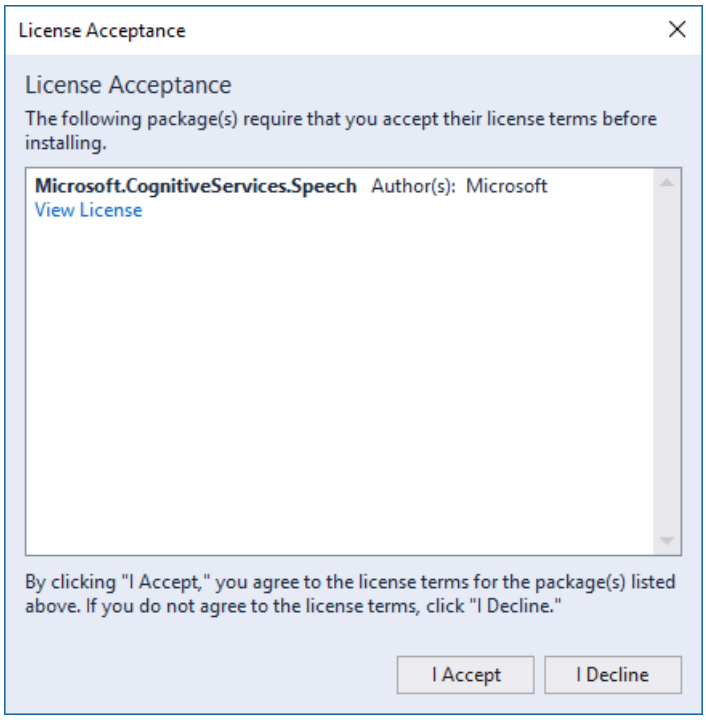
6. Install and reference the Speech SDK NuGet package. In Solution Explorer, right-click the solution, and select **Manage NuGet Packages for Solution**.



7. In the upper-right corner, in the **Package Source** field, select **nuget.org**. Search for the `Microsoft.CognitiveServices.Speech` package, and install it into the **helloworld** project.



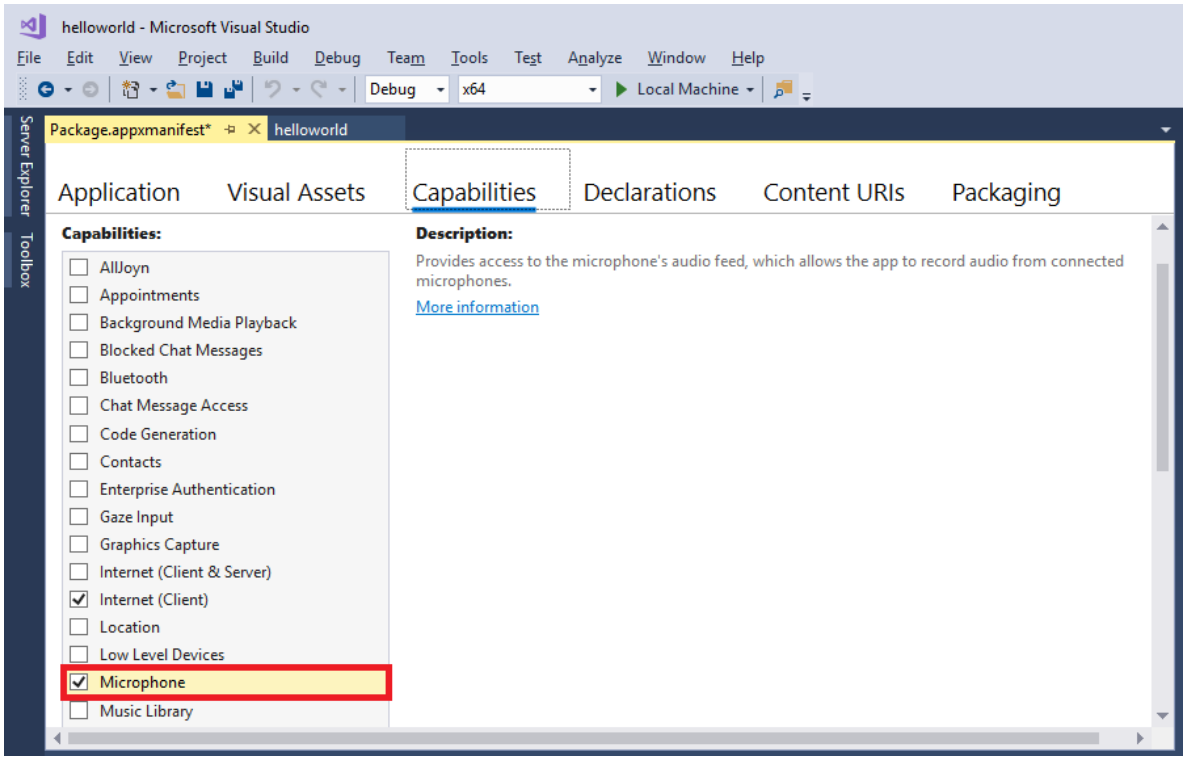
8. Accept the displayed license to begin installation of the NuGet package.



9. The following output line appears in the Package Manager console.

```
Successfully installed 'Microsoft.CognitiveServices.Speech 1.0.1' to helloworld
```

10. Because the application uses the microphone for speech input, add the **Microphone** capability to the project. In Solution Explorer, double-click **Package.appxmanifest** to edit your application manifest. Then switch to the **Capabilities** tab, select the box for the **Microphone** capability, and save your changes.



Add sample code

1. The application's user interface is defined by using XAML. Open `MainPage.xaml` in Solution Explorer. In the designer's XAML view, insert the following XAML snippet into the Grid tag (between `<Grid>` and `</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="SpeechRecognitionButton" Content="Speech recognition with microphone input"
Margin="0,10,10,0" Click="SpeechRecognitionFromMicrophone_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>
```

2. Open the code-behind source file `MainPage.xaml.cs` (find it grouped under `MainPage.xaml`). Replace all the code in it with the following.

```
using System;
using System.Text;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        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 async void SpeechRecognitionFromMicrophone_ButtonClicked(object sender,
RoutedEventArgs e)
        {
            // Creates an instance of a speech config with specified subscription key and service
region.
            // Replace with your own subscription key and service region (e.g., "westus").
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            try
            {
                // Creates a speech recognizer using microphone as audio input.
                using (var recognizer = new SpeechRecognizer(config))
                {
                    // Starts recognition. It returns when the first utterance has been recognized.
                    var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);

                    // Checks result.
                    StringBuilder sb = new StringBuilder();
                    if (result.Reason == ResultReason.RecognizedSpeech)
                    {
                        sb.AppendLine($"RECOGNIZED: Text={result.Text}");
                    }
                    else if (result.Reason == ResultReason.NoMatch)
                    {
                        sb.AppendLine($"NOMATCH: Speech could not be recognized.");
                    }
                    else if (result.Reason == ResultReason.Canceled)
                    {
                        var cancellation = CancellationDetails.FromResult(result);
                        sb.AppendLine($"CANCELED: Reason={cancellation.Reason}");

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

                    // Update the UI
                    NotifyUser(sb.ToString(), NotifyType.StatusMessage);
                }
            }
            catch(Exception ex)
            {
                NotifyUser($"Enable Microphone First.\n {ex.ToString()}", NotifyType.ErrorMessage);
            }
        }

        private enum NotifyType
        {
            StatusMessage,
            ErrorMessage
        };

        private void NotifyUser(string strMessage, NotifyType type)
        {
            // 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, () =>
```

```
                var task = DispatcherRunAsync(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;

            // Collapse the StatusBlock if it has no text to conserve real estate.
            StatusBorder.Visibility = !string.IsNullOrEmpty(StatusBlock.Text) ? Visibility.Visible :
Visibility.Collapsed;
            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);
            }
        }
    }
}
```

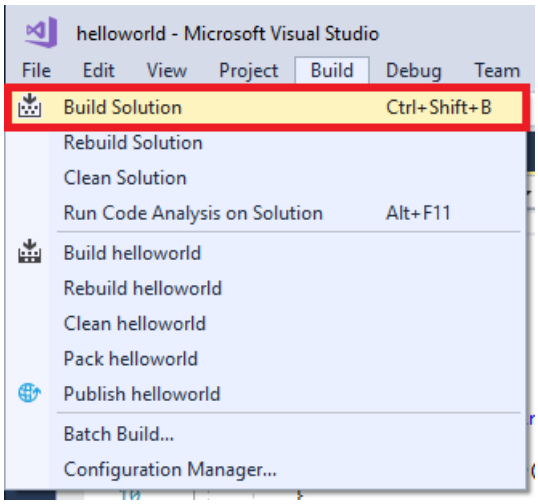
3. In the `SpeechRecognitionFromMicrophone_ButtonClicked` handler in this file, replace the string `YourSubscriptionKey` with your subscription key.

4. In the `SpeechRecognitionFromMicrophone_ButtonClicked` handler, replace the string `YourServiceRegion` with the region associated with your subscription (for example, `westus` for the free trial subscription).

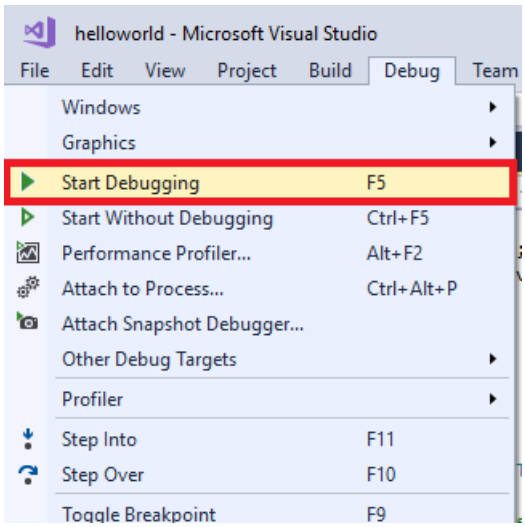
5. Save all changes to the project.

Build and run the app

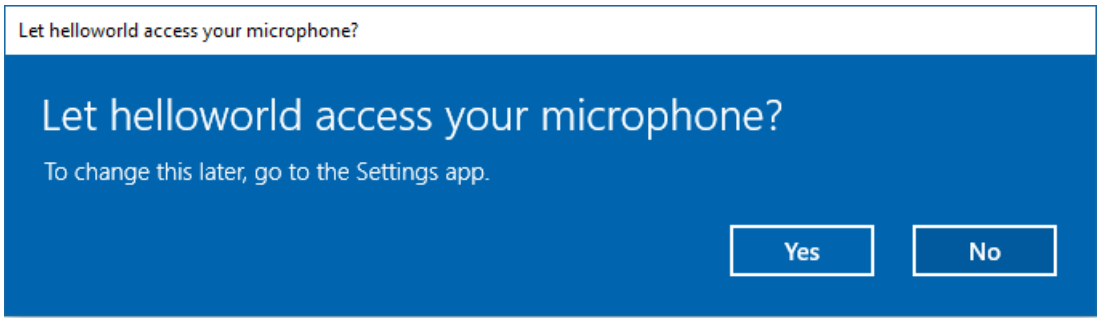
1. Build the application. From the menu bar, select **Build** > **Build Solution**. The code should compile without errors now.



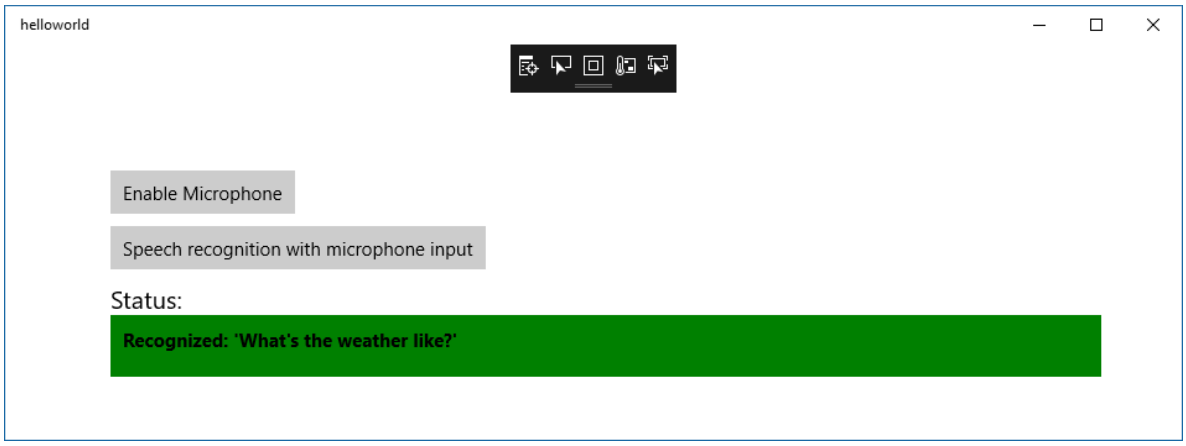
2. Start the application. From the menu bar, select **Debug** > **Start Debugging**, or press **F5**.



3. A window pops up. Select **Enable Microphone**, and acknowledge the permission request that pops up.



4. Select **Speech recognition with microphone input**, and speak an English phrase or sentence into your device's microphone. Your speech is transmitted to the Speech service and transcribed to text, which appears in the window.



Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/csharp-uwp` folder.

Next steps

Recognize intents from speech by using the Speech SDK for C#

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in C++ on Windows by using the Speech SDK

10/19/2018 • 3 minutes to read • Edit Online

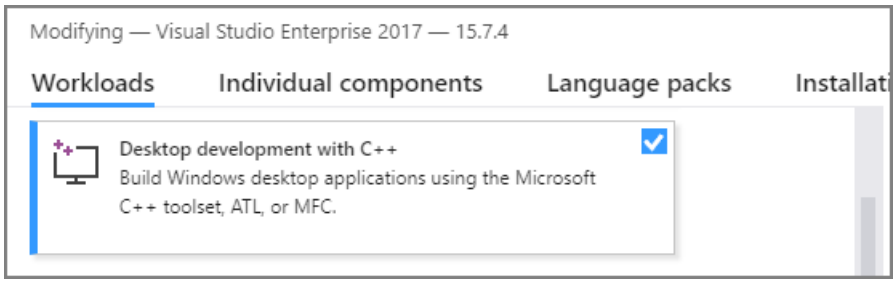
In this article, you create a C++ console application for Windows. You use the Cognitive Services Speech SDK to transcribe speech to text in real time from your PC's microphone. The application is built with the Speech SDK NuGet package and Microsoft Visual Studio 2017 (any edition).

Prerequisites

You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech service for free for details.

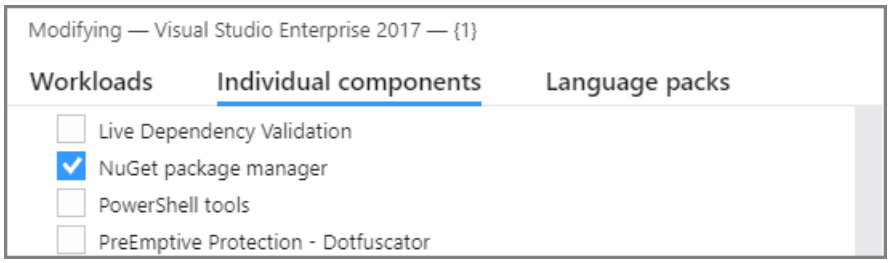
Create a Visual Studio project

1. Start Visual Studio 2017.
2. Make sure the **Desktop development with C++** workload is available. Choose **Tools** > **Get Tools and Features** from the Visual Studio menu bar to open the Visual Studio installer. If this workload is already enabled, skip to the next step.

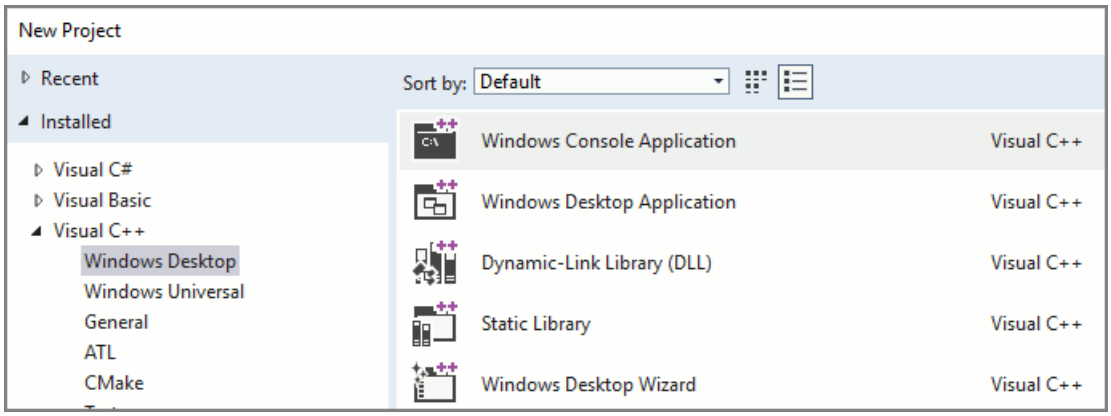


   Otherwise, check the box next to **Desktop development with C++**.

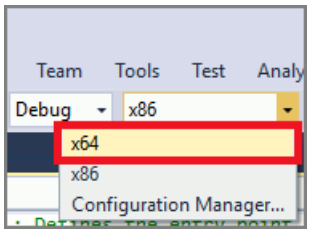
3. Make sure the **NuGet package manager** component is available. Switch to the **Individual components** tab of the Visual Studio installer dialog box, and select **NuGet package manager** if it is not already enabled.



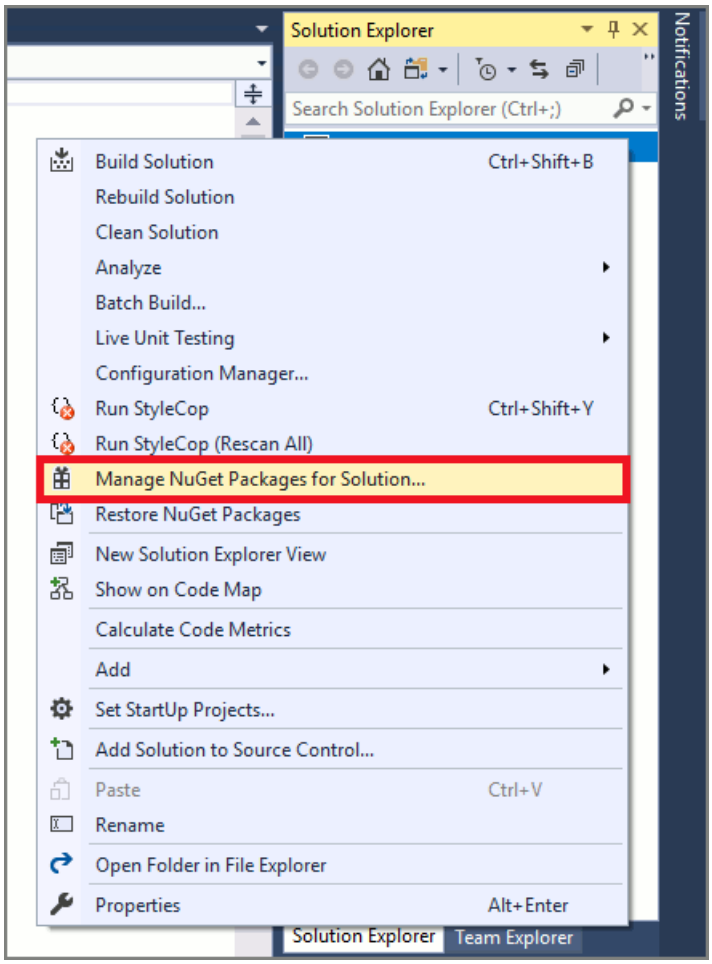
4. If you needed to enable either the C++ workload or NuGet, select **Modify** (at the lower right corner of the dialog box). Installation of the new features takes a moment. If both features were already enabled, close the dialog box instead.
5. Create a new Visual C++ Windows Desktop Windows Console Application. First, choose **File** > **New** > **Project** from the menu. In the **New Project** dialog box, expand **Installed** > **Visual C++** > **Windows Desktop** in the left pane. Then select **Windows Console Application**. For the project name, enter *helloworld*.



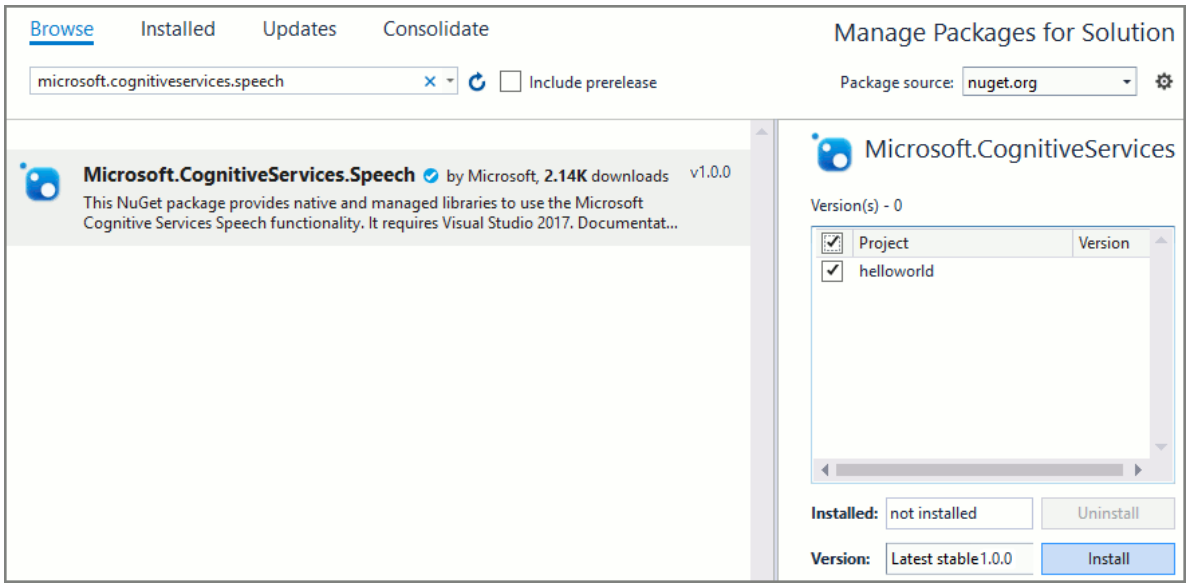
6. If you're running 64-bit Windows, you may switch your build platform to `x64` by using the drop-down menu in the Visual Studio toolbar. (64-bit versions of Windows can run 32-bit applications, so this is not a requirement.)



7. In Solution Explorer, right-click the solution and choose **Manage NuGet Packages for Solution**.



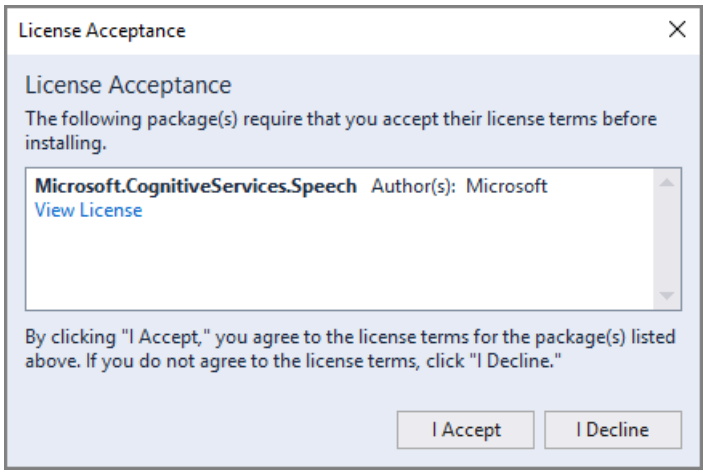
8. In the upper-right corner, in the **Package Source** field, select **nuget.org**. Search for the `Microsoft.CognitiveServices.Speech` package, and install it into the **helloworld** project.



NOTE

The current version of the Cognitive Services Speech SDK is `1.0.1`.

9. Accept the displayed license to begin installation of the NuGet package.



After the package is installed, a confirmation appears in the Package Manager console.

Add sample code

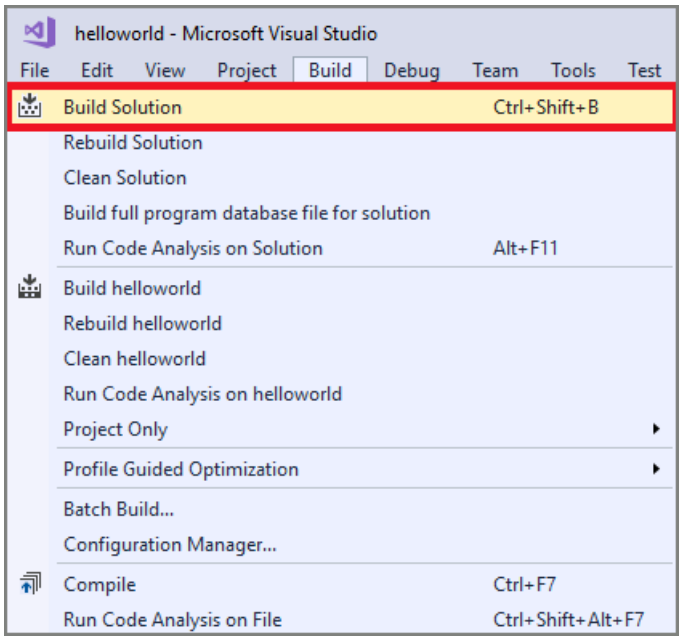
1. Open the source file *helloworld.cpp*. Replace all the code below the initial include statement (`#include "stdafx.h"` or `#include "pch.h"`) with the following:

```
#include <iostream>
#include <speechapi_cxx.h>

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

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

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

    // Performs recognition. RecognizeOnceAsync() returns when the first utterance has been
recognized,
    // so it is suitable only for single shot recognition like command or query. For long-running
    // recognition, use StartContinuousRecognitionAsync() instead.
    auto result = recognizer->RecognizeOnceAsync().get();

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

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

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

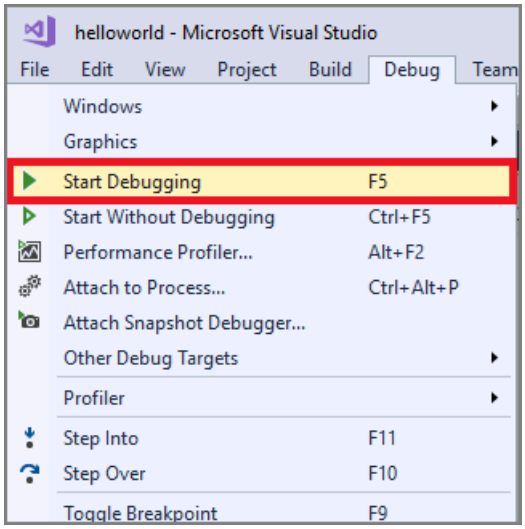
2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Replace the string `YourServiceRegion` with the region associated with your subscription (for example, `westus` for the free trial subscription).
4. Save changes to the project.

Build and run the app

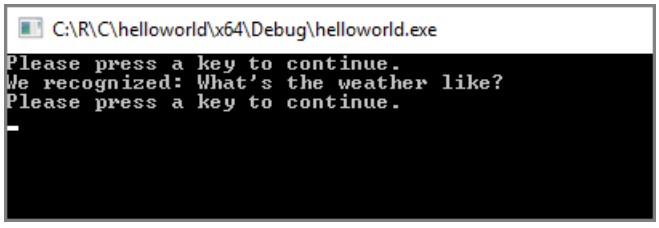
1. Build the application. From the menu bar, choose **Build** > **Build Solution**. The code should compile without errors.



2. Start the application. From the menu bar, choose **Debug** > **Start Debugging**, or press **F5**.



3. A console window appears, prompting you to say something. Speak an English phrase or sentence. Your speech is transmitted to the Speech service and transcribed to text, which appears in the same window.



Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/cpp-windows` folder.

Next steps

Recognize intents from speech by using the Speech SDK for C++

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in C++ on Linux by using the Speech SDK

10/19/2018 • 4 minutes to read • Edit Online

In this article, you create a C++ console application for Ubuntu Linux 16.04. You use the Cognitive Services Speech SDK to transcribe speech to text in real time from your PC's microphone. The application is built with the Speech SDK for Linux and your Linux distribution's C++ compiler (for example, `g++`).

Prerequisites

You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech service for free for details.

Install Speech SDK

> **IMPORTANT**
> By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

The current version of the Cognitive Services Speech SDK is `1.0.1`.

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 tarfile from https://aka.ms/csspeech/linuxbinary.

Download and install the SDK as follows:

1. Make sure the SDK's dependencies are installed.

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libcurl3 libasound2 wget
```

2. 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"
```

3. Create the directory if it doesn't exist yet.

```
mkdir -p "$SPEECHSDK_ROOT"
```

4. 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"
```

5. 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
`third-party-notices.md`	Third-party notices.
`include`	The required header files for C and C++
`lib/x64`	Native library for x64 required to link your application
`lib/x86`	Native library for x86 required to link your application

Add sample code

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

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

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

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

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

    // Performs recognition. RecognizeOnceAsync() returns when the first utterance has been
recognized,
    // so it is suitable only for single shot recognition like command or query. For long-running
    // recognition, use StartContinuousRecognitionAsync() instead.
    auto result = recognizer->RecognizeOnceAsync().get();

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

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

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

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

Build the app

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

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

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L
"$SPEECHSDK_ROOT/lib/x64" -l:libssl.so.1.0.0 -l:libcurl.so.4 -l:libasound.so.2
```

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

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L
"$SPEECHSDK_ROOT/lib/x86" -l:libssl.so.1.0.0 -l:libcurl.so.4 -l:libasound.so.2
```

Run the app

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

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

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

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

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

2. Run the application.

   ```
   ./helloworld
   ```

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

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

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/cpp-linux` folder.

Next steps

Recognize intents from speech by using the Speech SDK for C++

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in Java on Android by using the Speech SDK

10/19/2018 • 5 minutes to read • Edit Online

In this article, you'll learn how to create a Java application for Android using the Cognitive Services Speech SDK to transcribe speech to text. The application is based on the Microsoft Cognitive Services Speech SDK Maven Package, version 1.0.1, and Android Studio 3.1. The Speech SDK is currently compatible with Android devices having 32-bit or 64-bit ARM processors.

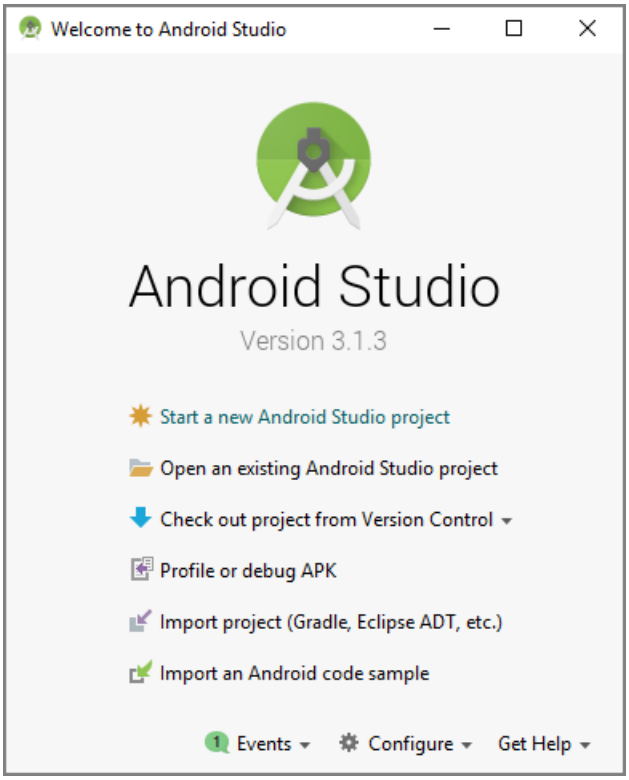
> **NOTE**
> For the Speech Devices SDK and the Roobo device, see Speech Devices SDK.

Prerequisites

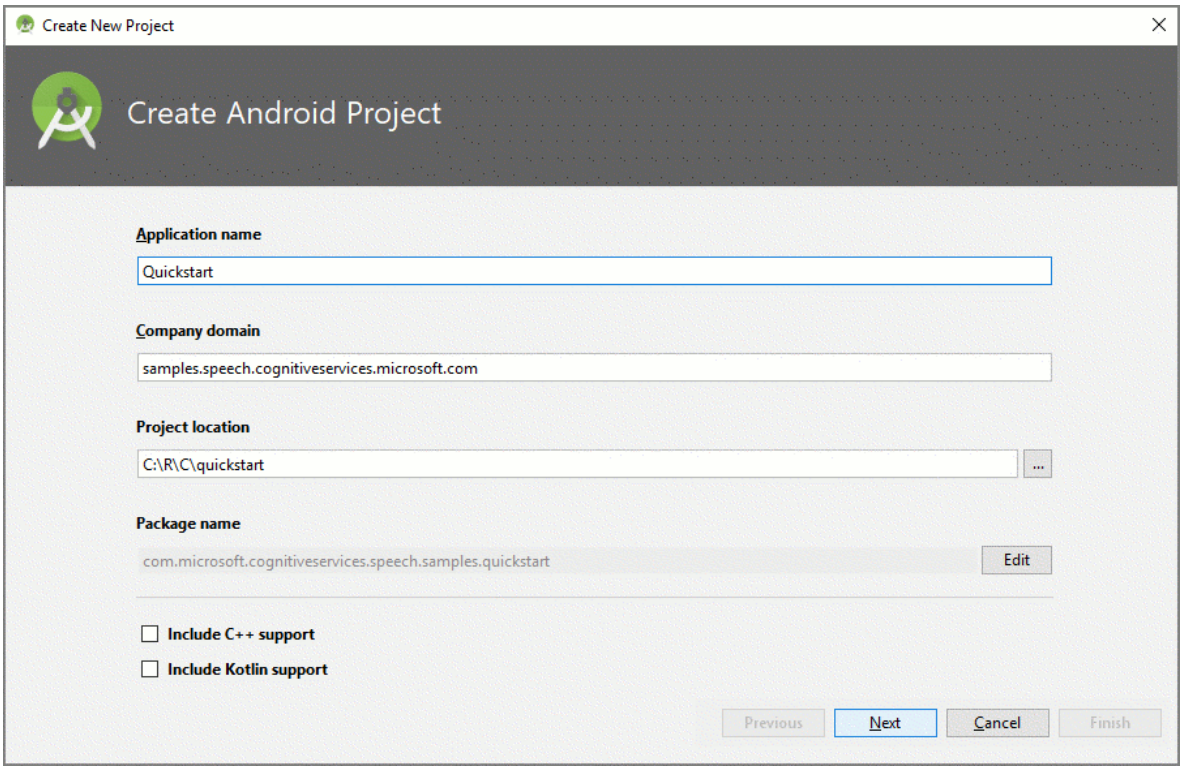
You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech service for free for details.

Create and configure a project

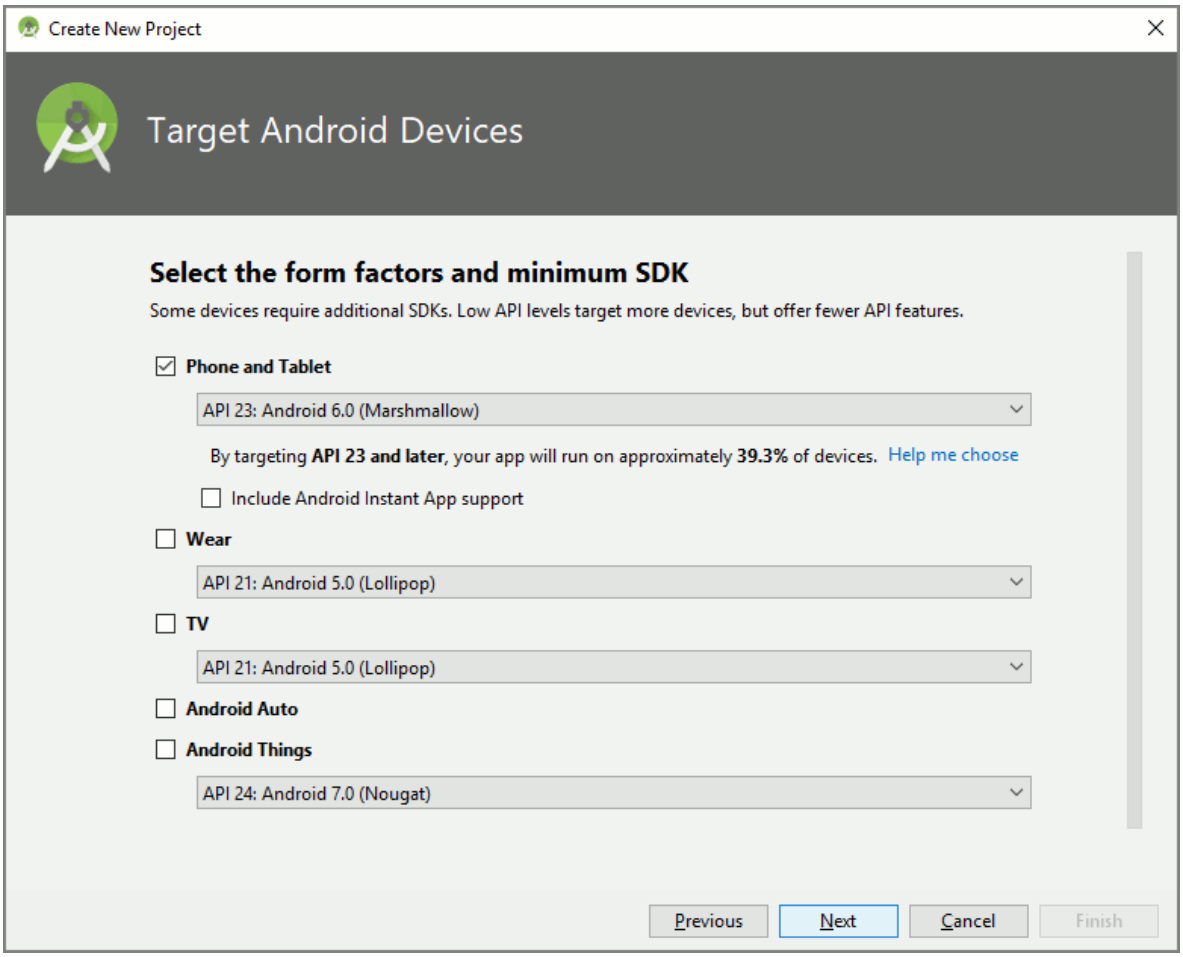
1. Launch Android Studio, and choose **Start a new Android Studio project** in the Welcome window.



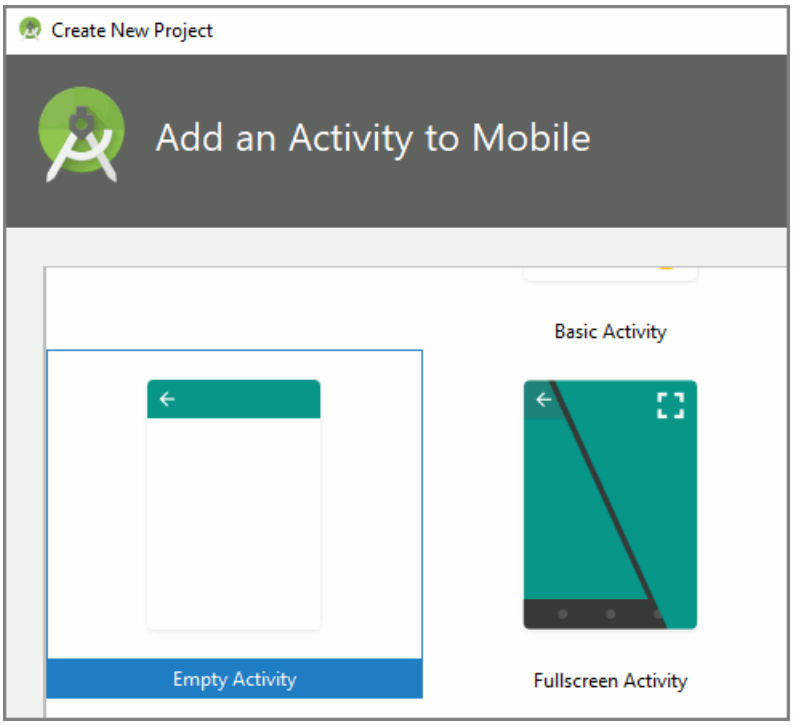
2. The **Create New Project** wizard appears. In the **Create Android Project** screen, enter **Quickstart** as **application name**, **samples.speech.cognitiveservices.microsoft.com** as **company domain**, and choose a project directory. Leave the C++ and Kotlin check boxes unchecked, and select **Next**.



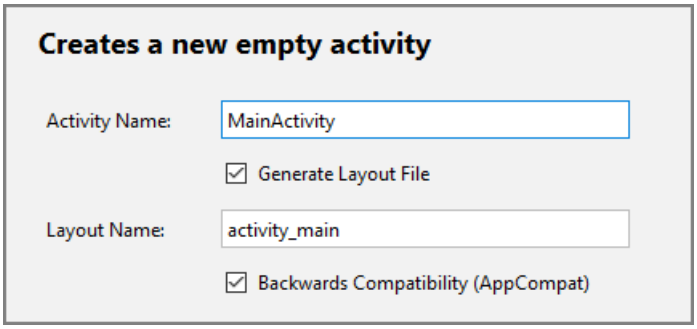
3. In the **Target Android Devices** screen, select only **Phone and Tablet**. In the drop-down list below it, choose **API 23: Android 6.0 (Marshmallow)**, and select **Next**.



4. In the **Add an Activity to Mobile** screen, select **Empty Activity**, and click **Next**.



5. In the **Configure Activity** screen, use **MainActivity** as the activity name and **activity_main** as the layout name. Select both check boxes, and select **Finish**.



Android Studio takes a moment to prepare your new Android project. Next, configure the project to know about the Speech SDK and to use Java 8.

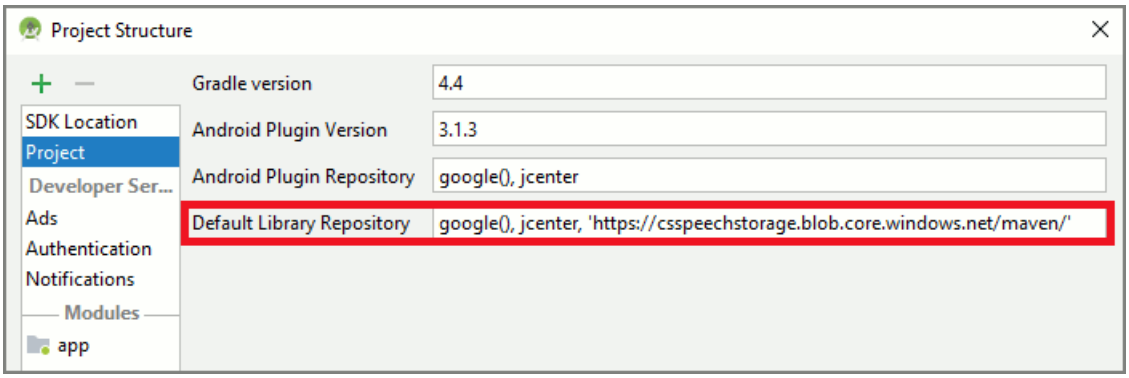
> **IMPORTANT**
> By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

The current version of the Cognitive Services Speech SDK is `1.0.1`.

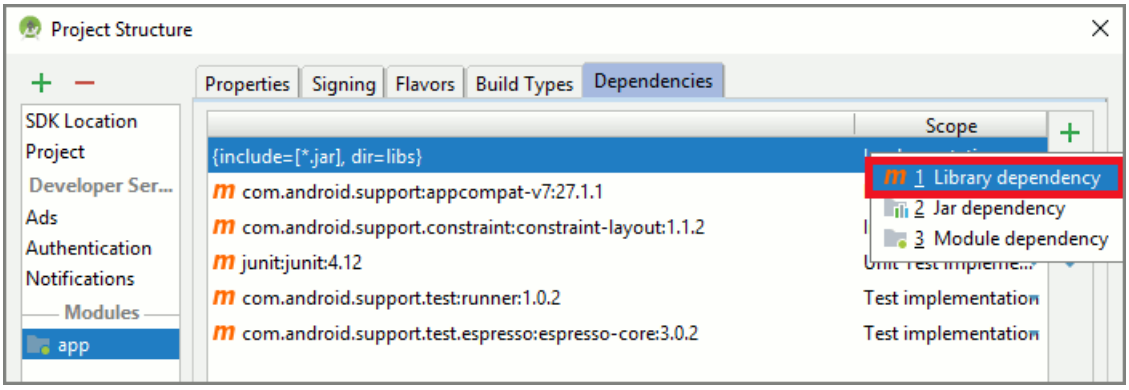
The Speech SDK for Android is packaged as an AAR (Android Library), which includes the necessary libraries as well as required Android permissions for using it. It is hosted in a Maven repository at https://csspeechstorage.blob.core.windows.net/maven/.

Set up your project to use the Speech SDK. Open the Project Structure window by choosing **File** > **Project Structure** from the Android Studio menu bar. In the Project Structure window, make the following changes:

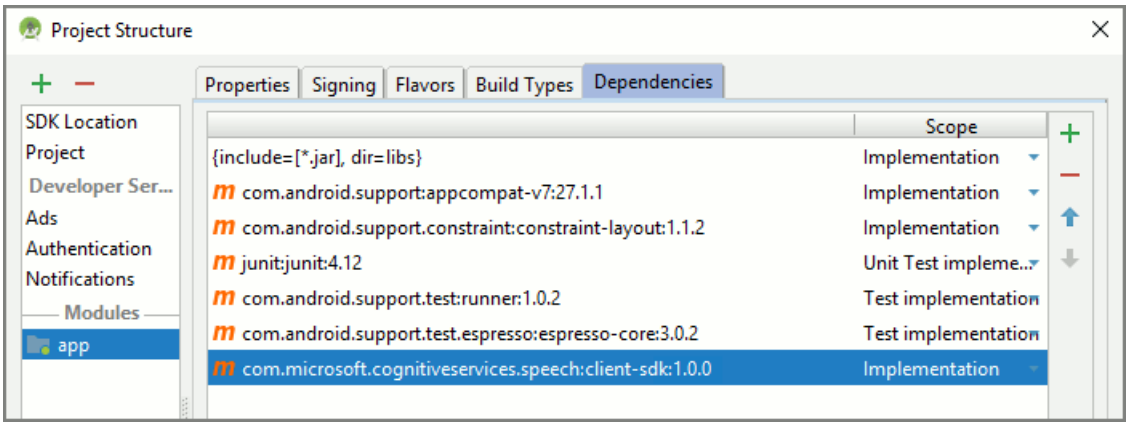
1. In the list on the left side of the window, select **Project**. Edit the **Default Library Repository** settings by appending a comma and our Maven repository URL enclosed in single quotes. 'https://csspeechstorage.blob.core.windows.net/maven/'



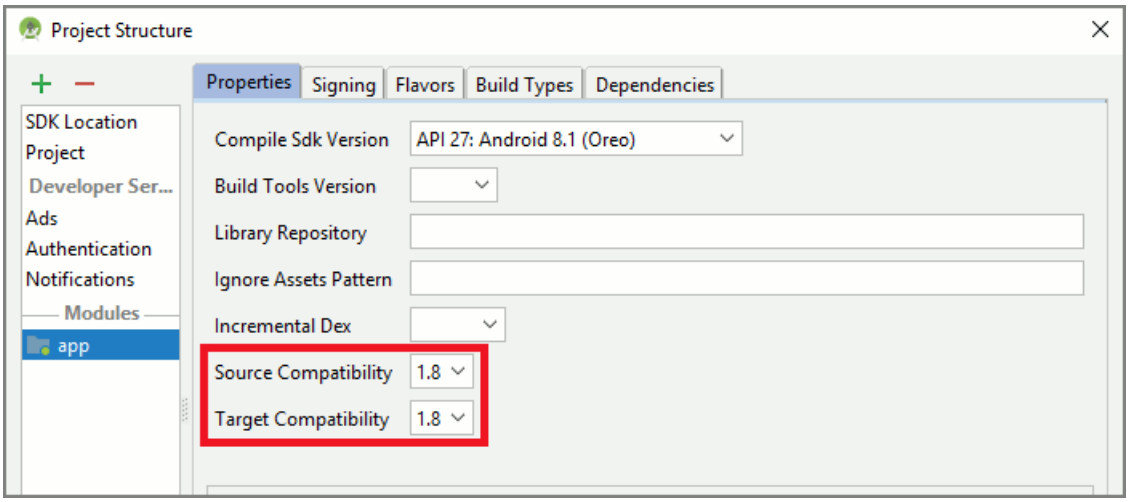
2. In the same screen, on the left side, select **app**. Then select the **Dependencies** tab at the top of the window. Select the green plus sign (+), and choose **Library dependency** from the drop-down menu.



3. In the window that comes up, enter the name and version of our Speech SDK for Android, `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1`. Then select **OK**. The Speech SDK should be added to the list of dependencies now, as shown below:



4. Select the **Properties** tab. For both **Source Compatibility** and **Target Compatibility**, select **1.8**.

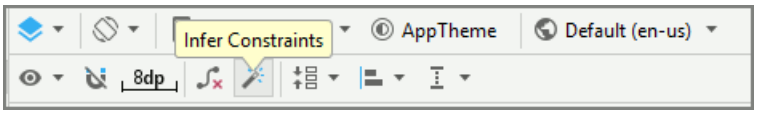


5. Select **OK** to close the Project Structure window and apply your changes to the project.

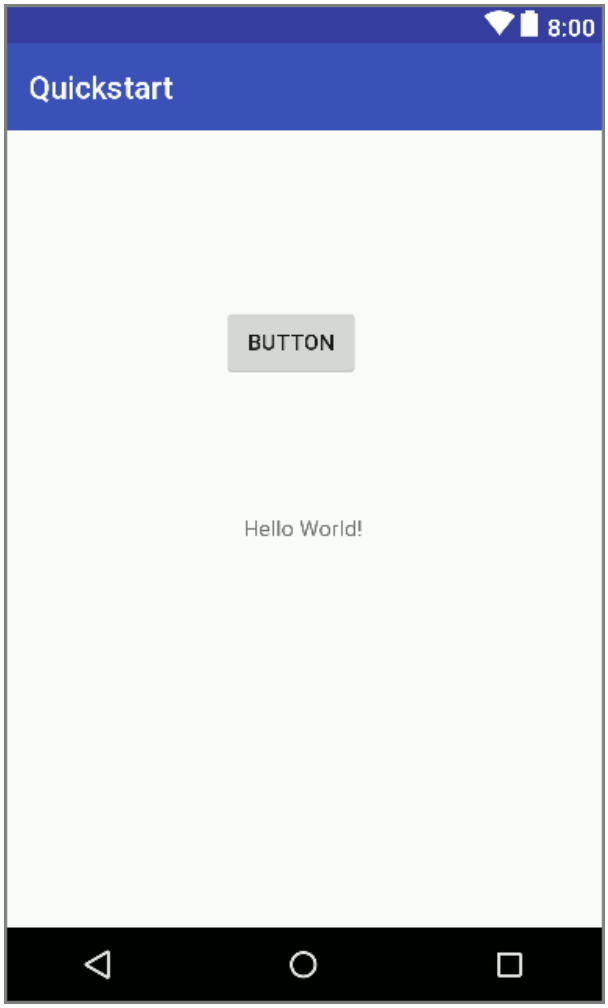
Create user interface

We will create a basic user interface for the application. Edit the layout for your main activity, `activity_main.xml`. Initially, the layout includes a title bar with your application's name, and a TextView containing the text "Hello World!"

- Click the TextView element. Change its ID attribute in the upper-right corner to `hello`.
- From the Palette in the upper left of the `activity_main.xml` window, drag a button into the empty space above the text.
- In the button's attributes on the right, in the value for the `onClick` attribute, enter `onSpeechButtonClicked`. We'll write a method with this name to handle the button event. Change its ID attribute in the upper-right corner to `button`.
- Use the magic wand icon at the top of the designer to infer layout constraints.



The text and graphical representation of your UI should now look like this.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/hello"
        android:layout_width="366dp"
        android:layout_height="295dp"
        android:text="Hello World!"

app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toTopOf="parent"

app:layout_constraintVertical_bias="0.925" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:onClick="onSpeechButtonClicked"
        android:text="Button"

app:layout_constraintBottom_toTopOf="@+id/hello"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="parent"

app:layout_constraintVertical_bias="0.072" />

</android.support.constraint.ConstraintLayout>
```

Add sample code

1. Open the source file `MainActivity.java`. Replace all the code in this file with the following.

```
package com.microsoft.cognitiveservices.speech.samples.quickstart;

import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

import com.microsoft.cognitiveservices.speech.ResultReason;
import com.microsoft.cognitiveservices.speech.SpeechConfig;
import com.microsoft.cognitiveservices.speech.SpeechRecognitionResult;
import com.microsoft.cognitiveservices.speech.SpeechRecognizer;

import java.util.concurrent.Future;

import static android.Manifest.permission.*;
```

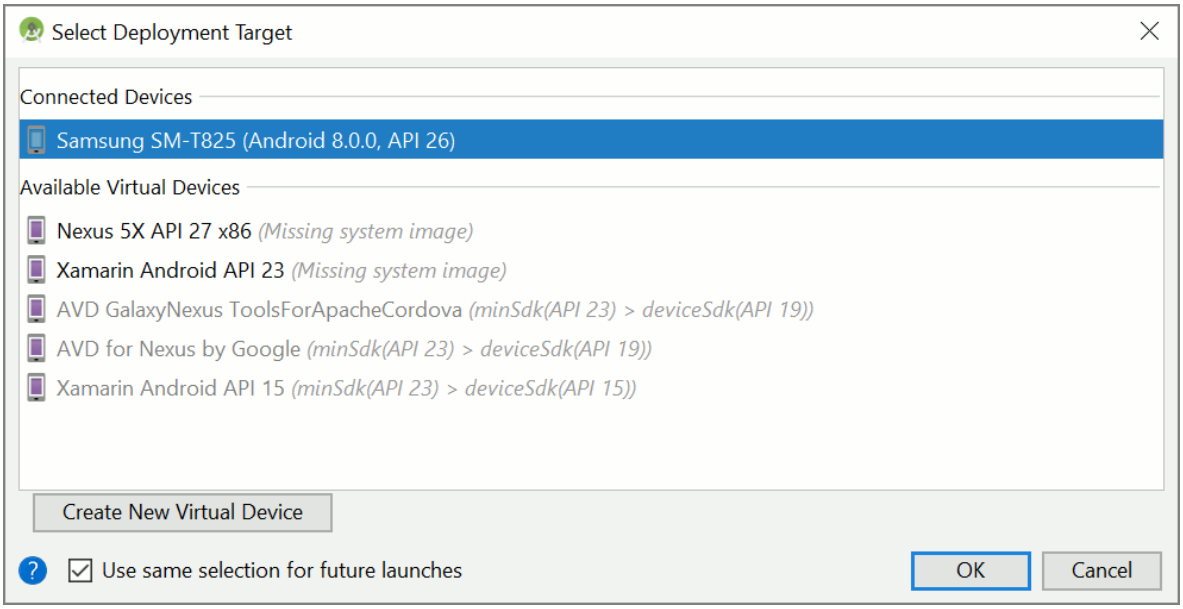
```
public class MainActivity extends AppCompatActivity {

    // Replace below with your own subscription key
    private static String speechSubscriptionKey = "YourSubscriptionKey";
    // Replace below with your own service region (e.g., "westus").
    private static String serviceRegion = "YourServiceRegion";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Note: we need to request the permissions
        int requestCode = 5; // unique code for the permission request
        ActivityCompat.requestPermissions(MainActivity.this, new String[]{RECORD_AUDIO, INTERNET},
requestCode);
    }

    public void onSpeechButtonClicked(View v) {
        TextView txt = (TextView) this.findViewById(R.id.hello); // 'hello' is the ID of your text
view

        try {
            SpeechConfig config = SpeechConfig.fromSubscription(speechSubscriptionKey, serviceRegion);
            assert(config != null);

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

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

            // Note: this will block the UI thread, so eventually, you want to
            //        register for the event (see full samples)
            SpeechRecognitionResult result = task.get();
            assert(result != null);

            if (result.getReason() == ResultReason.RecognizedSpeech) {
                txt.setText(result.toString());
            }
            else {
                txt.setText("Error recognizing. Did you update the subscription info?" +
System.lineSeparator() + result.toString());
            }

            reco.close();
        } catch (Exception ex) {
            Log.e("SpeechSDKDemo", "unexpected " + ex.getMessage());
            assert(false);
        }
    }
}
```

- The `onCreate` method includes code that requests microphone and internet permissions, and initializes the native platform binding. Configuring the native platform bindings is only required once. It should be done early during application initialization.
- The method `onSpeechButtonClicked` is, as noted earlier, the button click handler. A button press triggers speech to text transcription.

2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Also replace the string `YourServiceRegion` with the region associated with your subscription (for example, `westus` for the free trial subscription).

Build and run the app

1. Connect your Android device to your development PC. Make sure you have enabled development mode and USB debugging on the device.
2. To build the application, press Ctrl+F9, or choose **Build** > **Make Project** from the menu bar.
3. To launch the application, press Shift+F10, or choose **Run** > **Run 'app'**.
4. In the deployment target window that appears, choose your Android device.



Press the button in the application to begin a speech recognition section. The next 15 seconds of English speech will be sent to the Speech service and transcribed. The result appears in the Android application, and in the logcat window in Android Studio.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/java-android` folder.

Next steps

Recognize intents from speech by using the Speech SDK for Java

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in Java on Windows or Linux by using the Speech Service SDK

10/19/2018 • 3 minutes to read • Edit Online

In this article, you create a Java console application by using the Speech Service SDK. You transcribe speech to text in real time from your PC's microphone. The application is built with the Speech SDK Maven package, and the Eclipse Java IDE (v4.8) on 64-bit Windows or Ubuntu Linux 16.04. It runs on a 64-bit Java 8 runtime environment (JRE).

> **NOTE**
> For the Speech Devices SDK and the Roobo device, see Speech Devices SDK.

Prerequisites

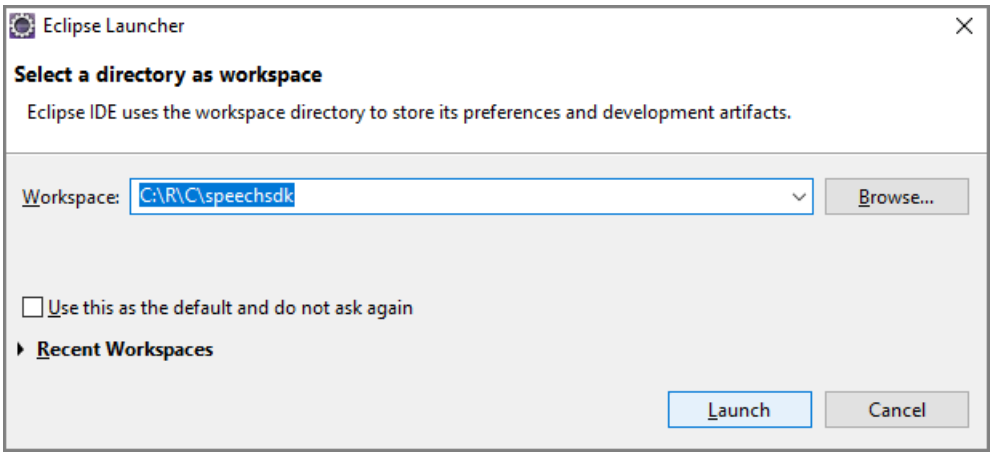
You need a Speech service subscription key to complete this Quickstart. You can get one for free. See Try the Speech Service for free for details.

Create and configure project

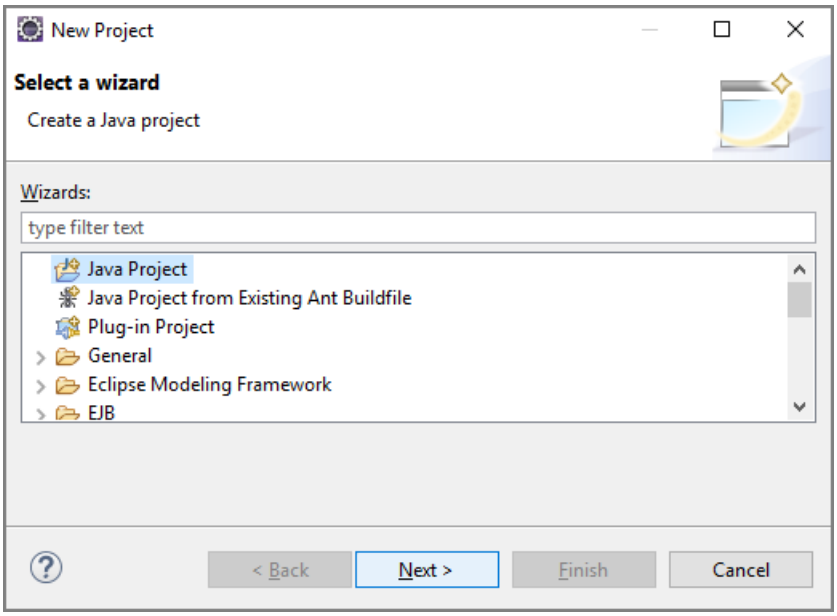
If you are using Ubuntu 16.04, before starting Eclipse, run the following commands to make sure that required packages are installed.

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libcurl3 libasound2 wget
```

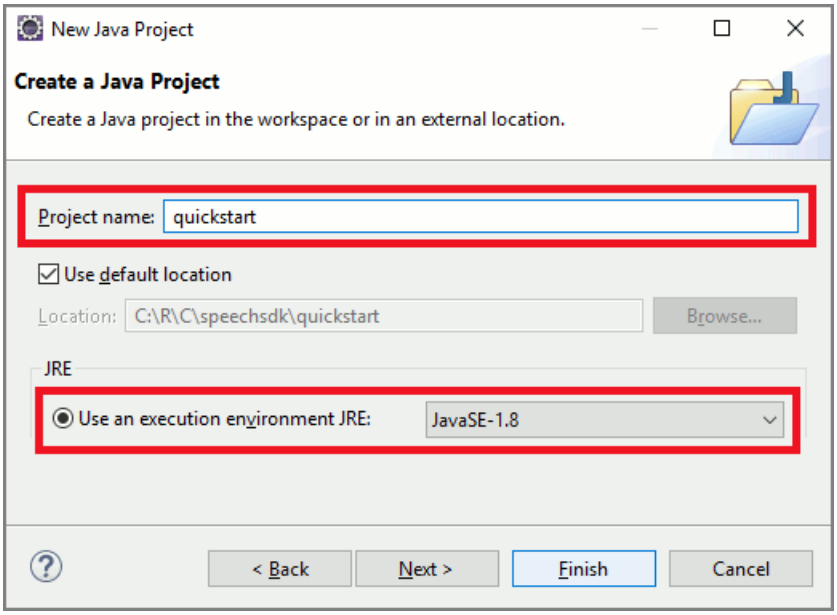
1. Start Eclipse.
2. In the Eclipse Launcher, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.



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**.

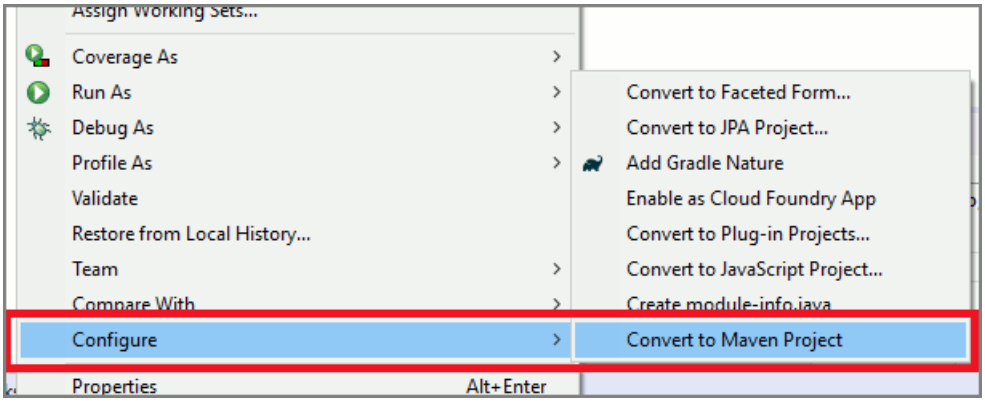


6. The New Java Project wizard starts. In the **Project name** field, enter **quickstart**, and choose **JavaSE-1.8** as the execution environment. Select **Finish**.

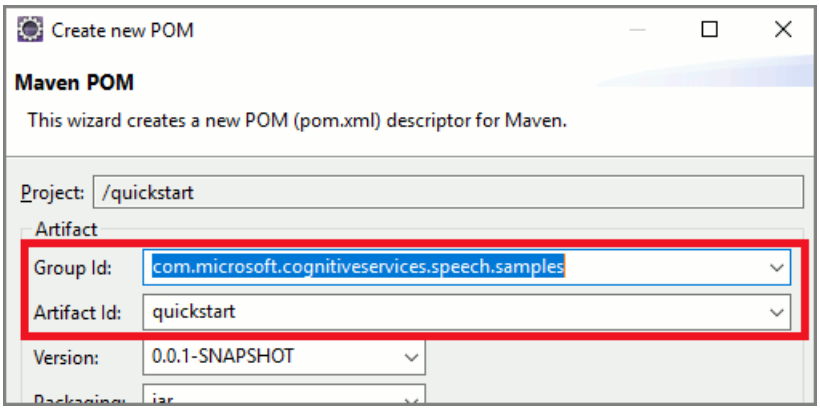


7. If the **Open Associated Perspective?** window appears, select **Open Perspective**.

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 1.0.1 as a dependency:

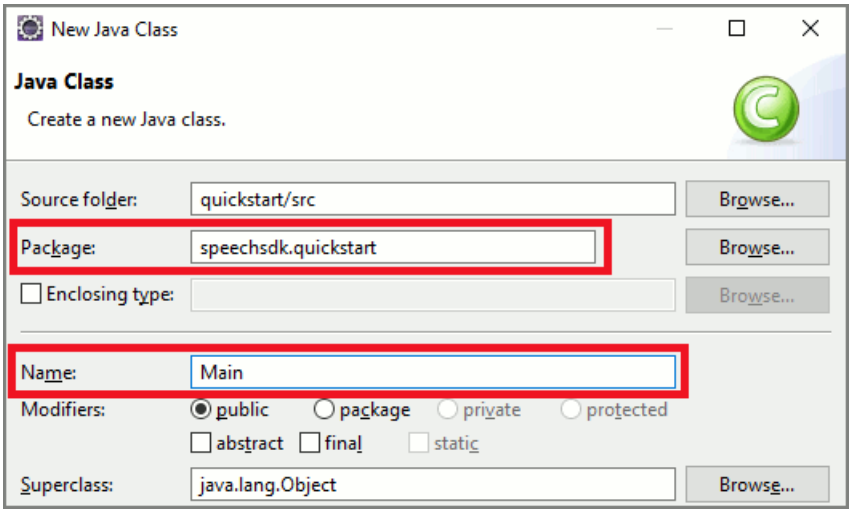
      ```
      <dependencies>
        <dependency>
          <groupId>com.microsoft.cognitiveservices.speech</groupId>
          <artifactId>client-sdk</artifactId>
          <version>1.0.1</version>
        </dependency>
      </dependencies>
      ```

    - Save the changes.

Add sample code

1. To add a new empty class to your Java project, select **File** > **New** > **Class**.

2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



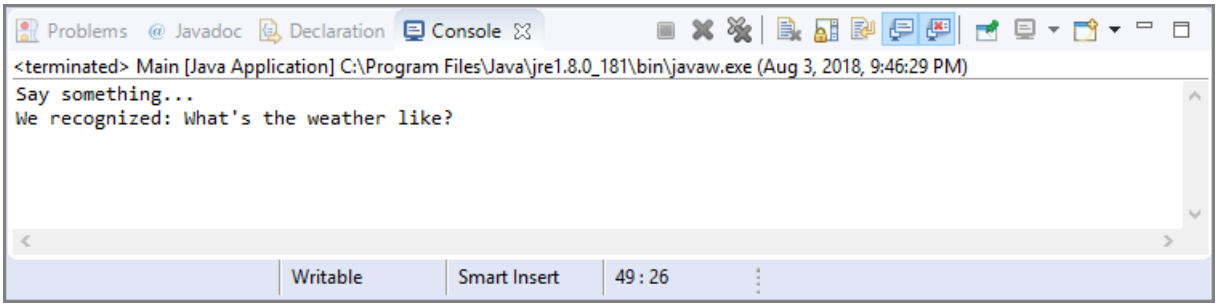
3. Replace all code in `Main.java` with the following snippet:

```
package speechsdk.quickstart;

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

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

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

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

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

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

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

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

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

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

            reco.close();

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

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

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

Build and run the app

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



Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/java-jre` folder.

Next steps

Recognize intents from speech by using the Speech SDK for Java

See also

- Translate speech
- Customize acoustic models
- Customize language models

Quickstart: Recognize speech in JavaScript in a browser using the Speech Service SDK

10/19/2018 • 4 minutes to read • Edit Online

In this article, you'll learn how to create a website using the JavaScript binding of the Cognitive Services Speech SDK to transcribe speech to text. The application is based on the Microsoft Cognitive Services Speech SDK (Download version 1.0.1).

Prerequisites

- A subscription key for the Speech service. See Try the Speech Service for free.
- A PC or Mac, with a working microphone.
- A text editor.
- A current version of Chrome or Microsoft Edge.
- Optionally, a web server that supports hosting PHP scripts.

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

> **IMPORTANT**
>
> By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

Download the Speech SDK as a .zip package and unpack it into the newly created folder. This should result in two files being unpacked, i.e., `microsoft.cognitiveservices.speech.sdk.bundle.js` and `microsoft.cognitiveservices.speech.sdk.bundle.js.map`. The latter file is optional and used to help debugging into SDK code, if necessary.

Create an index.html page

Create a new file in the folder, named `index.html` and open this file with a text editor.

1. Create the following HTML skeleton:

```
<html>
<head>
   <title>Microsoft Cognitive Service Speech SDK JavaScript Quickstart</title>
</head>
<body>
 <!-- UI code goes here -->

 <!-- SDK reference goes here -->

 <!-- Optional authorization token request goes here -->

 <!-- Sample code goes here -->
</body>
</html>
```

2. Add the following UI code to your file, below the first comment:

```
<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/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="regionKey" type="text" size="40" value="YourSubscriptionServiceRegion"></td>
    </tr>
    <tr>
      <td></td>
      <td><button id="startRecognizeOnceAsyncButton">Start recognition</button></td>
    </tr>
    <tr>
      <td align="right" valign="top">Results</td>
      <td><textarea id="phraseDiv" style="display: inline-block;width:500px;height:200px"></textarea>
</td>
    </tr>
  </table>
</div>
```

3. Add a reference to the Speech SDK

```
<!-- Speech SDK reference sdk. -->
<script src="microsoft.cognitiveservices.speech.sdk.bundle.js"></script>
```

4. Wire up handlers for the recognition button, recognition result, and subscription related fields defined by the UI code:

```
<!-- Speech SDK USAGE -->
<script>
  // status fields and start button in UI
  var phraseDiv;
```

```
  var startRecognizeOnceAsyncButton;

  // subscription key and region key for speech services.
  var subscriptionKey, regionKey;
  var authorizationToken;
  var SpeechSDK;
  var recognizer;

  document.addEventListener("DOMContentLoaded", function () {
    startRecognizeOnceAsyncButton = document.getElementById("startRecognizeOnceAsyncButton");
    subscriptionKey = document.getElementById("subscriptionKey");
    regionKey = document.getElementById("regionKey");
    phraseDiv = document.getElementById("phraseDiv");

    startRecognizeOnceAsyncButton.addEventListener("click", function () {
      startRecognizeOnceAsyncButton.disabled = true;
      phraseDiv.innerHTML = "";

      // if we got an authorization token, use the token. Otherwise use the provided subscription key
      var speechConfig;
      if (authorizationToken) {
        speechConfig = SpeechSDK.SpeechConfig.fromAuthorizationToken(authorizationToken,
regionKey.value);
      } else {
        if (subscriptionKey.value === "" || subscriptionKey.value === "subscription") {
          alert("Please enter your Microsoft Cognitive Services Speech subscription key!");
          return;
        }
        speechConfig = SpeechSDK.SpeechConfig.fromSubscription(subscriptionKey.value,
regionKey.value);
      }

      speechConfig.speechRecognitionLanguage = "en-US";
      var audioConfig  = SpeechSDK.AudioConfig.fromDefaultMicrophoneInput();
      recognizer = new SpeechSDK.SpeechRecognizer(speechConfig, audioConfig);

      recognizer.recognizeOnceAsync(
        function (result) {
          startRecognizeOnceAsyncButton.disabled = false;
          phraseDiv.innerHTML += result.text;
          window.console.log(result);

          recognizer.close();
          recognizer = undefined;
        },
        function (err) {
          startRecognizeOnceAsyncButton.disabled = false;
          phraseDiv.innerHTML += err;
          window.console.log(err);

          recognizer.close();
          recognizer = undefined;
        });
    });

    if (!!window.SpeechSDK) {
      SpeechSDK = window.SpeechSDK;
      startRecognizeOnceAsyncButton.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 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.

1. Create a new file named `token.php`. In this example we assume your web server supports the PHP scripting language. 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 = 'YourSubscriptionServiceRegion';

$ch = curl_init();
curl_setopt($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);
?>
```

2. Edit the `index.html` file and add the following code to your file:

```
<!-- Speech SDK Authorization token -->
<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]));
        regionKey.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>
```

> **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 subscription key and region and trigger a recognition using 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 region, and trigger a recognition using the microphone. If configured, it will acquire a token from your token source.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/js-browser` folder.

Next steps

Get our samples

Quickstart: Recognize speech in Objective-C on iOS using the Speech Service SDK

10/19/2018 • 7 minutes to read • Edit Online

In this article, you learn how to create an iOS app in Objective-C using the Cognitive Services Speech SDK to transcribe an audio file with recorded speech to text.

Prerequisites

- A subscription key for the Speech service. See Try the Speech Service for free.
- A Mac with Xcode 9.4.1 installed as iOS development environment. This tutorial targets iOS versions 11.4. If you don't have Xcode yet, you can install it from the App Store.

Get the Speech SDK for iOS

> **IMPORTANT**
> By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

The current version of the Cognitive Services Speech SDK is `1.0.1`.

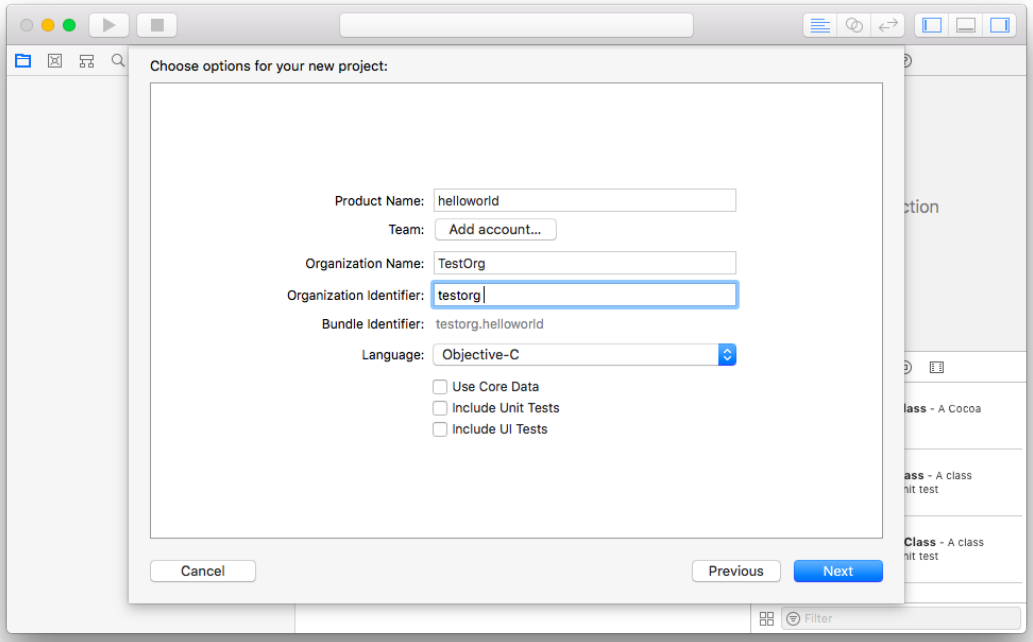
The Cognitive Services Speech SDK for Mac and iOS is currently distributed as a Cocoa Framework. It can be downloaded from https://aka.ms/csspeech/iosbinary. Download the file to your home directory.

Create an Xcode Project

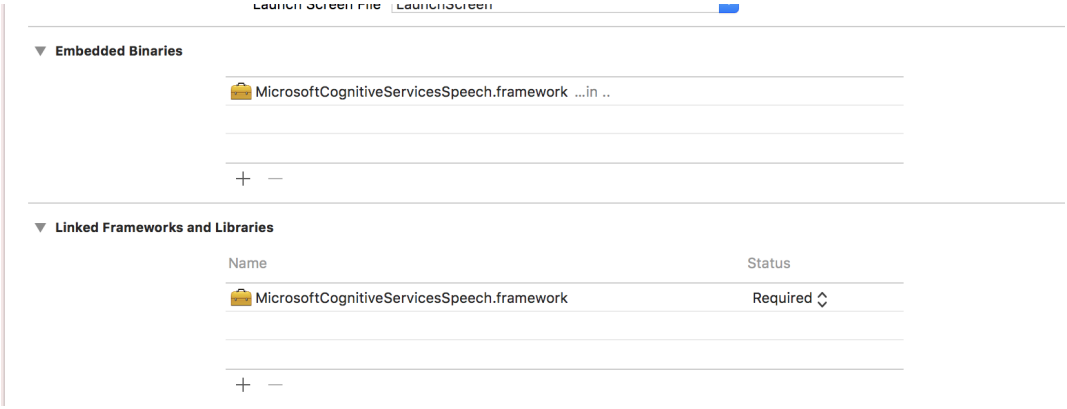
Start Xcode, and start a new project by clicking **File** > **New** > **Project**. In the template selection dialog, choose the "iOS Single View App" template.

In the dialogs that follow, make the following selections:

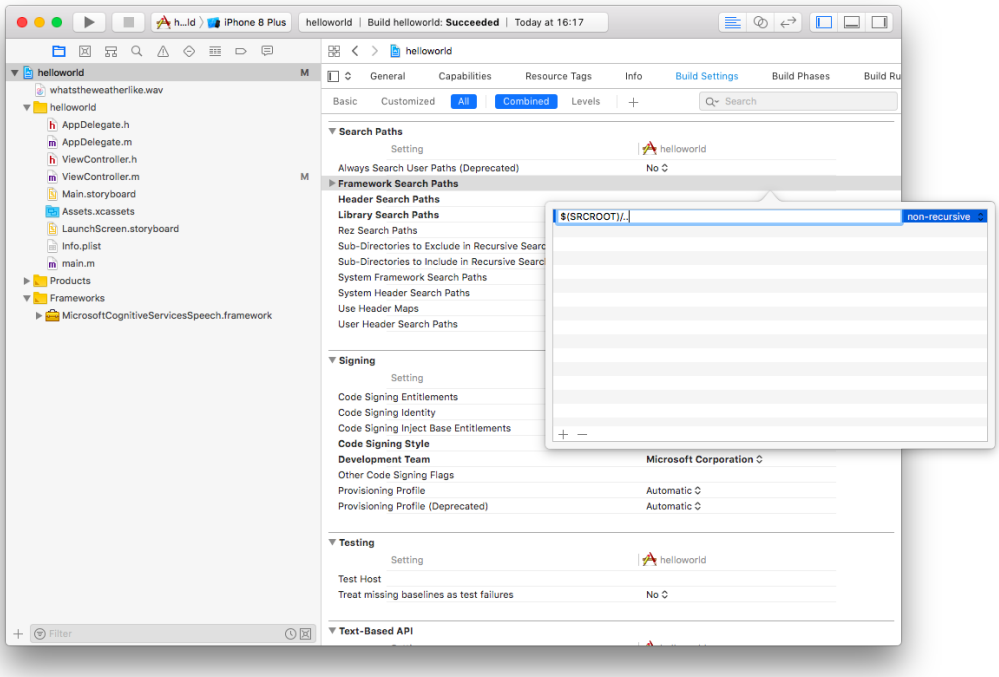
1. Project Options Dialog
   a. Enter a name for the quickstart app, for example `helloworld`.
   b. Enter an appropriate organization name and organization identifier, if you already have an Apple developer account. For testing purposes, you can just pick any name like `testorg`. In order to sign the app, you also need a proper provisioning profile. Please refer to the Apple developer site for details.
   c. Make sure Objective-C is chosen as the language for the project.
   d. Disable all checkboxes for tests and core data.



2. Select project directory
   a. Choose your home directory to put the project in. This will create a `helloworld` directory in your home directory that contains all the files for the Xcode project.
   b. Disable the creation of a Git repo for this example project.
   c. Adjust the paths to the SDK in the *Project Settings*.
      a. In the **General** tab under the **Embedded Binaries** header, add the SDK library as a framework: **Add embedded binaries** > **Add other...** > Navigate to your home directory and choose the file `MicrosoftCognitiveServicesSpeech.framework`. This will also automatically add the SDK library to the header **Linked Framework and Libraries**.



      b. Go to the **Build Settings** tab and activate **All** settings.
      c. Add the directory `$(SRCROOT)/..` to the *Framework Search Paths* under the **Search Paths** heading.



Set up the UI

The example app will have a very simple UI: Two buttons to start speech recognition either from file or from microphone input, and a text label to display the result. The UI is set up in the `Main.storyboard` part of the project. Open the XML view of the storyboard by right-clicking the `Main.storyboard` entry of the project tree and selecting **Open As...** > **Source Code**. Replace the autogenerated XML with this:

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0" toolsVersion="14113"
targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES"
useSafeAreas="YES" colorMatched="YES" initialViewController="BYZ-38-t0r">
    <device id="retina4_7" orientation="portrait">
        <adaptation id="fullscreen"/>
    </device>
    <dependencies>
        <deployment identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="14088"/>
        <capability name="Safe area layout guides" minToolsVersion="9.0"/>
        <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
    </dependencies>
    <scenes>
        <!--View Controller-->
        <scene sceneID="tne-QT-ifu">
            <objects>
                <viewController id="BYZ-38-t0r" customClass="ViewController" sceneMemberID="viewController">
                    <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
                        <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <subviews>
                            <button opaque="NO" contentMode="scaleToFill" fixedFrame="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="qFP-u7-47Q">
                                <rect key="frame" x="84" y="247" width="207" height="82"/>
                                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES"
flexibleMaxY="YES"/>
                                <accessibility key="accessibilityConfiguration" hint="Start speech
recognition from file" identifier="recognize_file_button">
                                    <accessibilityTraits key="traits" button="YES" staticText="YES"/>
                                    <bool key="isElement" value="YES"/>
                                </accessibility>
                                <fontDescription key="fontDescription" type="system" pointSize="30"/>
```

```
                                <fontDescription key="fontDescription" type="system" pointSize="30" />
                                <state key="normal" title="Recognize (File)"/>
                                <connections>
                                    <action selector="recognizeFromFileButtonTapped:" destination="BYZ-38-
t0r" eventType="touchUpInside" id="Vfr-ah-nbC"/>
                                </connections>
                            </button>
                            <label opaque="NO" userInteractionEnabled="NO" contentMode="center"
horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Recognition result"
textAlignment="center" lineBreakMode="tailTruncation" numberOfLines="5" baselineAdjustment="alignBaselines"
adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="tq3-GD-ljB">
                                <rect key="frame" x="20" y="408" width="335" height="148"/>
                                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES"
flexibleMaxY="YES"/>
                                <accessibility key="accessibilityConfiguration" hint="The result of speech
recognition" identifier="result_label">
                                    <accessibilityTraits key="traits" notEnabled="YES"/>
                                    <bool key="isElement" value="NO"/>
                                </accessibility>
                                <fontDescription key="fontDescription" type="system" pointSize="30"/>
                                <color key="textColor" red="0.5" green="0.5" blue="0.5" alpha="1"
colorSpace="custom" customColorSpace="sRGB"/>
                                <nil key="highlightedColor"/>
                            </label>
                            <button opaque="NO" contentMode="scaleToFill" fixedFrame="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="91d-Ki-IyR">
                                <rect key="frame" x="16" y="209" width="339" height="30"/>
                                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES"
flexibleMaxY="YES"/>
                                <accessibility key="accessibilityConfiguration" hint="Start speech
recognition from microphone" identifier="recognize_microphone_button"/>
                                <fontDescription key="fontDescription" type="system" pointSize="30"/>
                                <state key="normal" title="Recognize (Microphone)"/>
                                <connections>
                                    <action selector="recognizeFromMicButtonTapped:" destination="BYZ-38-
t0r" eventType="touchUpInside" id="2n3-kA-ySa"/>
                                </connections>
                            </button>
                        </subviews>
                        <color key="backgroundColor" red="1" green="1" blue="1" alpha="1"
colorSpace="custom" customColorSpace="sRGB"/>
                        <viewLayoutGuide key="safeArea" id="6Tk-OE-BBY"/>
                    </view>
                    <connections>
                        <outlet property="recognitionResultLabel" destination="tq3-GD-ljB" id="kP4-o4-s0Q"/>
                    </connections>
                </viewController>
                <placeholder placeholderIdentifier="IBFirstResponder" id="dkx-z0-nzr"
sceneMemberID="firstResponder"/>
            </objects>
            <point key="canvasLocation" x="135.19999999999999" y="132.68365817091455"/>
        </scene>
    </scenes>
</document>
```

Add the sample code

1. Download the sample wav file by right-clicking the link and choosing **Save target as...**. Add the wav file to the project as a resource by dragging it from a Finder window into the root level of the Project view. Click **Finish** in the following dialog without changing the settings.
2. Replace the contents of the autogenerated `ViewController.m` file by:

```
#import "ViewController.h"
#import <MicrosoftCognitiveServicesSpeech/SPXSpeechApi.h>
```

```
@interface ViewController () {
    NSString *speechKey;
    NSString *serviceRegion;
}

@property (weak, nonatomic) IBOutlet UIButton *recognizeFromFileButton;
@property (weak, nonatomic) IBOutlet UIButton *recognizeFromMicButton;
@property (weak, nonatomic) IBOutlet UILabel *recognitionResultLabel;
- (IBAction)recognizeFromFileButtonTapped:(UIButton *)sender;
- (IBAction)recognizeFromMicButtonTapped:(UIButton *)sender;
@end

@implementation ViewController

- (void)viewDidLoad {
    speechKey = @"YourSubscriptionKey";
    serviceRegion = @"YourServiceRegion";
}

- (IBAction)recognizeFromFileButtonTapped:(UIButton *)sender {
    dispatch_async(dispatch_get_global_queue(QOS_CLASS_DEFAULT, 0), ^{
        [self recognizeFromFile];
    });
}

- (IBAction)recognizeFromMicButtonTapped:(UIButton *)sender {
    dispatch_async(dispatch_get_global_queue(QOS_CLASS_DEFAULT, 0), ^{
        [self recognizeFromMicrophone];
    });
}

- (void)recognizeFromFile {
    NSBundle *mainBundle = [NSBundle mainBundle];
    NSString *weatherFile = [mainBundle pathForResource: @"whatstheweatherlike" ofType:@"wav"];
    NSLog(@"weatherFile path: %@", weatherFile);
    if (!weatherFile) {
        NSLog(@"Cannot find audio file!");
        [self updateRecognitionErrorText:(@"Cannot find audio file")];
        return;
    }

    SPXAudioConfiguration* weatherAudioSource = [[SPXAudioConfiguration alloc]
initWithWavFileInput:weatherFile];
    if (!weatherAudioSource) {
        NSLog(@"Loading audio file failed!");
        [self updateRecognitionErrorText:(@"Audio Error")];
        return;
    }

    SPXSpeechConfiguration *speechConfig = [[SPXSpeechConfiguration alloc]
initWithSubscription:speechKey region:serviceRegion];
    if (!speechConfig) {
        NSLog(@"Could not load speech config");
        [self updateRecognitionErrorText:(@"Speech Config Error")];
        return;
    }

    [self updateRecognitionStatusText:(@"Recognizing...")];

    SPXSpeechRecognizer* speechRecognizer = [[SPXSpeechRecognizer alloc]
initWithSpeechConfiguration:speechConfig audioConfiguration:weatherAudioSource];
    if (!speechRecognizer) {
        NSLog(@"Could not create speech recognizer");
        [self updateRecognitionResultText:(@"Speech Recognition Error")];
        return;
    }

    SPXSpeechRecognitionResult *speechResult = [speechRecognizer recognizeOnce];
```

```
    if (SPXResultReason_Canceled == speechResult.reason) {
        SPXCancellationDetails *details = [[SPXCancellationDetails alloc]
initFromCanceledRecognitionResult:speechResult];
        NSLog(@"Speech recognition was canceled: %@. Did you pass the correct key/region
combination?", details.errorDetails);
        [self updateRecognitionErrorText:([NSString stringWithFormat:@"Canceled: %@",
details.errorDetails ])];
    } else if (SPXResultReason_RecognizedSpeech == speechResult.reason) {
        NSLog(@"Speech recognition result received: %@", speechResult.text);
        [self updateRecognitionResultText:(speechResult.text)];
    } else {
        NSLog(@"There was an error.");
        [self updateRecognitionErrorText:(@"Speech Recognition Error")];
    }
}

- (void)recognizeFromMicrophone {
    SPXSpeechConfiguration *speechConfig = [[SPXSpeechConfiguration alloc]
initWithSubscription:speechKey region:serviceRegion];
    if (!speechConfig) {
        NSLog(@"Could not load speech config");
        [self updateRecognitionErrorText:(@"Speech Config Error")];
        return;
    }

    [self updateRecognitionStatusText:(@"Recognizing...")];

    SPXSpeechRecognizer* speechRecognizer = [[SPXSpeechRecognizer alloc] init:speechConfig];
    if (!speechRecognizer) {
        NSLog(@"Could not create speech recognizer");
        [self updateRecognitionResultText:(@"Speech Recognition Error")];
        return;
    }

    SPXSpeechRecognitionResult *speechResult = [speechRecognizer recognizeOnce];
    if (SPXResultReason_Canceled == speechResult.reason) {
        SPXCancellationDetails *details = [[SPXCancellationDetails alloc]
initFromCanceledRecognitionResult:speechResult];
        NSLog(@"Speech recognition was canceled: %@. Did you pass the correct key/region
combination?", details.errorDetails);
        [self updateRecognitionErrorText:([NSString stringWithFormat:@"Canceled: %@",
details.errorDetails ])];
    } else if (SPXResultReason_RecognizedSpeech == speechResult.reason) {
        NSLog(@"Speech recognition result received: %@", speechResult.text);
        [self updateRecognitionResultText:(speechResult.text)];
    } else {
        NSLog(@"There was an error.");
        [self updateRecognitionErrorText:(@"Speech Recognition Error")];
    }
}

- (void)updateRecognitionResultText:(NSString *) resultText {
    dispatch_async(dispatch_get_main_queue(), ^{
        self.recognitionResultLabel.textColor = UIColor.blackColor;
        self.recognitionResultLabel.text = resultText;
    });
}

- (void)updateRecognitionErrorText:(NSString *) errorText {
    dispatch_async(dispatch_get_main_queue(), ^{
        self.recognitionResultLabel.textColor = UIColor.redColor;
        self.recognitionResultLabel.text = errorText;
    });
}

- (void)updateRecognitionStatusText:(NSString *) statusText {
    dispatch_async(dispatch_get_main_queue(), ^{
        self.recognitionResultLabel.textColor = UIColor.grayColor;
        self.recognitionResultLabel.text = statusText;
```

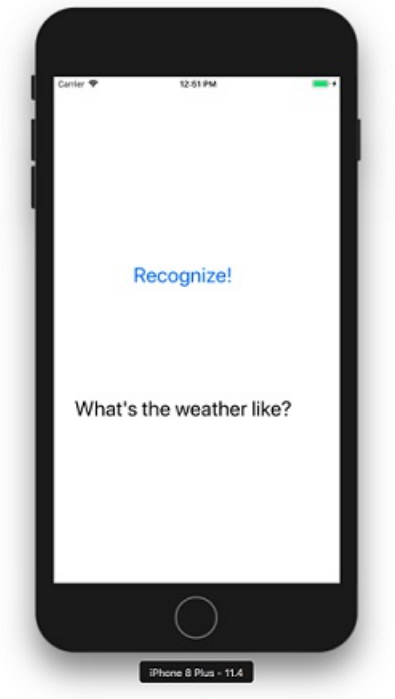
```
    });
}

@end
```

3. Replace the string `YourSubscriptionKey` with your subscription key.
4. Replace the string `YourServiceRegion` with the region associated with your subscription (for example, `westus` for the free trial subscription).
5. Add the request for microphone access. Right click the `Info.plist` entry of the project tree and select **Open As...** > **Source Code**. Add the following lines into the `<dict>` section and then save the file.

```
xml <key>NSMicrophoneUsageDescription</key> <string>Need microphone access for speech recognition from microphone.</string>
```

Building and Running the Sample

1. Make the debug output visible (**View** > **Debug Area** > **Activate Console**).
2. Choose either the iOS simulator or a iOS device connected to your development machine as the destination for the app from the list in the **Product** -> **Destination** menu.
3. Build and run the example code in the iOS simulator by selecting **Product** -> **Run** from the menu or clicking the **Play** button. Currently the Speech SDK only supports 64bit iOS platforms.
4. After you click the "Recognize (File)" button in the app, you should see the contents of the audio file "What's the weather like?" on the lower part of the screen.



5. After you click the "Recognize (Microphone)" button in the app and say a few words, you should see the text you have spoken on the lower part of the screen.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for this sample in the `quickstart/objectivec-ios` folder.

Next steps

Get our samples

Get started with the Speech Devices SDK

10/19/2018 • 6 minutes to read • Edit Online

This article describes how to configure your development PC and Speech device development kit for developing speech-enabled devices by using the Speech Devices SDK. Then you build and deploy a sample application to the device.

The source code for the sample application is included with the Speech Devices SDK. It's also available on GitHub.

Prerequisites

Before you begin developing with the Speech Devices SDK, gather the information and software you need:

- Get a development kit from ROOBO. Kits are available with linear or circular microphone array configurations. Choose the correct configuration for your needs.

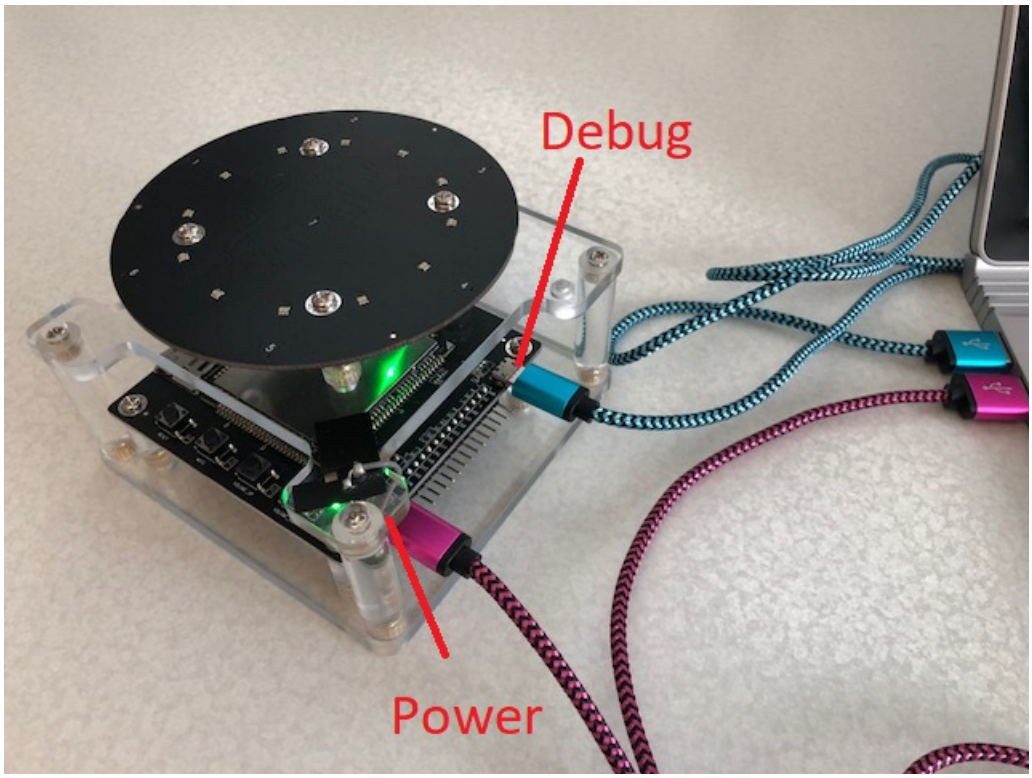
DEVELOPMENT KIT CONFIGURATION	SPEAKER LOCATION
Circular	Any direction from the device
Linear	In front of the device

- Get the latest version of the Speech Devices SDK, which includes an Android sample app, from the Speech Devices SDK download site. Extract the .zip file to a local folder, like C:\SDSDK.
- Install Android Studio and Vysor on your PC.
- Get a Speech service subscription key. You can get a 30-day free trial or get a key from your Azure dashboard.
- If you want to use the Speech service's intent recognition, subscribe to the Language Understanding service (LUIS) and get a subscription key.

  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.

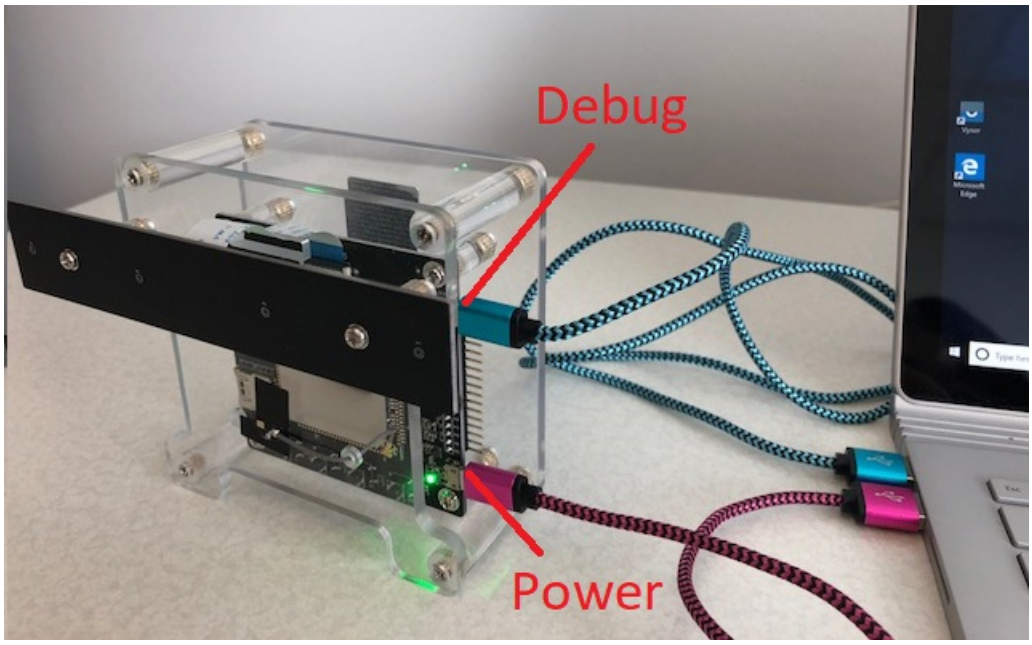
Set up the development kit

1. Connect the development kit to a PC or power adapter by using a mini USB cable. When the kit is connected, a green power indicator lights up under the top board.
2. Connect the development kit to a computer by using a second mini USB cable.



3. 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)



4. Install the certificates and the wake word (keyword) table file, 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
```

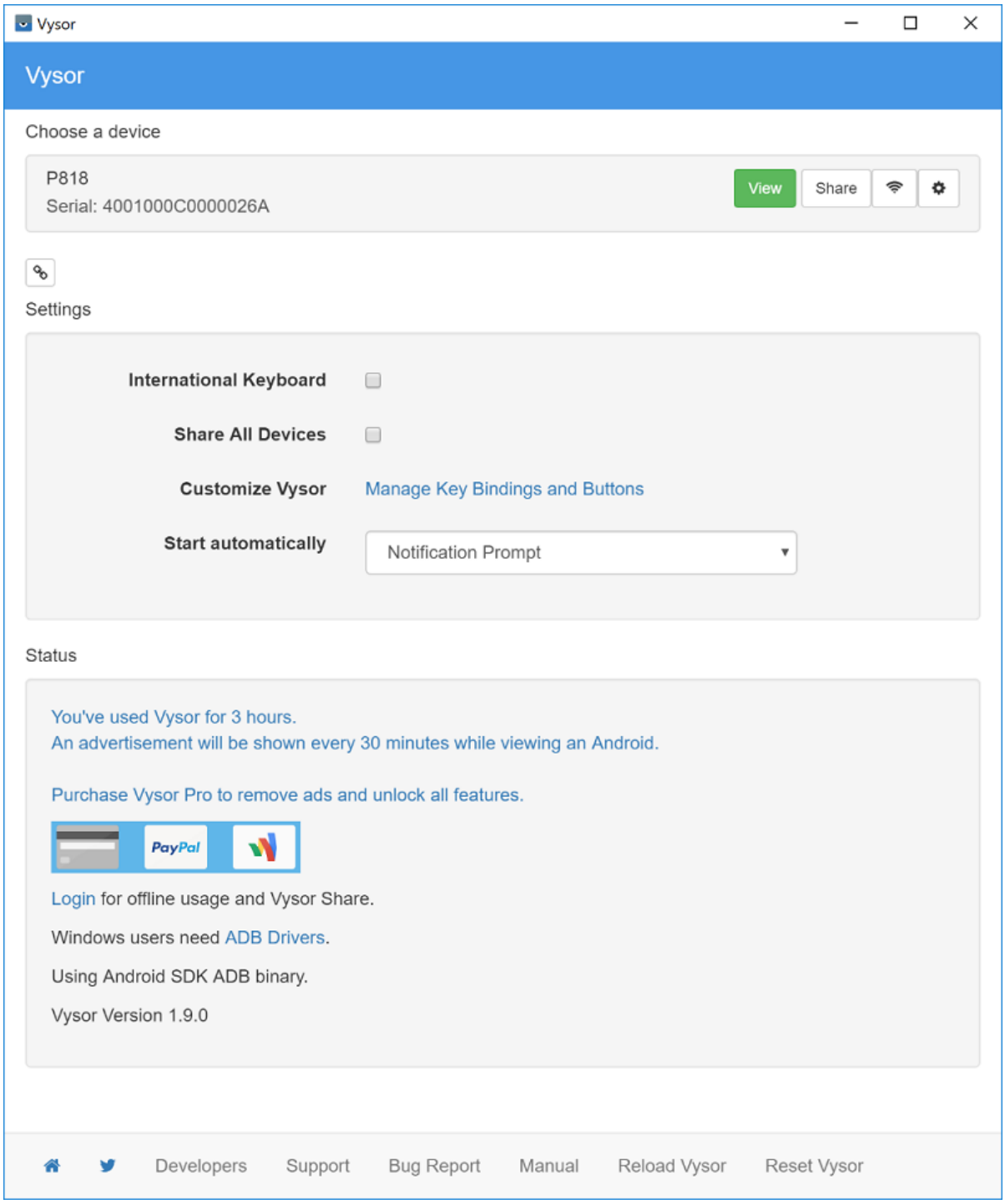
NOTE

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`.

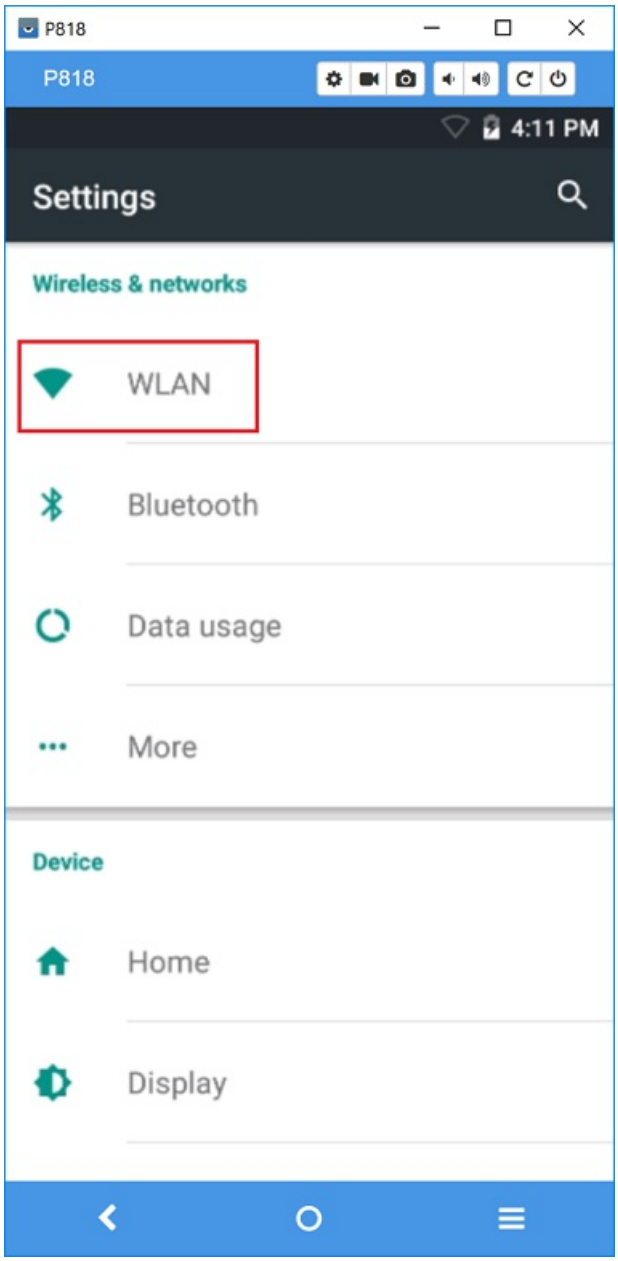
TIP

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.

5. Start Vysor on your computer.



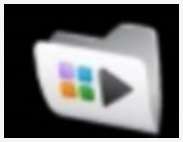
6. Your device should be listed under **Choose a device**. Select the **View** button next to the device.
7. Connect to your wireless network by selecting the folder icon, and then select **Settings** > **WLAN**.



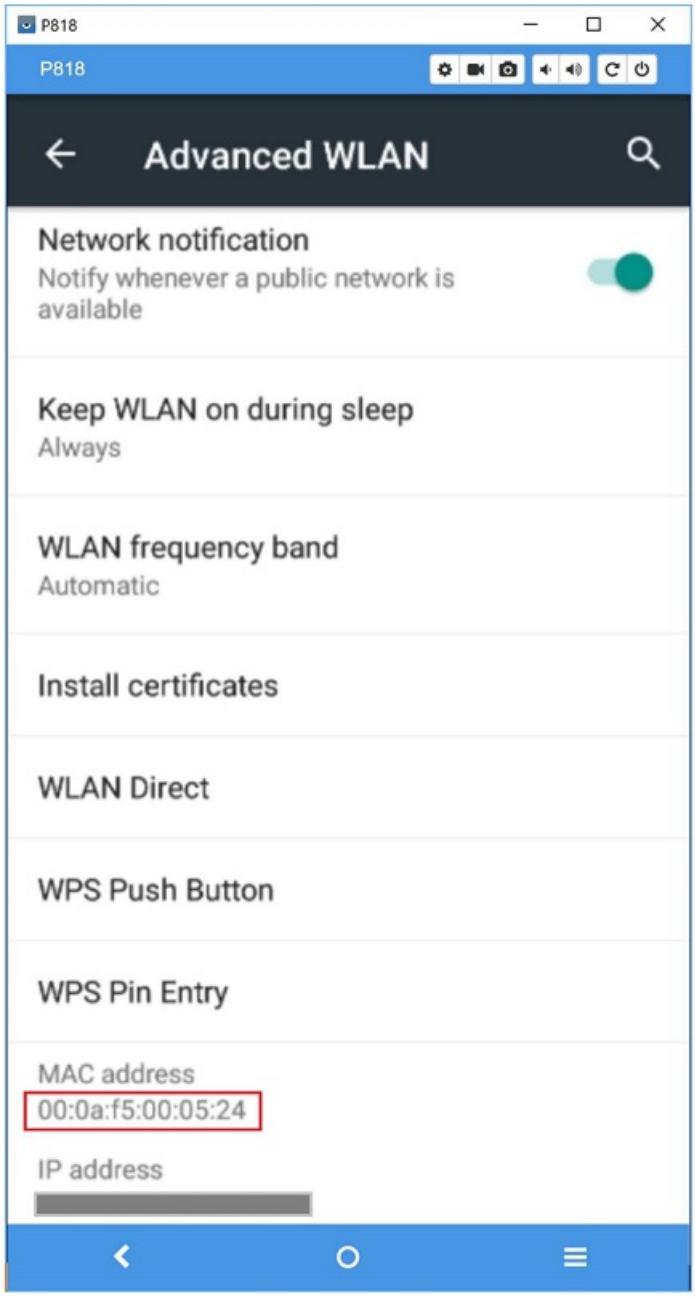
NOTE

If your company has policies about connecting devices to its Wi-Fi system, you need to obtain the MAC address and contact your IT department about how to connect it to your company's Wi-Fi.

To find the MAC address of the dev kit, select the file folder icon on the desktop of the dev kit.

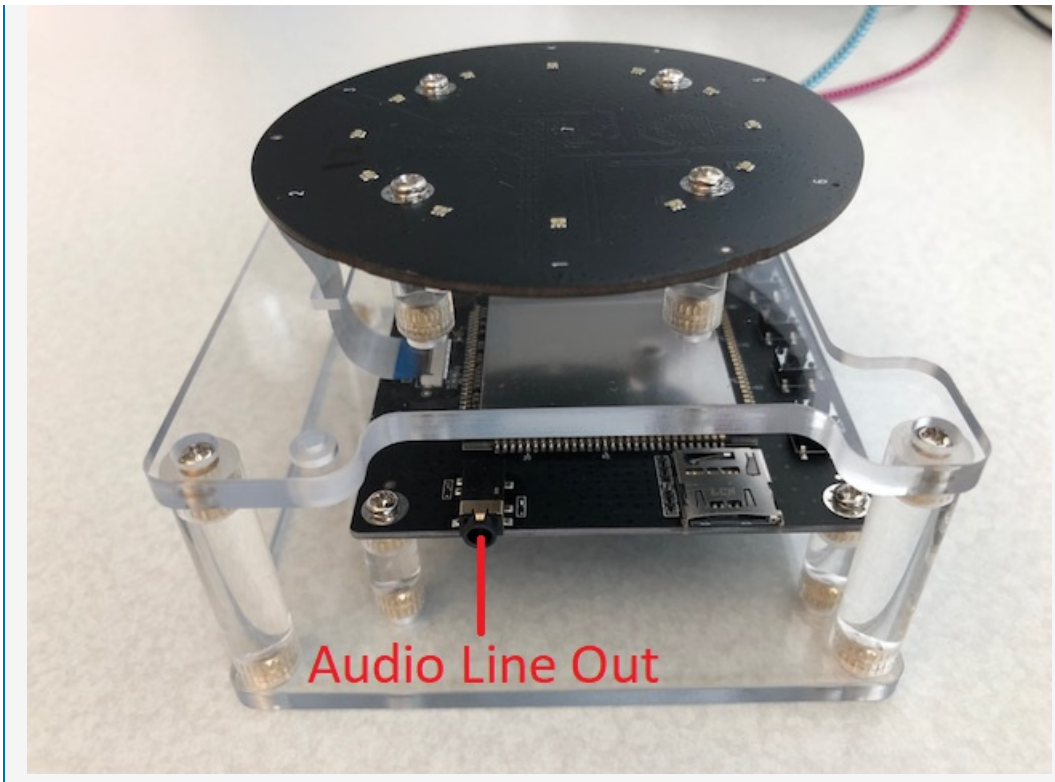


Select **Settings**. Search for "mac address", and then select **Mac address** > **Advanced WLAN**. Write down the MAC address that appears near the bottom of the dialog box.



Some companies might have a time limit on how long a device can be connected to their Wi-Fi system. You might need to extend the dev kit's registration with your Wi-Fi system after a specific number of days.

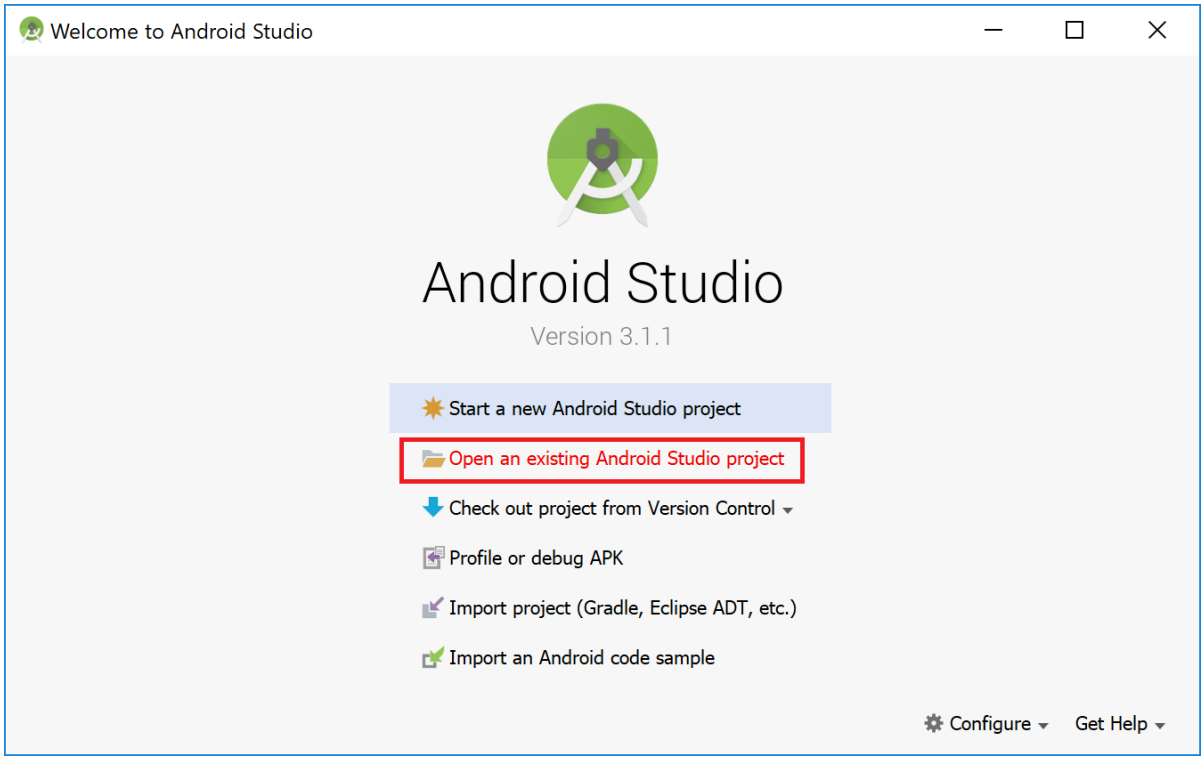
If you want to attach a speaker to the dev kit, you can connect it to the audio line out. You should choose a good-quality, 3.5-mm speaker.



Run a sample application

To run the ROOBO tests and validate your development kit setup, build and install the sample application:

1. Start Android Studio.
2. Select **Open an existing Android Studio project**.



3. Go to C:\SDSDK\Android-Sample-Release\example. Select **OK** to open the example project.
4. Add your Speech subscription key to the source code. If you want to try intent recognition, also add your Language Understanding service subscription key and application ID.

Your keys and application information go in the following lines in the source file MainActivity.java:

```
// Subscription
private static final String SpeechSubscriptionKey = "[your speech key]";
private static final String SpeechRegion = "westus";
private static final String LuisSubscriptionKey = "[your LUIS key]";
private static final String LuisRegion = "westus2.api.cognitive.microsoft.com";
private static final String LuisAppId = "[your LUIS app ID]"
```

5. The default wake word (keyword) is "Computer". You can also try one of the other provided wake words, like "Machine" or "Assistant". The resource files for these alternate wake words 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 wake word "Computer".

   You can also create a custom wake word.

   To install the wake word you want to use:

   - Create a keyword folder in the data folder on the device by running the following commands in a Command Prompt window:

     ```
     adb shell
     cd /data
     mkdir keyword
     exit
     ```

   - Copy the files kws.table, kws_k.fst, and words_kw.txt to the device's \data\keyword folder. Run the following commands in a Command Prompt window. If you created a custom wake word, the kws.table file generated from the web is in the same directory as the kws.table, kws_k.fst, and words_kw.txt files. For a custom wake word, use the `adb push C:\SDSDK\Android-Sample-Release\keyword\[wake_word_name]\kws.table /data/keyword` command to push the kws.table file to the dev kit:

     ```
     adb push C:\SDSDK\Android-Sample-Release\keyword\kws.table /data/keyword
     adb push C:\SDSDK\Android-Sample-Release\keyword\Computer\kws_k.fst /data/keyword
     adb push C:\SDSDK\Android-Sample-Release\keyword\Computer\words_kw.txt /data/keyword
     ```

   - Reference these files in the sample application. Find the following lines in MainActivity.java. Make sure that the keyword specified is the one you're using, and that the path points to the `kws.table` file that you pushed to the device.

     ```
     private static final String Keyword = "Computer";
     private static final String KeywordModel = "/data/keyword/kws.table";
     ```

     NOTE

     In your own code, you can use the kws.table file to create a keyword model instance and start recognition:

     ```
     KeywordRecognitionModel km = KeywordRecognitionModel.fromFile(KeywordModel);
     final Task<?> task = reco.startKeywordRecognitionAsync(km);
     ```

6. Update the following lines, which contain the microphone array geometry settings:

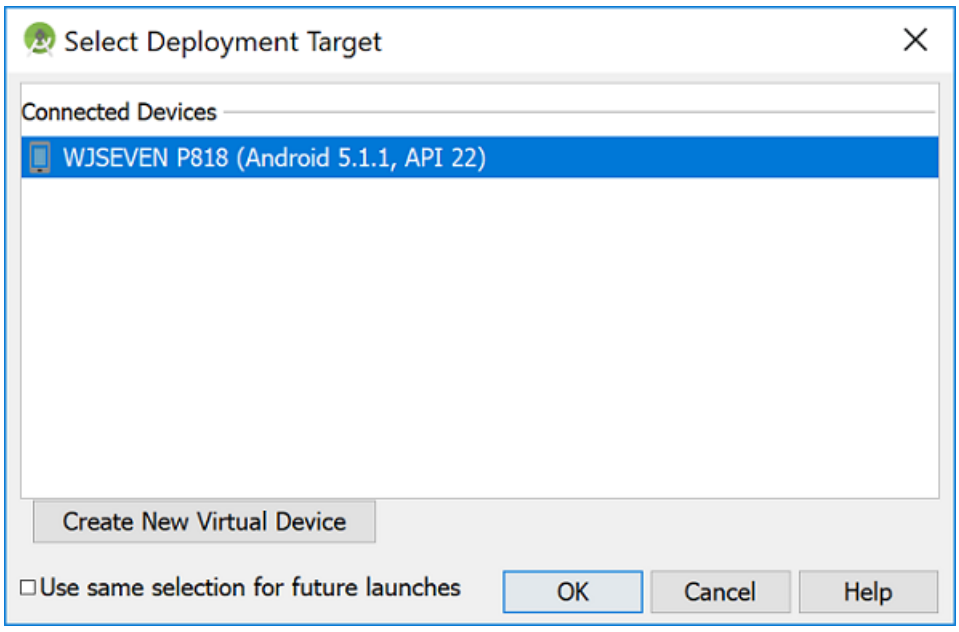
```
private static final String DeviceGeometry = "Circular6+1";
private static final String SelectedGeometry = "Circular6+1";
```

The following table describes the available values:

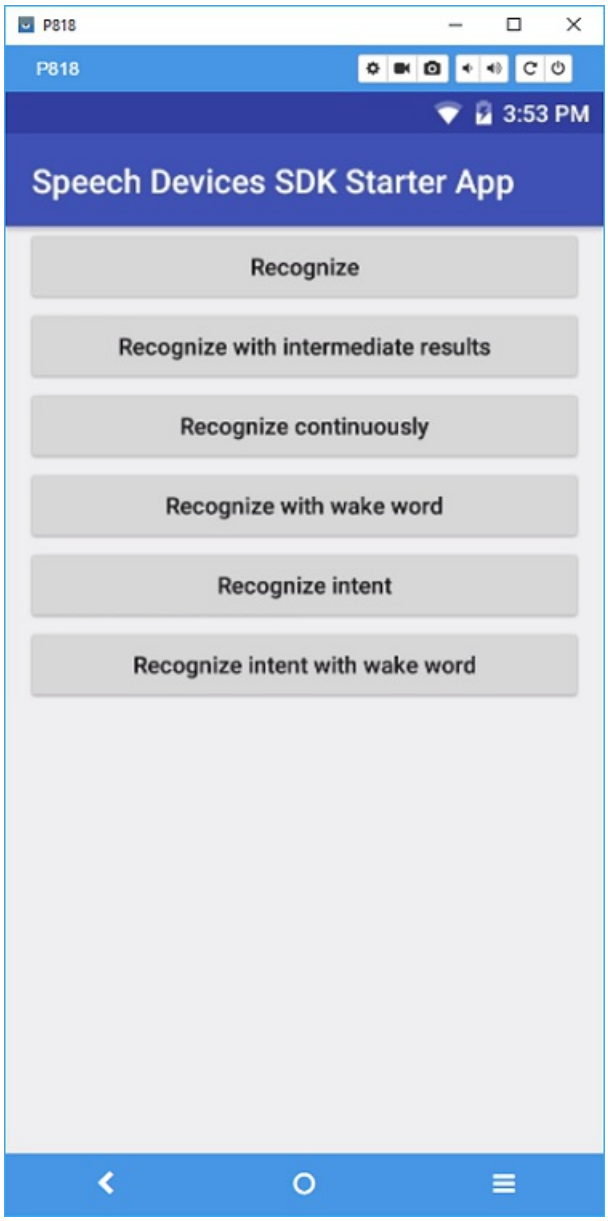
VARIABLE	MEANING	AVAILABLE VALUES
`DeviceGeometry`	Physical mic configuration	For a circular dev kit: `Circular6+1`
		For a linear dev kit: `Linear4`
`SelectedGeometry`	Software mic configuration	For a circular dev kit that uses all mics: `Circular6+1`
		For a circular dev kit that uses four mics: `Circular3+1`
		For a linear dev kit that uses all mics: `Linear4`
		For a linear dev kit that uses two mics: `Linear2`

7. To build the application, on the **Run** menu, select **Run 'app'**. The **Select Deployment Target** dialog box appears.

8. Select your device, and then select **OK** to deploy the application to the device.



9. The Speech Devices SDK example application starts and displays the following options:



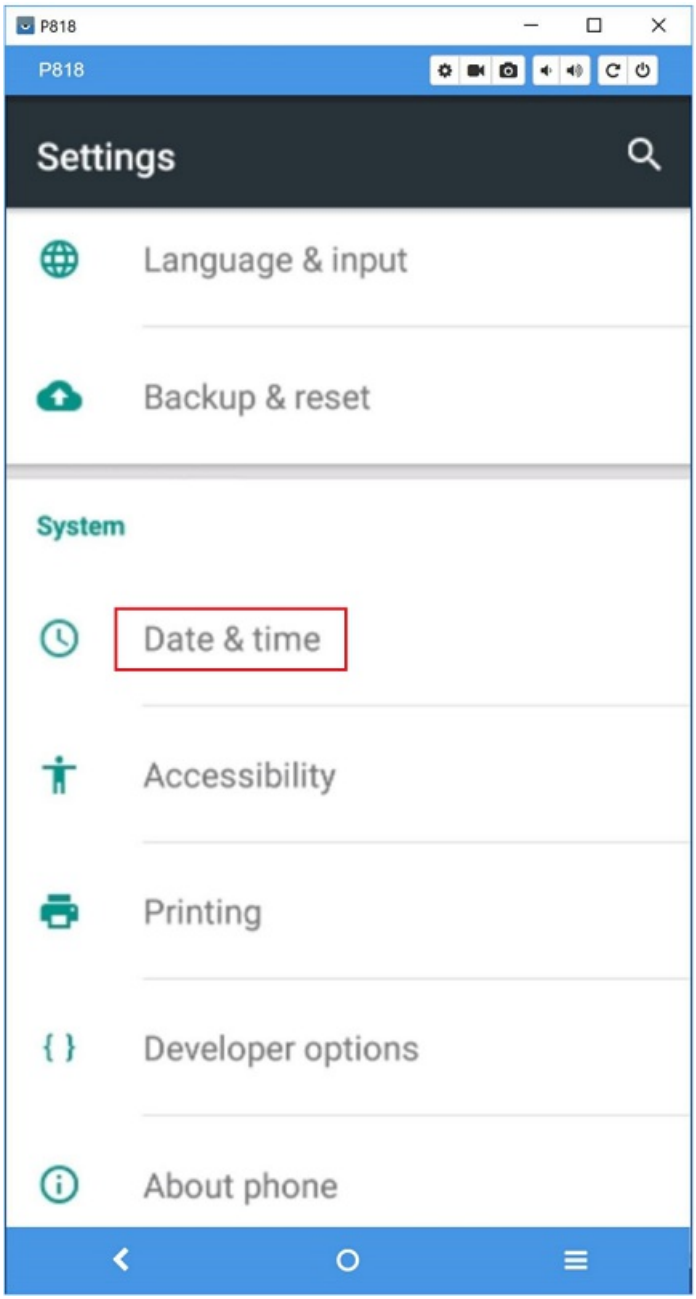
10. Experiment!

Troubleshooting

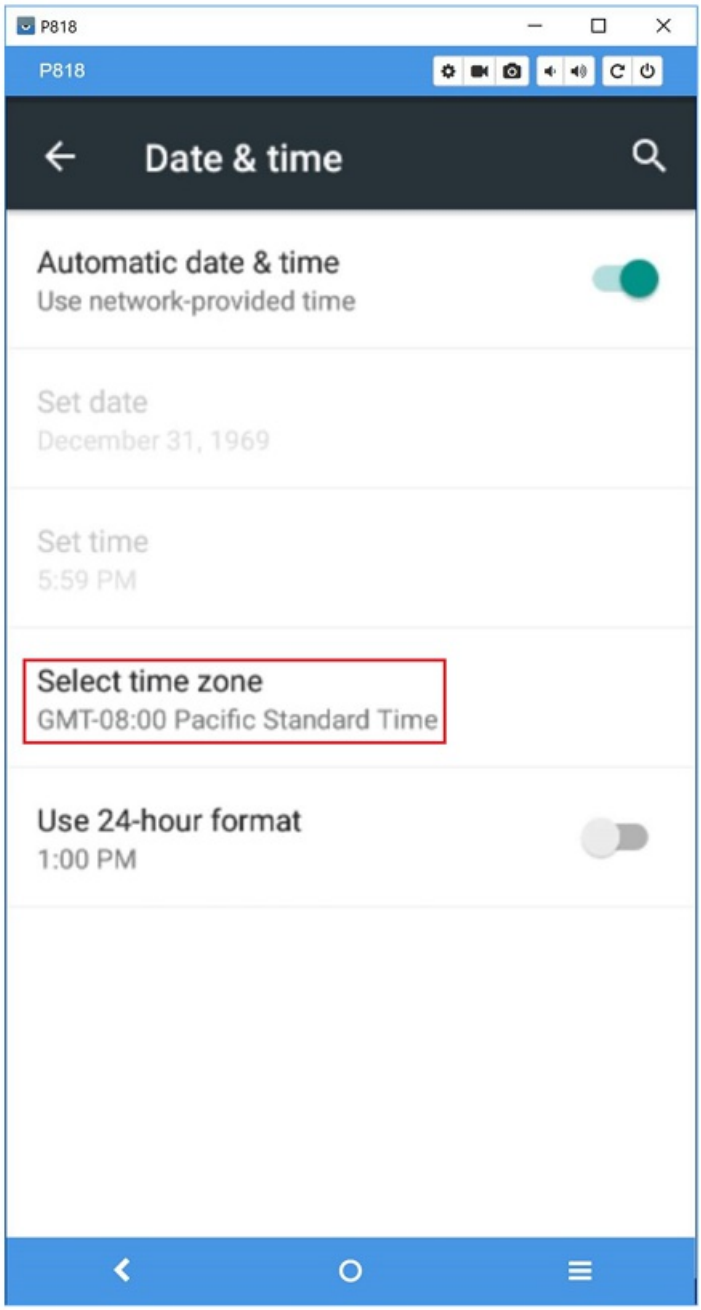
Certificate failures

If you get certificate failures when you use 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.

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.

Recognize speech by using the Speech SDK for C#

10/19/2018 • 4 minutes to read • Edit Online

The Cognitive Services Speech SDK provides the simplest way to use **Speech to Text** in your application with full functionality.

1. Create a speech configuration and provide a Speech service subscription key (or an authorization token) and a region as parameters. Change the configuration as needed. For example, provide a custom endpoint to specify a non-standard service endpoint, or select the spoken input language or output format.
2. Create a speech recognizer from the speech configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results. Otherwise, your application receives only a final transcription result.
4. Start recognition. For single-shot recognition, such as command or query recognition, use the `RecognizeOnceAsync()` method. This method returns the first recognized utterance. For long-running recognition like transcription, use the `StartContinuousRecognitionAsync()` method. Tie up the events for asynchronous recognition results.

See the following code snippets for speech recognition scenarios that use the Speech SDK.

> **NOTE**
> Information on how to setup a development project for your platform and preferred development environment can be found in the Quickstarts articles in this documentation. In any case, you will need a subscription key, see Try the speech service for free.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

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

Speech recognition from a microphone

The following code snippet shows how to recognize speech input from a microphone in the default language (en-US).

```
// Creates an instance of a speech config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
// The default language is "en-us".
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

// Creates a speech recognizer using microphone as audio input.
using (var recognizer = new SpeechRecognizer(config))
{
    // Starts recognizing.
    Console.WriteLine("Say something...");

    // Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
    // so it is suitable only for single shot recognition like command or query. For long-running
    // recognition, use StartContinuousRecognitionAsync() instead.
    var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);

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

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

Speech recognition with a customized model

The Custom Speech service (CRIS) allows the customization of the Microsoft speech-to-text engine for your application. The following code snippet shows how to recognize speech from a microphone by using your CRIS model. Fill in your subscription key and your own endpoint identification before you run the code.

```
// Creates an instance of a speech config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
// Replace with the CRIS endpoint id of your customized model.
config.EndpointId = "YourEndpointId";

// Creates a speech recognizer using microphone as audio input.
using (var recognizer = new SpeechRecognizer(config))
{
    Console.WriteLine("Say something...");

    // Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
    // so it is suitable only for single shot recognition like command or query. For long-running
    // recognition, use StartContinuousRecognitionAsync() instead.
    var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);

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

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

Continuous speech recognition from a file

The following code snippet continuously recognizes speech input from an audio file in the default language (en-US). The supported format is single-channel (mono) WAV / PCM with a sampling rate of 16 kHz.

> **NOTE**
>
> Need a sample audio? Right-click this WAV file and choose **Save target as**.

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

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(@"whatstheweatherlike.wav"))
{
    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)
            {
                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);
    }
}
```

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/csharp/sharedcontent/console folder.

Next steps

- How to recognize intent from speech
- How to translate speech

Recognize speech by using the Speech SDK for C++

10/19/2018 • 5 minutes to read • Edit Online

The Cognitive Services Speech SDK provides the simplest way to use **Speech to Text** in your application with full functionality.

1. Create a speech configuration and provide a Speech service subscription key (or an authorization token) and a region as parameters. Change the configuration as needed. For example, provide a custom endpoint to specify a non-standard service endpoint, or select the spoken input language or output format.
2. Create a speech recognizer from the speech configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results. Otherwise, your application receives only a final transcription result.
4. Start recognition. For single-shot recognition, such as command or query recognition, use the `RecognizeOnceAsync()` method. This method returns the first recognized utterance. For long-running recognition like transcription, use the `StartContinuousRecognitionAsync()` method. Tie up the events for asynchronous recognition results.

See the following code snippets for speech recognition scenarios that use the Speech SDK.

> **NOTE**
> Information on how to setup a development project for your platform and preferred development environment can be found in the Quickstarts articles in this documentation. In any case, you will need a subscription key, see Try the speech service for free.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
#include <speechapi_cxx.h>
#include <fstream>

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

Speech recognition from a microphone

The following code snippet shows how to recognize speech input from a microphone in the default language (en-US).

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

// Creates a speech recognizer using microphone as audio input. The default language is "en-us".
auto recognizer = SpeechRecognizer::FromConfig(config);
cout << "Say something...\n";

// Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
// so it is suitable only for single shot recognition like command or query. For long-running
// recognition, use StartContinuousRecognitionAsync() instead.
auto result = recognizer->RecognizeOnceAsync().get();

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

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

Speech recognition with a customized model

The Custom Speech service (CRIS) allows the customization of the Microsoft speech-to-text engine for your application. The following code snippet shows how to recognize speech from a microphone by using your CRIS model. Fill in your subscription key and your own endpoint identification before you run the code.

```
// Creates an instance of a speech config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
auto config = SpeechConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");
// Set the endpoint ID of your customized model
// Replace with your own CRIS endpoint ID.
config->SetEndpointId("YourEndpointId");

// Creates a speech recognizer using microphone as audio input.
auto recognizer = SpeechRecognizer::FromConfig(config);

cout << "Say something...\n";

// Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
// so it is suitable only for single shot recognition like command or query. For long-running
// recognition, use StartContinuousRecognitionAsync() instead.
auto result = recognizer->RecognizeOnceAsync().get();

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

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

Continuous speech recognition from a file

The following code snippet continuously recognizes speech input from an audio file in the default language (en-US). The supported format is single-channel (mono) WAV / PCM with a sampling rate of 16 kHz.

> **NOTE**
>
> Need a sample audio? Right-click this WAV file and choose **Save target as**.

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

// Creates a speech recognizer using file as audio input.
// Replace with your own audio file name.
auto audioInput = AudioConfig::FromWavFileInput("whatstheweatherlike.wav");
auto recognizer = SpeechRecognizer::FromConfig(config, audioInput);

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

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

recognizer->Recognized.Connect([] (const SpeechRecognitionEventArgs& e)
{
    if (e.Result->Reason == ResultReason::RecognizedSpeech)
    {
        cout << "RECOGNIZED: Text=" << e.Result->Text << std::endl
             << "  Offset=" << e.Result->Offset() << std::endl
             << "  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: ErrorDetails=" << e.ErrorDetails << std::endl;
        cout << "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.
});

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

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

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

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/cpp/windows/console folder.

Next steps

- How to recognize intents from speech
- How to translate speech

Recognize speech by using the Speech SDK for Java

10/19/2018 • 4 minutes to read • Edit Online

The Cognitive Services Speech SDK provides the simplest way to use **Speech to Text** in your application with full functionality.

1. Create a speech configuration and provide a Speech service subscription key (or an authorization token) and a region as parameters. Change the configuration as needed. For example, provide a custom endpoint to specify a non-standard service endpoint, or select the spoken input language or output format.
2. Create a speech recognizer from the speech configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results. Otherwise, your application receives only a final transcription result.
4. Start recognition. For single-shot recognition, such as command or query recognition, use the `RecognizeOnceAsync()` method. This method returns the first recognized utterance. For long-running recognition like transcription, use the `StartContinuousRecognitionAsync()` method. Tie up the events for asynchronous recognition results.

See the following code snippets for speech recognition scenarios that use the Speech SDK.

> **NOTE**
> Information on how to setup a development project for your platform and preferred development environment can be found in the Quickstarts articles in this documentation. In any case, you will need a subscription key, see Try the speech service for free.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.*;
```

Speech recognition from a microphone

The following code snippet shows how to recognize speech input from a microphone in the default language (en-US).

```
// Creates an instance of a speech config with specified
// subscription key and service region. Replace with your own subscription key
// and service region (e.g., "westus").
// The default language is "en-us".
SpeechConfig config = SpeechConfig.fromSubscription("YourSubscriptionKey", "YourServiceRegion");

// Creates a speech recognizer using microphone as audio input.
SpeechRecognizer recognizer = new SpeechRecognizer(config);
{
    // Starts recognizing.
    System.out.println("Say something...");

    // Starts recognition. It returns when the first utterance has been recognized.
    SpeechRecognitionResult result = recognizer.recognizeOnceAsync().get();

    // Checks result.
    if (result.getReason() == ResultReason.RecognizedSpeech) {
        System.out.println("RECOGNIZED: Text=" + result.getText());
    }
    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: ErrorDetails=" + cancellation.getErrorDetails());
            System.out.println("CANCELED: Did you update the subscription info?");
        }
    }
}
recognizer.close();
```

Speech recognition with a customized model

The Custom Speech service (CRIS) allows the customization of the Microsoft speech-to-text engine for your application. The following code snippet shows how to recognize speech from a microphone by using your CRIS model. Fill in your subscription key and your own endpoint identification before you run the code.

```
// Creates an instance of a speech config with specified
// subscription key and service region. Replace with your own subscription key
// and service region (e.g., "westus").
SpeechConfig config = SpeechConfig.fromSubscription("YourSubscriptionKey", "YourServiceRegion");
// Replace with the CRIS endpoint id of your customized model.
config.setEndpointId("YourEndpointId");

// Creates a speech recognizer using microphone as audio input.
SpeechRecognizer recognizer = new SpeechRecognizer(config);
{
    // Starts recognizing.
    System.out.println("Say something...");

    // Starts recognition. It returns when the first utterance has been recognized.
    SpeechRecognitionResult result = recognizer.recognizeOnceAsync().get();

    // Checks result.
    if (result.getReason() == ResultReason.RecognizedSpeech) {
        System.out.println("RECOGNIZED: Text=" + result.getText());
    }
    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: ErrorDetails=" + cancellation.getErrorDetails());
            System.out.println("CANCELED: Did you update the subscription info?");
        }
    }
}

recognizer.close();
```

Continuous speech recognition from a file

The following code snippet continuously recognizes speech input from an audio file in the default language (en-US). The supported format is single-channel (mono) WAV / PCM with a sampling rate of 16 kHz.

> **NOTE**
>
> Need a sample audio? Right-click this WAV file and choose **Save target as**.

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

// Creates a speech recognizer using file as audio input.
// Replace with your own audio file name.
AudioConfig audioInput = AudioConfig.fromWavFileInput("YourAudioFile.wav");
SpeechRecognizer recognizer = new SpeechRecognizer(config, audioInput);
{
    // Subscribes to events.
    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: ErrorDetails=" + e.getErrorDetails());
            System.out.println("CANCELED: Did you update the subscription info?");
        }
    });

    recognizer.sessionStarted.addEventListener((s, e) -> {
        System.out.println("\n    Session started event.");
    });

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

    // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
    System.out.println("Say something...");
    recognizer.startContinuousRecognitionAsync().get();

    System.out.println("Press any key to stop");
    new Scanner(System.in).nextLine();

    recognizer.stopContinuousRecognitionAsync().get();
}
```

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/java/jre/console folder.

Next steps

- How to recognize intents from speech
- How to translate speech

Recognize speech by using the REST API

10/19/2018 • 2 minutes to read • Edit Online

The REST API can be used to recognize short utterances by using an HTTP POST request.

The REST API is the simplest way to recognize speech if you aren't using a language that's supported by the SDK. You make an HTTP POST request to the service endpoint and pass the entire utterance in the body of the request. You receive a response that has the recognized text.

> **NOTE**
> Utterances are limited to 15 seconds or less when you use the REST API. Check out the Speech SDK for recognition of longer utterances.

For more information on the **Speech to Text** REST API, see the REST APIs article. To see the API in action, download the REST API samples from GitHub.

Next steps

- See the REST API overview.

Tutorial: Recognize intents from speech using the Speech SDK for C#

10/19/2018 • 11 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 have 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 tutorial, 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 config 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 before you begin this tutorial.

- A LUIS account. You can get one for free through the LUIS portal.
- Visual Studio 2017 (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 two kinds of keys:

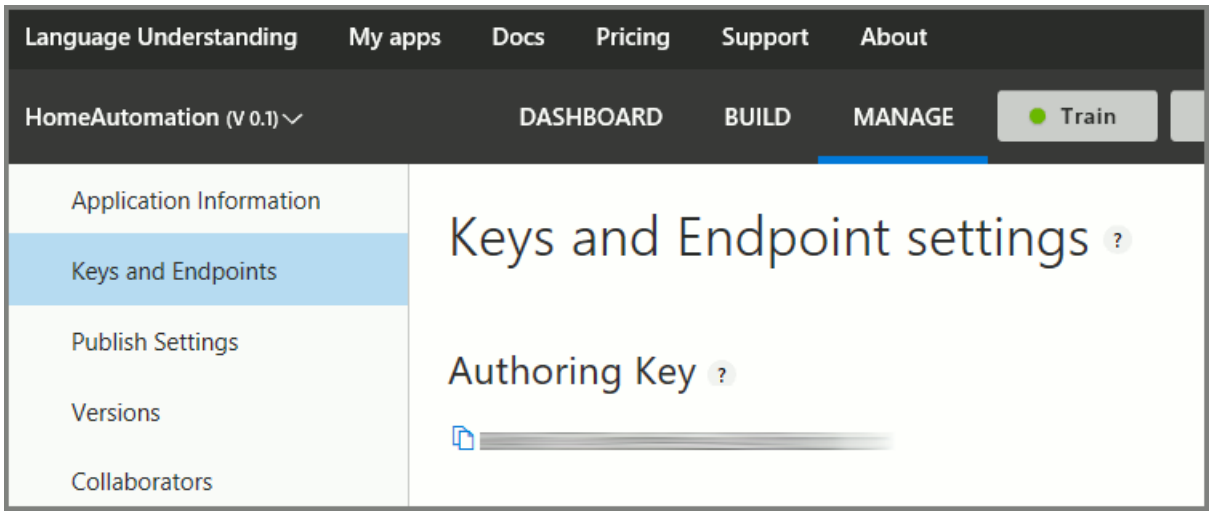
KEY TYPE	PURPOSE
authoring	lets you create and modify LUIS apps programmatically
endpoint	authorizes access to a particular LUIS app

The endpoint key is the LUIS key needed for this tutorial. This tutorial uses the example Home Automation LUIS app, which you can create by following Use prebuilt Home automation app. If you have created a LUIS app of your own, you can use it instead.

When you create a LUIS app, a starter key is automatically generated so you can test the app using text queries.

This key does not enable the Speech service integration and won't work with this tutorial. You must create a LUIS resource in the Azure dashboard and assign it to the LUIS app. You can use the free subscription tier for this tutorial.

After creating 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, click **Keys and Endpoints** in the sidebar.

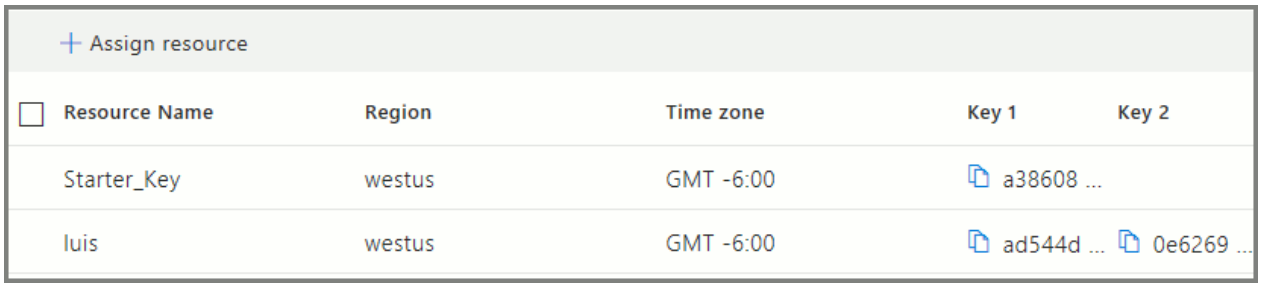


On the Keys and Endpoint settings page:

1. Scroll down to the Resources and Keys section and click **Assign resource**.
2. In the **Assign a key to your app** dialog, choose the following:

   - Choose Microsoft as the Tenant.
   - 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. Click the icon next to a key to copy it to the clipboard. (You may use either key.)

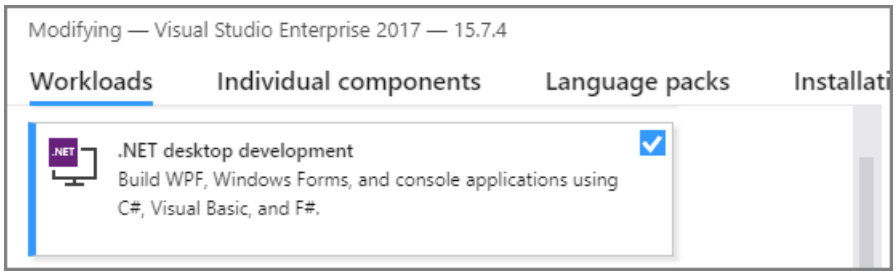


Create a speech project in Visual Studio

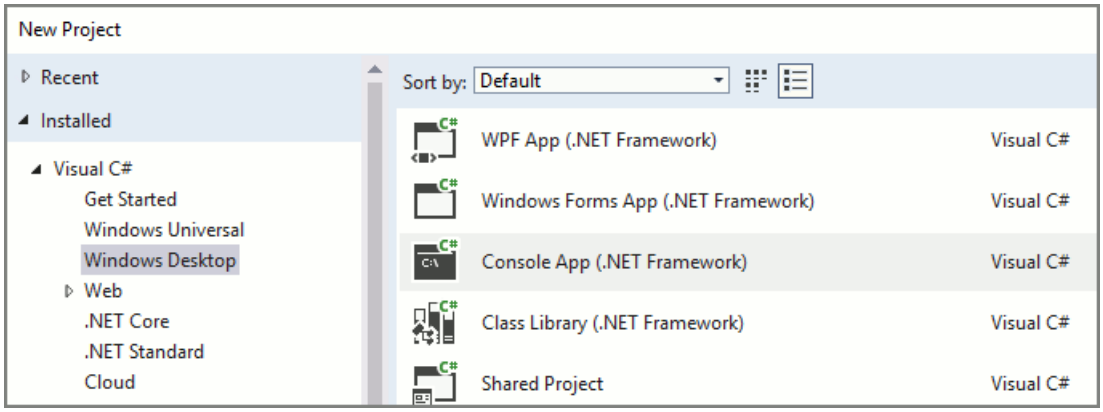
1. Start Visual Studio 2017.

2. Make sure the **.NET desktop environment** workload is available. Choose **Tools** > **Get Tools and Features** from the Visual Studio menu bar to open the Visual Studio installer. If this workload is already enabled, close the dialog.

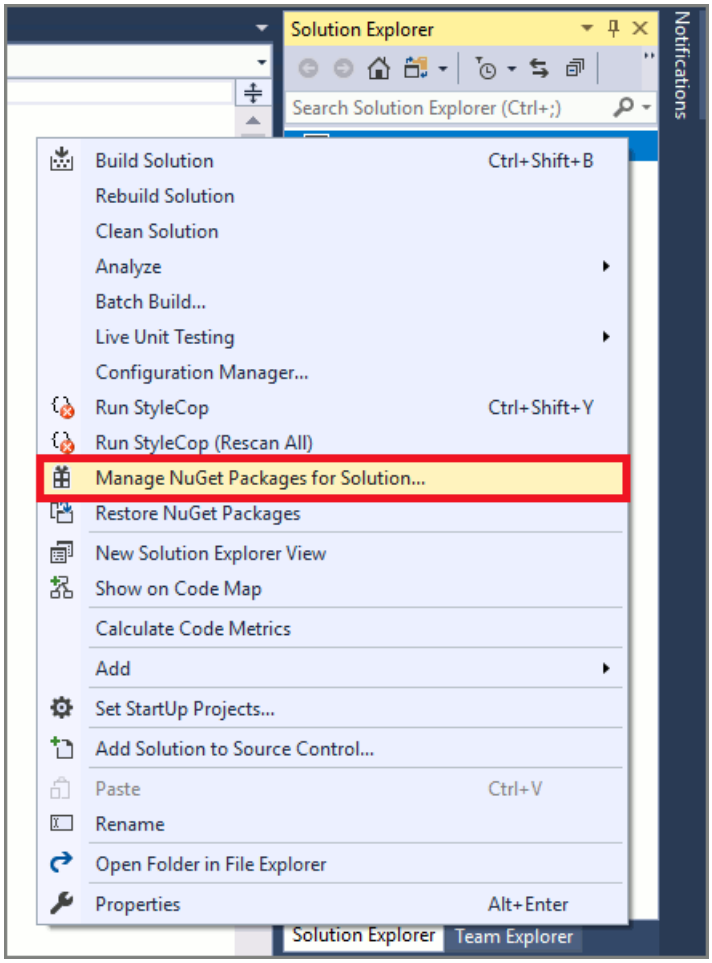
   Otherwise, mark the checkbox next to **.NET desktop development,** then click the **Modify** button at the lower right corner of the dialog. Installation of the new feature will take a moment.



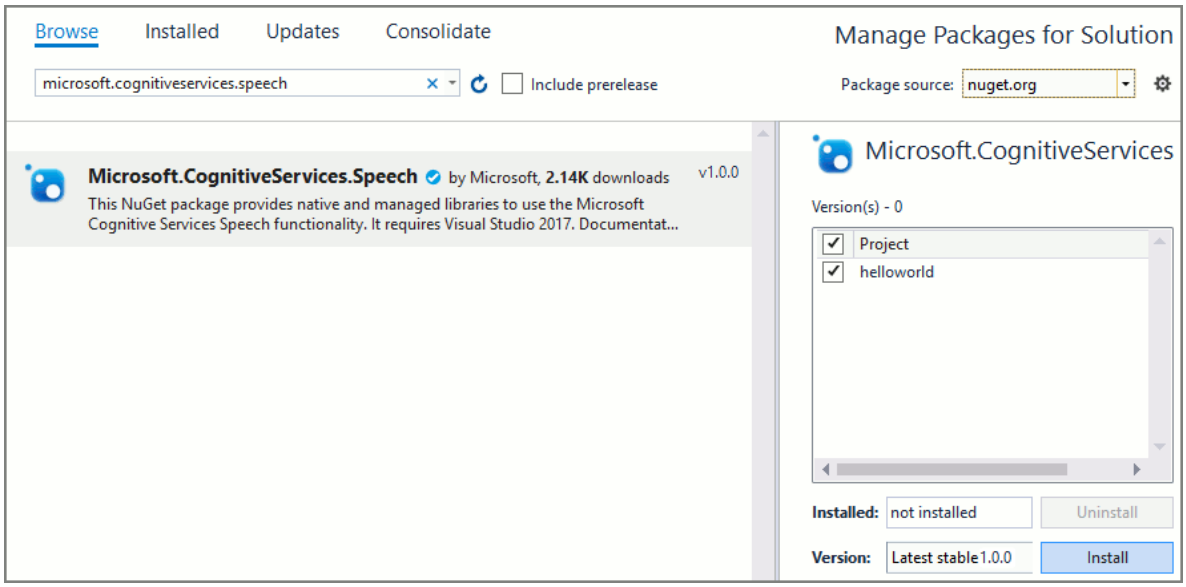
3. Create a new Visual C# Console App. In the **New Project** dialog box, from the left pane, expand **Installed** > **Visual C#** > **Windows Desktop** and then choose **Console App (.NET Framework)**. For the project name, enter *helloworld*.



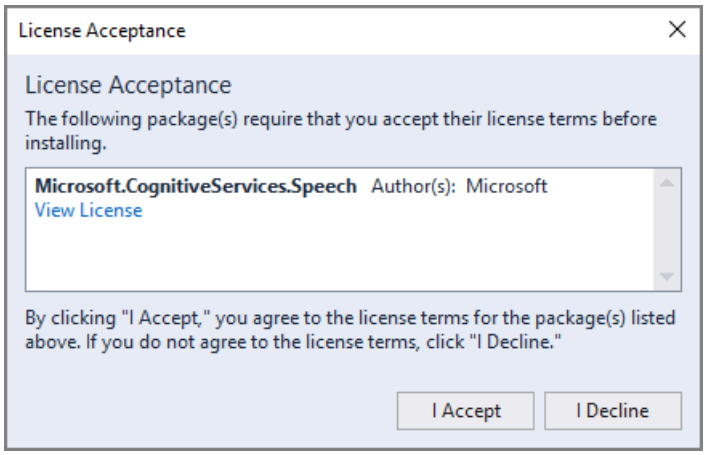
4. Install and reference the Speech SDK NuGet package. In the Solution Explorer, right-click the solution and select **Manage NuGet Packages for Solution**.



5. In the upper-right corner, in the **Package Source** field, select **nuget.org**. Search for the `Microsoft.CognitiveServices.Speech` package and install it into the **helloworld** project.

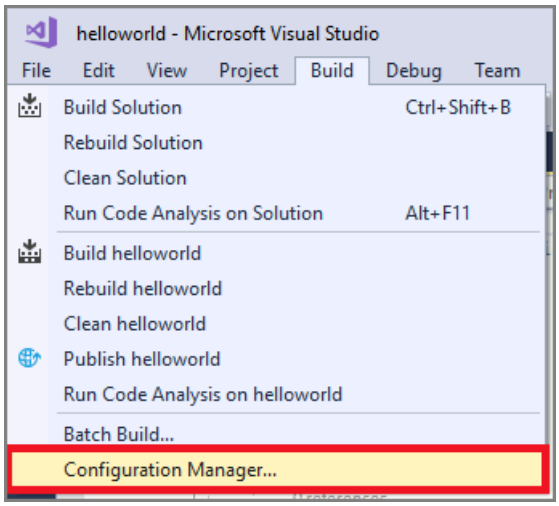


6. Accept the displayed license to begin installation of the NuGet package.

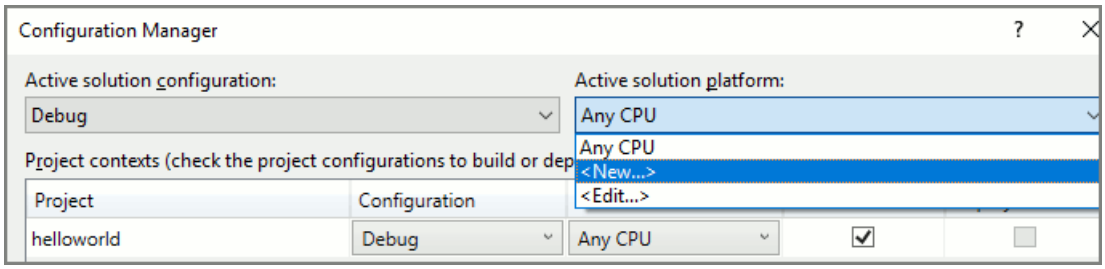


After the package is installed, a confirmation appears in the Package Manager console.

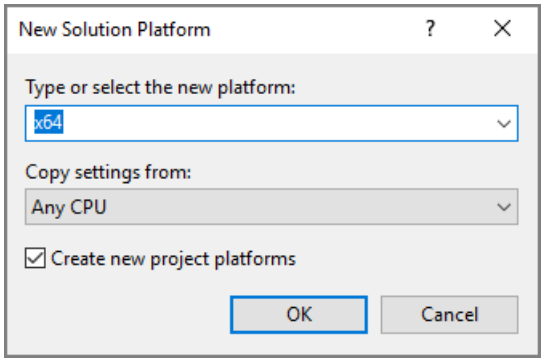
7. Create a platform configuration matching your PC architecture via the Configuration Manager. Select **Build** > **Configuration Manager**.



8. In the **Configuration Manager** dialog box, add a new platform. From the **Active solution platform** drop-down list, select **New**.



9. If you are running 64-bit Windows, create a new platform configuration named `x64`. If you are running 32-bit Windows, create a new platform configuration named `x86`.



Add the code

Open the file `Program.cs` in the Visual Studio project and 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;
```

Inside the provided `Main()` method, add the following code.

```
RecognizeIntentAsync().Wait();
Console.WriteLine("Please press Enter to continue.");
Console.ReadLine();
```

Create an empty asynchronous method `RecognizeIntentAsync()`, as shown here.

```
static async Task RecognizeIntentAsync()
{
}
```

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...");

    // Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
    // so it is suitable only for single shot recognition like command or query. For long-running
    // 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($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
            Console.WriteLine($"CANCELED: Did you update the subscription info?");
        }
    }
}
```

Replace the placeholders in this method with your LUIS subscription key, region, and app ID as follows.

PLACEHOLDER	REPLACE WITH

PLACEHOLDER	REPLACE WITH
`YourLanguageUnderstandingSubscriptionKey`	Your LUIS endpoint key. As previously noted, this must be a key obtained 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 of the LUIS portal.

With these changes made, you can build (Control-Shift-B) and run (F5) the tutorial application. When prompted, try saying "Turn off the lights" into your PC's microphone. The result is displayed in the console window.

The following sections include a discussion of the code.

Create an intent recognizer

The first step in recognizing intents in speech is to create a speech config from your LUIS endpoint key and region. Speech configs can be used to create recognizers for the various capabilities of the Speech SDK. The speech config 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 of a Speech Service subscription.

Next, create an intent recognizer using `new IntentRecognizer(config)`. Since the configuration already knows which subscription to use, there's no 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. It is not necessary to add all the app's intents if you do not need to recognize them all in your application.

Adding intents requires three arguments: the LUIS model (which has just 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. Must match the LUIS intent name exactly.
intentID	An ID assigned to a recognized intent by the Speech SDK. Can be whatever you like; does not 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 a device on, and another for turning a device

off. 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");
```

Start recognition

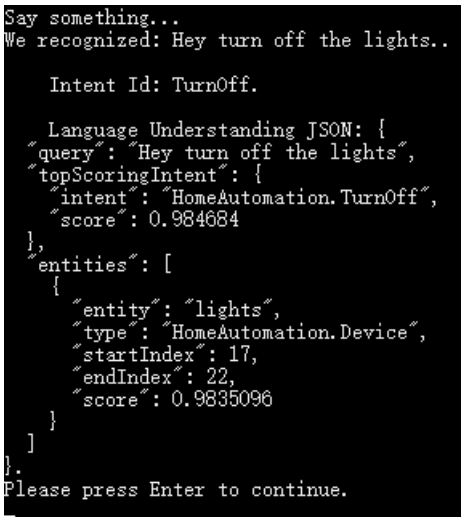
With the recognizer created and the intents added, recognition can begin. The Speech SDK supports both single-shot and continuous recognition.

RECOGNITION MODE	METHODS TO CALL	RESULT
Single-shot	`RecognizeOnceAsync()`	Returns the recognized intent, if any, after one utterance.
Continuous	`StartContinuousRecognitionAsync()` `StopContinuousRecognitionAsync()`	Recognizes multiple utterances. Emits events (e.g. `IntermediateResultReceived`) when results are available.

The tutorial application uses single-shot mode and so calls `RecognizeOnceAsync()` to begin recognition. The result is an `IntentRecognitionResult` object containing information about the intent recognized. The LUIS JSON response is extracted by the following expression:

```
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)
```

The tutorial application doesn't parse the JSON result, only displaying it in the console window.



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 tutorial application before creating the recognizer to recognize intents in German. See Supported Languages.

Continuous recognition from a file

The following code illustrates two additional capabilities of intent recognition using the Speech SDK. The first,

previously mentioned, is continuous recognition, where the recognizer emits events when results are available. These events can then be processed by event handlers that you provide. With continuous recognition, you call the recognizer's `StartContinuousRecognitionAsync()` to start recognition instead of `RecognizeOnceAsync()`.

The other capability is reading the audio containing the speech to be processed from a WAV file. This 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, replace the body of the `RecognizeIntentAsync()` method with the following 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").
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($"CANCELED: ErrorDetails={e.ErrorDetails}");
                Console.WriteLine($"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 audio file. Then build and run.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code from this article in the samples/csharp/sharedcontent/console folder.

Next steps

How to recognize speech

Recognize intents from speech by using the Speech SDK for C++

10/19/2018 • 6 minutes to read • Edit Online

The Microsoft Cognitive Services Speech SDK provides a way to recognize **intents from speech** and is supported by the Cognitive Services Language Understanding service (LUIS).

1. Create a speech configuration with a LUIS subscription key and region as parameters. The LUIS subscription key is called **endpoint key** in the service documentation. You can't use the LUIS authoring key. (See the note later in this section.)
2. Create an intent recognizer from the speech configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Get the language understanding model that's based on your **AppId**. Add the intents you require.
4. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results (includes intents). If you don't tie up the events, your application receives only a final transcription result.
5. Start intent recognition. For single-shot recognition, such as command or query recognition, use the `RecognizeOnceAsync()` method. This method returns the first recognized utterance. For long-running recognition, use the `StartContinuousRecognitionAsync()` method. Tie up the events for asynchronous recognition results.

See the following code snippets for intent recognition scenarios that use the Speech SDK. Replace the values in the sample with your own LUIS subscription key (endpoint key), the region of your subscription, and the **AppId** of your intent model.

> **NOTE**
> In contrast to other services supported by the Speech SDK, intent recognition requires a specific subscription key (LUIS endpoint key). For information about the intent recognition technology, see the LUIS website. For information on how to acquire the **endpoint key**, see Create a LUIS endpoint key.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
#include <speechapi_cxx.h>

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

Intent recognition from a microphone

The following code shows how to recognize intent from microphone input in the default language (en-US).

```
// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
// The default recognition language is "en-us".
auto config = SpeechConfig::FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using microphone as audio input.
auto recognizer = IntentRecognizer::FromConfig(config);

// Creates a Language Understanding model using the app id, and adds specific intents from your model
auto model = LanguageUnderstandingModel::FromAppId("YourLanguageUnderstandingAppId");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

cout << "Say something...\n";

// Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
// so it is suitable only for single shot recognition like command or query. For long-running
// 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: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
        cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}
```

Intent recognition for a specified language

The following code shows how to recognize intent from microphone input in a specified language. In this example, the language is German (de-de).

```
// 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 service subscription key
// and service region (e.g., "westus").
auto config = SpeechConfig::FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer in the specified language using microphone as audio input.
auto lang = "de-de";
config->SetSpeechRecognitionLanguage(lang);
auto recognizer = IntentRecognizer::FromConfig(config);

// Creates a Language Understanding model using the app id, and adds specific intents from your model
auto model = LanguageUnderstandingModel::FromAppId("YourLanguageUnderstandingAppId");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

cout << "Say something in " << lang << "..." << std::endl;

// Performs recognition. RecognizeOnceAsync() returns when the first utterance has been recognized,
// so it is suitable only for single shot recognition like command or query. For long-running
// 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: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
        cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}
```

Intent recognition from a file with events

The following code shows how to recognize intent in the default language (en-US) in a continuous way. The code allows access to additional information like intermediate results. Input is taken from an audio file. The supported format is single-channel (mono) WAV/PCM with a sampling rate of 16 kHz.

NOTE

Need a sample audio? Right-click this WAV file and choose **Save target as**.

```
// 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").
auto config = SpeechConfig::FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using file as audio input.
// Replace with your own audio file name.
auto audioInput = AudioConfig::FromWavFileInput("whatstheweatherlike.wav");
auto recognizer = IntentRecognizer::FromConfig(config, audioInput);

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

// 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");

// Subscribes to events.
recognizer->Recognizing.Connect([] (const IntentRecognitionEventArgs& e)
{
    cout << "Recognizing:" << e.Result->Text << std::endl;
});

recognizer->Recognized.Connect([] (const IntentRecognitionEventArgs& e)
{
    if (e.Result->Reason == ResultReason::RecognizedIntent)
    {
        cout << "RECOGNIZED: Text=" << e.Result->Text << std::endl;
        cout << "  Intent Id: " << e.Result->IntentId << std::endl;
        cout << "  Intent Service JSON: " << e.Result-
>Properties.GetProperty(PropertyId::LanguageUnderstandingServiceResponse_JsonResult) << std::endl;
    }
    else if (e.Result->Reason == ResultReason::RecognizedSpeech)
    {
        cout << "RECOGNIZED: Text=" << e.Result->Text << " (intent could not be recognized)" << std::endl;
    }
    else if (e.Result->Reason == ResultReason::NoMatch)
    {
        cout << "NOMATCH: Speech could not be recognized." << std::endl;
    }
});

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

    if (e.Reason == CancellationReason::Error)
    {
        cout << "CANCELED: ErrorDetails=" << e.ErrorDetails << std::endl;
        cout << "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.
});

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

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

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

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/cpp/windows/console folder.

Next steps

- How to recognize speech
- How to translate speech

Recognize intents from speech by using the Speech SDK for Java

10/19/2018 • 5 minutes to read • Edit Online

The Microsoft Cognitive Services Speech SDK provides a way to recognize **intents from speech** and is supported by the Cognitive Services Language Understanding service (LUIS).

1. Create a speech configuration with a LUIS subscription key and region as parameters. The LUIS subscription key is called **endpoint key** in the service documentation. You can't use the LUIS authoring key. (See the note later in this section.)
2. Create an intent recognizer from the speech configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Get the language understanding model that's based on your **AppId**. Add the intents you require.
4. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results (includes intents). If you don't tie up the events, your application receives only a final transcription result.
5. Start intent recognition. For single-shot recognition, such as command or query recognition, use the `RecognizeOnceAsync()` method. This method returns the first recognized utterance. For long-running recognition, use the `StartContinuousRecognitionAsync()` method. Tie up the events for asynchronous recognition results.

See the following code snippets for intent recognition scenarios that use the Speech SDK. Replace the values in the sample with your own LUIS subscription key (endpoint key), the region of your subscription, and the **AppId** of your intent model.

> **NOTE**
> In contrast to other services supported by the Speech SDK, intent recognition requires a specific subscription key (LUIS endpoint key). For information about the intent recognition technology, see the LUIS website. For information on how to acquire the **endpoint key**, see Create a LUIS endpoint key.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.*;
import com.microsoft.cognitiveservices.speech.intent.*;
```

Intent recognition from a microphone

The following code shows how to recognize intent from microphone input in the default language (en-US).

```
// Creates an instance of a speech config with specified
// subscription key (called 'endpoint key' by the Language Understanding service)
// and service region. Replace with your own subscription (endpoint) key
// and service region (e.g., "westus2").
// The default language is "en-us".
SpeechConfig config = SpeechConfig.fromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using microphone as audio input.
IntentRecognizer recognizer = new IntentRecognizer(config);

// Creates a language understanding model using the app id, and adds specific intents from your model
LanguageUnderstandingModel model = LanguageUnderstandingModel.fromAppId("YourLanguageUnderstandingAppId");
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

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

// Starts recognition. It returns when the first utterance has been recognized.
IntentRecognitionResult result = recognizer.recognizeOnceAsync().get();

// Checks result.
if (result.getReason() == ResultReason.RecognizedIntent) {
    System.out.println("RECOGNIZED: Text=" + result.getText());
    System.out.println("    Intent Id: " + result.getIntentId());
    System.out.println("    Intent Service Json: " +
result.getProperties().getProperty(PropertyId.SpeechServiceResponse_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: ErrorDetails=" + cancellation.getErrorDetails());
        System.out.println("CANCELED: Did you update the subscription info?");
    }
}
```

Intent recognition for a specified language

The following code shows how to recognize intent from microphone input in a specified language. In this example, the language is German (de-de).

```
// Creates an instance of a speech config with specified
// subscription key (called 'endpoint key' by the Language Understanding service)
// and service region. Replace with your own subscription (endpoint) key
// and service region (e.g., "westus2").
SpeechConfig config = SpeechConfig.fromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer in the specified language using microphone as audio input.
String lang = "de-de";
config.setSpeechRecognitionLanguage(lang);
IntentRecognizer recognizer = new IntentRecognizer(config);

// Creates a language understanding model using the app id, and adds specific intents from your model
LanguageUnderstandingModel model = LanguageUnderstandingModel.fromAppId("YourLanguageUnderstandingAppId");
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

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

// Starts recognition. It returns when the first utterance has been recognized.
IntentRecognitionResult result = recognizer.recognizeOnceAsync().get();

// Checks result.
if (result.getReason() == ResultReason.RecognizedIntent) {
    System.out.println("RECOGNIZED: Text=" + result.getText());
    System.out.println("    Intent Id: " + result.getIntentId());
    System.out.println("    Intent Service Json: " +
result.getProperties().getProperty(PropertyId.SpeechServiceResponse_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: ErrorDetails=" + cancellation.getErrorDetails());
        System.out.println("CANCELED: Did you update the subscription info?");
    }
}
```

Intent recognition from a file with events

The following code shows how to recognize intent in the default language (en-US) in a continuous way. The code allows access to additional information like intermediate results. Input is taken from an audio file. The supported format is single-channel (mono) WAV/PCM with a sampling rate of 16 kHz.

> **NOTE**
>
> Need a sample audio? Right-click this WAV file and choose **Save target as**.

```
// Creates an instance of a speech config with specified
// subscription key (called 'endpoint key' by the Language Understanding service)
// and service region. Replace with your own subscription (endpoint) key
// and service region (e.g., "westus2").
SpeechConfig config = SpeechConfig.fromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using file as audio input.
// Replace with your own audio file name.
AudioConfig audioInput = AudioConfig.fromWavFileInput("YourAudioFile.wav");
IntentRecognizer recognizer = new IntentRecognizer(config, audioInput);

// 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");

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

recognizer.recognized.addEventListener((s, e) -> {
    if (e.getResult().getReason() == ResultReason.RecognizedIntent) {
        System.out.println("RECOGNIZED: Text=" + e.getResult().getText());
        System.out.println("    Intent Id: " + e.getResult().getIntentId());
        System.out.println("    Intent Service Json: " +
e.getResult().getProperties().getProperty(PropertyId.SpeechServiceResponse_JsonResult));
    }
    else if (e.getResult().getReason() == ResultReason.RecognizedSpeech) {
        System.out.println("RECOGNIZED: Text=" + e.getResult().getText());
        System.out.println("    Intent not recognized.");
    }
    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: ErrorDetails=" + e.getErrorDetails());
        System.out.println("CANCELED: Did you update the subscription info?");
    }
});

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

System.out.println("Press any key to stop...");
new Scanner(System.in).nextLine();

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

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/java/jre/console folder.

Next steps

- How to recognize speech
- How to translate speech

Translate speech with the Cognitive Services Speech SDK for C#

10/19/2018 • 4 minutes to read • Edit Online

The Microsoft Cognitive Services Speech SDK provides the simplest way to use **speech translation** in your application. The SDK provides the full functionality of the service. The basic process for performing speech translation includes the following steps:

1. Create a speech translation configuration and provide a Speech service subscription key (or an authorization token) and a region as parameters. Change the configuration as needed. For example, you can configure the source and target translation languages, as well as specify whether you want text or speech output.
2. Create a translation recognizer from the speech translation configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results, as well as a synthesis event for the optional audio output. Otherwise, your application receives only a final transcription result.
4. Start recognition. For single-shot translation use the `RecognizeOnceAsync()` method, which returns the first recognized utterance. For long-running translations, use the `StartContinuousRecognitionAsync()` method and tie up the events for asynchronous recognition results.

See the following code snippets for speech translation scenarios that use the Speech SDK.

> **NOTE**
> Information on how to setup a development project for your platform and preferred development environment can be found in the Quickstarts articles in this documentation. In any case, you will need a subscription key, see Try the speech service for free.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Translation;
#if NET461
using System.Media;
#endif
```

Translation from the microphone

The following code snippet shows you how to translate speech input from English to German and gets the voice output of the translated text. The code uses the microphone.

```
// Translation source language.
// Replace with a language of your choice.
string fromLanguage = "en-US";

// Voice name of synthesis output.
const string GermanVoice = "de-DE-Hedda";

// Creates an instance of a speech translation config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
var config = SpeechTranslationConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
config.SpeechRecognitionLanguage = fromLanguage;
config.VoiceName = GermanVoice;

// Translation target language(s).
// Replace with language(s) of your choice.
config.AddTargetLanguage("de");

// Creates a translation recognizer using microphone as audio input.
using (var recognizer = new TranslationRecognizer(config))
{
    // Subscribes to events.
    recognizer.Recognizing += (s, e) =>
    {
        Console.WriteLine($"RECOGNIZING in '{fromLanguage}': Text={e.Result.Text}");
        foreach (var element in e.Result.Translations)
        {
            Console.WriteLine($"    TRANSLATING into '{element.Key}': {element.Value}");
        }
    };

    recognizer.Recognized += (s, e) =>
    {
        if (e.Result.Reason == ResultReason.TranslatedSpeech)
        {
            Console.WriteLine($"RECOGNIZED in '{fromLanguage}': Text={e.Result.Text}");
            foreach (var element in e.Result.Translations)
            {
                Console.WriteLine($"    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.Synthesizing += (s, e) =>
    {
        var audio = e.Result.GetAudio();
        Console.WriteLine(audio.Length != 0
            ? $"AudioSize: {audio.Length}"
            : $"AudioSize: {audio.Length} (end of synthesis data)");

        if (audio.Length > 0)
        {
            #if NET461
            using (var m = new MemoryStream(audio))
            {
                SoundPlayer simpleSound = new SoundPlayer(m);
                simpleSound.PlaySync();
            }
            #endif
        }
```

```
    };

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

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

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

    recognizer.SessionStopped += (s, e) =>
    {
        Console.WriteLine("\nSession stopped event.");
    };

    // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
    Console.WriteLine("Say something...");
    await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

    do
    {
        Console.WriteLine("Press Enter to stop");
    } while (Console.ReadKey().Key != ConsoleKey.Enter);

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

Translation from file input

The following code snippet shows you how to translate speech input from English to German and French. The code uses a file as input.

```
// Translation source language.
// Replace with a language of your choice.
string fromLanguage = "en-US";

// Creates an instance of a speech translation config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
var config = SpeechTranslationConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
config.SpeechRecognitionLanguage = fromLanguage;

// Translation target language(s).
// Replace with language(s) of your choice.
config.AddTargetLanguage("de");
config.AddTargetLanguage("fr");

var stopTranslation = new TaskCompletionSource<int>();

// Creates a translation recognizer using file as audio input.
// Replace with your own audio file name.
using (var audioInput = AudioConfig.FromWavFileInput(@"whatstheweatherlike.wav"))
{
    using (var recognizer = new TranslationRecognizer(config, audioInput))
    {
        // Subscribes to events.
        recognizer.Recognizing += (s, e) =>
        {
```

```
{
    Console.WriteLine($"RECOGNIZING in '{fromLanguage}': Text={e.Result.Text}");
    foreach (var element in e.Result.Translations)
    {
        Console.WriteLine($"    TRANSLATING into '{element.Key}': {element.Value}");
    }
};

recognizer.Recognized += (s, e) => {
    if (e.Result.Reason == ResultReason.TranslatedSpeech)
    {
        Console.WriteLine($"RECOGNIZED in '{fromLanguage}': Text={e.Result.Text}");
        foreach (var element in e.Result.Translations)
        {
            Console.WriteLine($"    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($"CANCELED: ErrorDetails={e.ErrorDetails}");
        Console.WriteLine($"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);

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

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

```
    }
}
```

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/csharp/sharedcontent/console folder.

Next steps

- How to recognize speech
- How to recognize intents from speech

Translate speech with the Cognitive Services Speech SDK for C++

10/19/2018 • 3 minutes to read • Edit Online

The Microsoft Cognitive Services Speech SDK provides the simplest way to use **speech translation** in your application. The SDK provides the full functionality of the service. The basic process for performing speech translation includes the following steps:

1. Create a speech translation configuration and provide a Speech service subscription key (or an authorization token) and a region as parameters. Change the configuration as needed. For example, you can configure the source and target translation languages, as well as specify whether you want text or speech output.
2. Create a translation recognizer from the speech translation configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results, as well as a synthesis event for the optional audio output. Otherwise, your application receives only a final transcription result.
4. Start recognition. For single-shot translation use the `RecognizeOnceAsync()` method, which returns the first recognized utterance. For long-running translations, use the `StartContinuousRecognitionAsync()` method and tie up the events for asynchronous recognition results.

See the following code snippets for speech translation scenarios that use the Speech SDK.

> **NOTE**
> Information on how to setup a development project for your platform and preferred development environment can be found in the Quickstarts articles in this documentation. In any case, you will need a subscription key, see Try the speech service for free.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
#include <string>
#include <vector>
#include <speechapi_cxx.h>

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

Translation from the microphone

The following code snippet shows you how to translate speech input from English to German and gets the voice output of the translated text. The code uses the microphone.

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

// Sets source and target languages
// Replace with the languages of your choice.
auto fromLanguage = "en-US";
config->SetSpeechRecognitionLanguage(fromLanguage);
config->AddTargetLanguage("de");
config->AddTargetLanguage("fr");

// Creates a translation recognizer using microphone as audio input.
auto recognizer = TranslationRecognizer::FromConfig(config);
cout << "Say something...\n";

// Starts translation. RecognizeOnceAsync() returns when the first utterance has been recognized,
// so it is suitable only for single shot recognition like command or query. For long-running
// recognition, use StartContinuousRecognitionAsync() instead.
auto result = recognizer->RecognizeOnceAsync().get();

// Checks result.
if (result->Reason == ResultReason::TranslatedSpeech)
{
    cout << "RECOGNIZED: Text=" << result->Text << std::endl
         << "  Language=" << fromLanguage << std::endl;

    for (const auto& it : result->Translations)
    {
        cout << "TRANSLATED into '" << it.first.c_str() << "': " << it.second.c_str() << std::endl;
    }
}
else if (result->Reason == ResultReason::RecognizedSpeech)
{
    cout << "RECOGNIZED: Text=" << result->Text << " (text could not be translated)" << std::endl;
}
else if (result->Reason == ResultReason::NoMatch)
{
    cout << "NOMATCH: Speech could not be recognized." << std::endl;
}
else if (result->Reason == ResultReason::Canceled)
{
    auto cancellation = CancellationDetails::FromResult(result);
    cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

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

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/cpp/windows/console folder.

Next steps

- How to recognize speech
- How to recognize intents from speech

Translate speech with the Speech SDK for Java

10/19/2018 • 4 minutes to read • Edit Online

The Microsoft Cognitive Services Speech SDK provides the simplest way to use **speech translation** in your application. The SDK provides the full functionality of the service. The basic process for performing speech translation includes the following steps:

1. Create a speech translation configuration and provide a Speech service subscription key (or an authorization token) and a region as parameters. Change the configuration as needed. For example, you can configure the source and target translation languages, as well as specify whether you want text or speech output.
2. Create a translation recognizer from the speech translation configuration. Provide an audio configuration if you want recognize from a source other than your default microphone (for example, audio stream or audio file).
3. Tie up the events for asynchronous operation, if desired. The recognizer then calls your event handlers when it has interim and final results, as well as a synthesis event for the optional audio output. Otherwise, your application receives only a final transcription result.
4. Start recognition. For single-shot translation use the `RecognizeOnceAsync()` method, which returns the first recognized utterance. For long-running translations, use the `StartContinuousRecognitionAsync()` method and tie up the events for asynchronous recognition results.

See the following code snippets for speech translation scenarios that use the Speech SDK.

> **NOTE**
> Information on how to setup a development project for your platform and preferred development environment can be found in the Quickstarts articles in this documentation. In any case, you will need a subscription key, see Try the speech service for free.

Top-level declarations

For all code in the following sections, these top-level declarations should be in place:

```
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.audio.*;
import com.microsoft.cognitiveservices.speech.translation.*;
```

Translation from the microphone

The following code snippet shows you how to translate speech input from English to German and gets the voice output of the translated text. The code uses the microphone.

```
// Creates an instance of a speech translation config with specified
// subscription key and service region. Replace with your own subscription key
// and service region (e.g., "westus").
SpeechTranslationConfig config = SpeechTranslationConfig.fromSubscription("YourSubscriptionKey",
"YourServiceRegion");

// Sets source and target language(s).
```

```
String fromLanguage = "en-US";
config.setSpeechRecognitionLanguage(fromLanguage);
config.addTargetLanguage("de");

// Sets voice name of synthesis output.
String GermanVoice = "de-DE-Hedda";
config.setVoiceName(GermanVoice);

// Creates a translation recognizer using microphone as audio input.
TranslationRecognizer recognizer = new TranslationRecognizer(config);
{
    // Subscribes to events.
    recognizer.recognizing.addEventListener((s, e) -> {
        System.out.println("RECOGNIZING in '" + fromLanguage + "': Text=" + e.getResult().getText());

        Map<String, String> map = e.getResult().getTranslations();
        for(String element : map.keySet()) {
            System.out.println("    TRANSLATING into '" + element + "': " + map.get(element));
        }
    });

    recognizer.recognized.addEventListener((s, e) -> {
        if (e.getResult().getReason() == ResultReason.TranslatedSpeech) {
            System.out.println("RECOGNIZED in '" + fromLanguage + "': Text=" + e.getResult().getText());

            Map<String, String> map = e.getResult().getTranslations();
            for(String element : map.keySet()) {
                System.out.println("    TRANSLATED into '" + element + "': " + map.get(element));
            }
        }
        if (e.getResult().getReason() == ResultReason.RecognizedSpeech) {
            System.out.println("RECOGNIZED: Text=" + e.getResult().getText());
            System.out.println("    Speech not translated.");
        }
        else if (e.getResult().getReason() == ResultReason.NoMatch) {
            System.out.println("NOMATCH: Speech could not be recognized.");
        }
    });

    recognizer.synthesizing.addEventListener((s, e) -> {
        System.out.println("Synthesis result received. Size of audio data: " +
e.getResult().getAudio().length);
    });

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

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

    recognizer.sessionStarted.addEventListener((s, e) -> {
        System.out.println("\nSession started event.");
    });

    recognizer.sessionStopped.addEventListener((s, e) -> {
        System.out.println("\nSession stopped event.");
    });

    // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
    System.out.println("Say something...");
    recognizer.startContinuousRecognitionAsync().get();

    System.out.println("Press any key to stop");
    new Scanner(System.in).nextLine();

    recognizer.stopContinuousRecognitionAsync().get();
```

```
    recognizer.stopContinuousRecognitionAsync().get();
}
```

Translation from file input

The following code snippet shows you how to translate speech input from English to German and French. The code uses a file as input.

```
private static Semaphore stopTranslationWithFileSemaphore;

public static void translationWithFileAsync() throws InterruptedException, ExecutionException
{
    stopTranslationWithFileSemaphore = new Semaphore(0);

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

    // Sets source and target languages
    String fromLanguage = "en-US";
    config.setSpeechRecognitionLanguage(fromLanguage);
    config.addTargetLanguage("de");
    config.addTargetLanguage("fr");

    // Creates a translation recognizer using file as audio input.
    // Replace with your own audio file name.
    AudioConfig audioInput = AudioConfig.fromWavFileInput("YourAudioFile.wav");
    TranslationRecognizer recognizer = new TranslationRecognizer(config, audioInput);
    {
        // Subscribes to events.
        recognizer.recognizing.addEventListener((s, e) -> {
            System.out.println("RECOGNIZING in '" + fromLanguage + "': Text=" + e.getResult().getText());

            Map<String, String> map = e.getResult().getTranslations();
            for(String element : map.keySet()) {
                System.out.println("    TRANSLATING into '" + element + "': " + map.get(element));
            }
        });

        recognizer.recognized.addEventListener((s, e) -> {
            if (e.getResult().getReason() == ResultReason.TranslatedSpeech) {
                System.out.println("RECOGNIZED in '" + fromLanguage + "': Text=" + e.getResult().getText());

                Map<String, String> map = e.getResult().getTranslations();
                for(String element : map.keySet()) {
                    System.out.println("    TRANSLATED into '" + element + "': " + map.get(element));
                }
            }
            if (e.getResult().getReason() == ResultReason.RecognizedSpeech) {
                System.out.println("RECOGNIZED: Text=" + e.getResult().getText());
                System.out.println("    Speech not translated.");
            }
            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: ErrorDetails=" + e.getErrorDetails());
                System.out.println("CANCELED: Did you update the subscription info?");
            }
```

```
            stopTranslationWithFileSemaphore.release();;
        });

        recognizer.sessionStarted.addEventListener((s, e) -> {
            System.out.println("\nSession started event.");
        });

        recognizer.sessionStopped.addEventListener((s, e) -> {
            System.out.println("\nSession stopped event.");

            // Stops translation when session stop is detected.
            System.out.println("\nStop translation.");
            stopTranslationWithFileSemaphore.release();;
        });

        // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
        System.out.println("Start translation...");
        recognizer.startContinuousRecognitionAsync().get();

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

        // Stops translation.
        recognizer.stopContinuousRecognitionAsync().get();
    }
}
```

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Look for the code that's used in this article in the samples/java/jre/console folder.

Next steps

- How to recognize speech
- How to recognize intents from speech

Use "Text to Speech" in Speech Service

10/19/2018 • 2 minutes to read • Edit Online

The Speech service provides Text to Speech functionality through a straightforward HTTP request. You `POST` the text to be spoken to the appropriate endpoint, and the service returns an audio file (`.wav`) containing synthesized speech. Your application can then use this audio as it likes.

The body of the POST request for Text to Speech may be plain text (ASCII or UTF8) or an SSML document. Plain-text requests are spoken with a default voice. In most cases, you want to use an SSML body. The HTTP request must include an authorization token.

The regional Text to Speech endpoints are shown here. Use the one appropriate to your subscription.

REGION	TEXT TO SPEECH ENDPOINT
West US	`https://westus.tts.speech.microsoft.com/cognitiveservices/v1`
West US2	`https://westus2.tts.speech.microsoft.com/cognitiveservices/v1`
East US	`https://eastus.tts.speech.microsoft.com/cognitiveservices/v1`
East US2	`https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1`
East Asia	`https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1`
South East Asia	`https://southeastasia.tts.speech.microsoft.com/cognitiveservices/`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`
West Europe	`https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1`

NOTE

If you created a custom voice font, use the endpoint you created for it instead of the endpoints here.

Specify a voice

To specify a voice, use the `<voice>` SSML tag. For example:

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
  <voice  name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>
    Hello, world!
  </voice>
</speak>
```

See Text to Speech voices for a list of the available voices and their names.

Make a request

A Text to Speech HTTP request is made in POST mode with the text to be spoken in the body of the request. The maximum length of the HTTP request body is 1024 characters. The request must have the following headers:

HEADER	VALUES	COMMENTS
`Content-Type`	`application/ssml+xml`	The input text format.

`X-Microsoft-OutputFormat`	`raw-16khz-16bit-mono-pcm`
	`riff-16khz-16bit-mono-pcm`
	`raw-8khz-8bit-mono-mulaw`
	`riff-8khz-8bit-mono-mulaw`
	`audio-16khz-128kbitrate-mono-mp3`
	`audio-16khz-64kbitrate-mono-mp3`
	`audio-16khz-32kbitrate-mono-mp3`
	`raw-24khz-16bit-mono-pcm`
	`riff-24khz-16bit-mono-pcm`
	`audio-24khz-160kbitrate-mono-mp3`
	`audio-24khz-96kbitrate-mono-mp3`
	`audio-24khz-48kbitrate-mono-mp3`
		The output audio format.

`User-Agent`	Application name	The application name is required and must be fewer than 255 characters.
`Authorization`	Authorization token obtained by presenting your subscription key to the token service. Each token is valid for ten minutes. See REST APIs: Authentication.	

NOTE

If your selected voice and output format have different bit rates, the audio is resampled as necessary.

A sample request is shown below.

```
POST /cognitiveservices/v1
HTTP/1.1
Host: westus.tts.speech.microsoft.com
X-Microsoft-OutputFormat: riff-24khz-16bit-mono-pcm
Content-Type: application/ssml+xml
User-Agent: Test TTS application
Authorization: (authorization token)

<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>
    Hello, world!
</voice> </speak>
```

The response body with a status of 200 contains audio in the specified output format.

```
HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: audio/x-wav

Response audio payload
```

If an error occurs, the status codes below are used. The response body for the error also contains a description of the problem.

CODE	DESCRIPTION	PROBLEM

CODE	DESCRIPTION	PROBLEM
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.
413	Request Entity Too Large	The input SSML is too large or contains more than 3 `<voice>` elements.
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.

For more information on the Text to Speech REST API, see REST APIs.

Next steps

- Get your Speech trial subscription
- Recognize speech in C++
- Recognize speech in C#
- Recognize speech in Java

Batch transcription

11/1/2018 • 5 minutes to read • Edit Online

Batch transcription is ideal if you have large amounts of audio in storage. Using our Rest API, You can point to audio files by SAS URI and asynchronously receive transcriptions.

Batch transcription API

The Batch transcription API offers asynchronous speech to text transcription, along with additional features. It is a REST API exposing methods for:

1. Creating batch processing requests
2. Query Status
3. Downloading transcriptions

> **NOTE**
> The Batch transcription API is ideal for call centers, which typically accumulate thousands of hours of audio. The API is guided by a "fire and forget" philosophy, which makes it easy to transcribe large volume of audio recordings.

Supported formats

The Batch transcription API supports the following formats:

NAME	CHANNEL
mp3	Mono
mp3	Stereo
wav	Mono
wav	Stereo

For stereo audio streams, Batch transcription splits the left and right channel during the transcription. The two JSON files with the result are each created from a single channel. The timestamps per utterance enable the developer to create an ordered final transcript. The following JSON sample shows the output of a channel.

```
      {
       "recordingsUrl": "https://mystorage.blob.core.windows.net/cris-e2e-
datasets/TranscriptionsDataset/small_sentence.wav?st=2018-04-19T15:56:00Z&se=2040-04-
21T15:56:00Z&sp=rl&sv=2017-04-17&sr=b&sig=DtvXbMYquDWQ2OkhAenGuyZI%2BYgaa3cyvdQoHKIBGdQ%3D",
       "resultsUrls": {
           "channel_0": "https://mystorage.blob.core.windows.net/bestor-87a0286f-304c-4636-b6bd-
b3a96166df28/TranscriptionData/24265e4c-e459-4384-b572-5e3e7795221f?sv=2017-04-
17&sr=b&sig=IY2qd%2Fkgtz2PwRe2C88BphH4Hv%2F1VCb1UVJ33xsw%2BEY%3D&se=2018-04-23T14:48:24Z&sp=r"
       },
       "statusMessage": "None.",
       "id": "0bb95356-ff06-469d-acc7-81f9144a269a",
       "createdDateTime": "2018-04-20T14:11:57.167",
       "lastActionDateTime": "2018-04-20T14:12:54.643",
       "status": "Succeeded",
       "locale": "en-US"
   },
```


NOTE

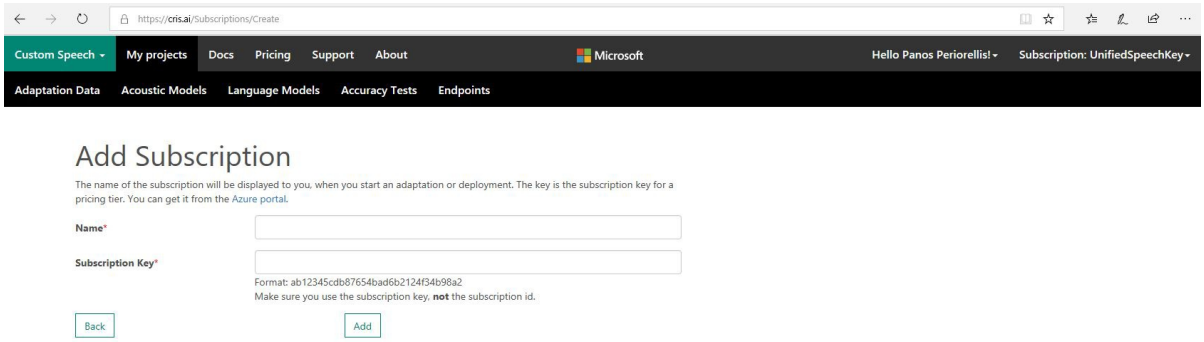
The Batch transcription API is using a REST service for requesting transcriptions, their status, and associated results. You can use the API from any language. The next section describes how it is used.

Authorization token

As with all features of the Speech Service, you create a subscription key from the Azure portal following our Get-Started guide. If you plan to get transcriptions from our baseline models then this is all you need to do.

If you plan on customizing and using a custom model then you need to add this subscritpion key to the custom speech portal as follows:

1. Sign in to Custom Speech.
2. Select **Subscriptions**.
3. Select **Connect Existing Subscription**.
4. Add the Subscription key and an alias in the view that pops up



5. Copy and paste that key in the client code in the following sample.

> **NOTE**
>
> If you plan to use a custom model, you will need the ID of that model too. Note that this is not the endpoint ID that you find on the Endpoint Details view. It is the model ID that you can retrieve when you select the details of that model.

Sample code

Customize the following sample code with a subscription key and an API key. This allows you to obtain a bearer token.

```
     public static CrisClient CreateApiV2Client(string key, string hostName, int port)

        {
            var client = new HttpClient();
            client.Timeout = TimeSpan.FromMinutes(25);
            client.BaseAddress = new UriBuilder(Uri.UriSchemeHttps, hostName, port).Uri;
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", key);

            return new CrisClient(client);
        }
```

After you obtain the token, you must specify the SAS URI pointing to the audio file requiring transcription. The rest of the code iterates through the status and displays results. Initially one would set up the key, region, models to use and the SA. as shown the code snippet below. This is followed by instantiation of the client and the POST request.

```
            private const string SubscriptionKey = "<your Speech subscription key>";
            private const string HostName = "westus.cris.ai";
            private const int Port = 443;

            // SAS URI
            private const string RecordingsBlobUri = "some SAS URI";

            // adapted model Ids
            private static Guid AdaptedAcousticId = new Guid("some guid");
            private static Guid AdaptedLanguageId = new Guid("some guid");

            // Creating a Batch transcription API Client
            var client = CrisClient.CreateApiV2Client(SubscriptionKey, HostName, Port);

            var transcriptionLocation = await client.PostTranscriptionAsync(Name, Description, Locale, new
Uri(RecordingsBlobUri), new[] { AdaptedAcousticId, AdaptedLanguageId }).ConfigureAwait(false);
```

Now that the request has been made the user can query and download transcription results as the code snippet shows.

```
            // get all transcriptions for the user
            transcriptions = await client.GetTranscriptionAsync().ConfigureAwait(false);

            // for each transcription in the list we check the status
            foreach (var transcription in transcriptions)
            {
                switch(transcription.Status)
                {
                    case "Failed":
                    case "Succeeded":

                            // we check to see if it was one of the transcriptions we created from this
client.
                        if (!createdTranscriptions.Contains(transcription.Id))
                        {
                            // not creted form here, continue
                            continue;
                        }

                        completed++;

                        // if the transcription was successfull, check the results
                        if (transcription.Status == "Succeeded")
                        {
                            var resultsUri = transcription.ResultsUrls["channel_0"];
                            WebClient webClient = new WebClient();
                            var filename = Path.GetTempFileName();
                            webClient.DownloadFile(resultsUri, filename);
                            var results = File.ReadAllText(filename);
                            Console.WriteLine("Transcription succedded. Results: ");
                            Console.WriteLine(results);
                        }

                    break;
                    case "Running":
                    running++;
                     break;
                    case "NotStarted":
                    notStarted++;
                    break;

                    }
                }
            }
        }
```

Our Swagger document provides full details on the above calls. The full sample shown here is on GitHub.

> **NOTE**
> In the preceding code, the subscription key is from the Speech resource that you create on the Azure portal. Keys obtained from the Custom Speech Service resource do not work.

Notice the asynchronous setup for posting audio and receiving transcription status. The client created is a .NET Http client. There is a `PostTranscriptions` method for sending the audio file details, and a `GetTranscriptions` method to receive the results. `PostTranscriptions` returns a handle, and `GetTranscriptions` uses this handle to create a handle to obtain the transcription status.

The current sample code does not specify any custom models. The service uses the baseline models for transcribing the file or files. To specify the models, you can pass on the same method the model IDs for the acoustic and the language model.

If you don't want to use the baseline, you must pass model Ids for both acoustic and language models.

> **NOTE**
>
> For baseline transcription, you don't have to declare the endpoints of the baseline models. If you want to use custom models, you provide their endpoints IDs as the Sample. If you want to use an acoustic baseline with a baseline language model, you only have to declare the custom model's endpoint ID. Microsoft detects the partner baseline model (be it acoustic or language), and uses that to fulfill the transcription request.

Supported storage

Currently the only storage supported is Azure Blob storage.

Downloading the sample

The sample shown here is on GitHub.

> **NOTE**
>
> Typically, an audio transcription requires a time span equal to the duration of the audio file, plus a 2-3 minute overhead.

Next steps

- Get your Speech trial subscription

Get the Cognitive Services Speech Devices SDK

10/19/2018 • 2 minutes to read • Edit Online

The Speech Devices SDK is in restricted preview and requires you to be enrolled in the program. Currently, Microsoft prefers large companies as candidates for access to this product.

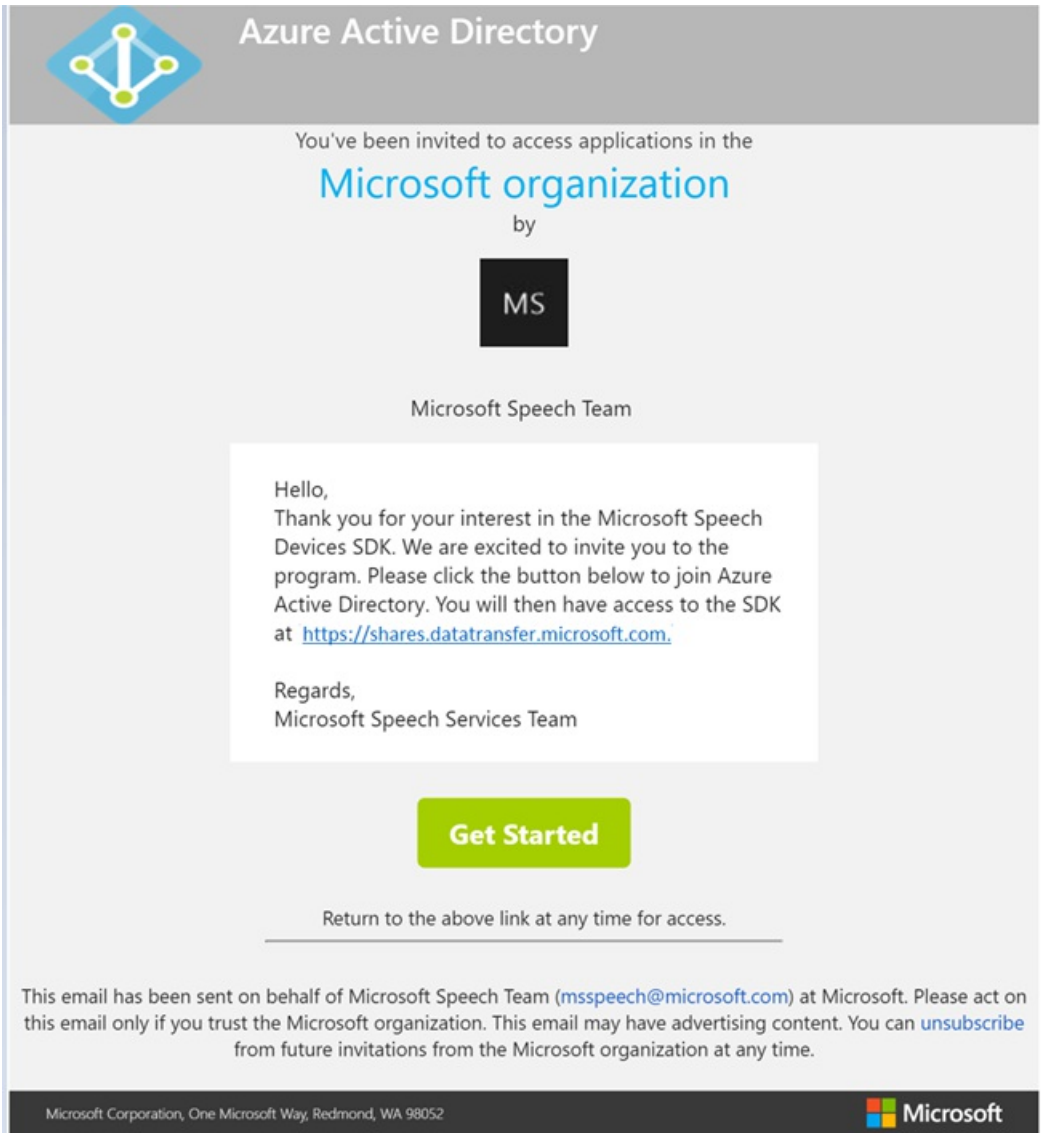
Request access

To get access to the Speech Devices SDK:

1. Go to the Microsoft Speech Devices SDK sign-up form.
2. Read the license agreement.
3. If you agree to the terms of the license agreement, select **I agree**.
4. Answer the questions in the form.
5. Submit the form.
6. If your email address is not already part of Azure Active Directory (Azure AD), you receive an invitation email like the following example when you're approved for access. If your email address is already in Azure AD, you receive an email message from the Microsoft Speech Team when you're approved for access, and you can skip ahead to Download the Speech Devices SDK.

Approval e-mail

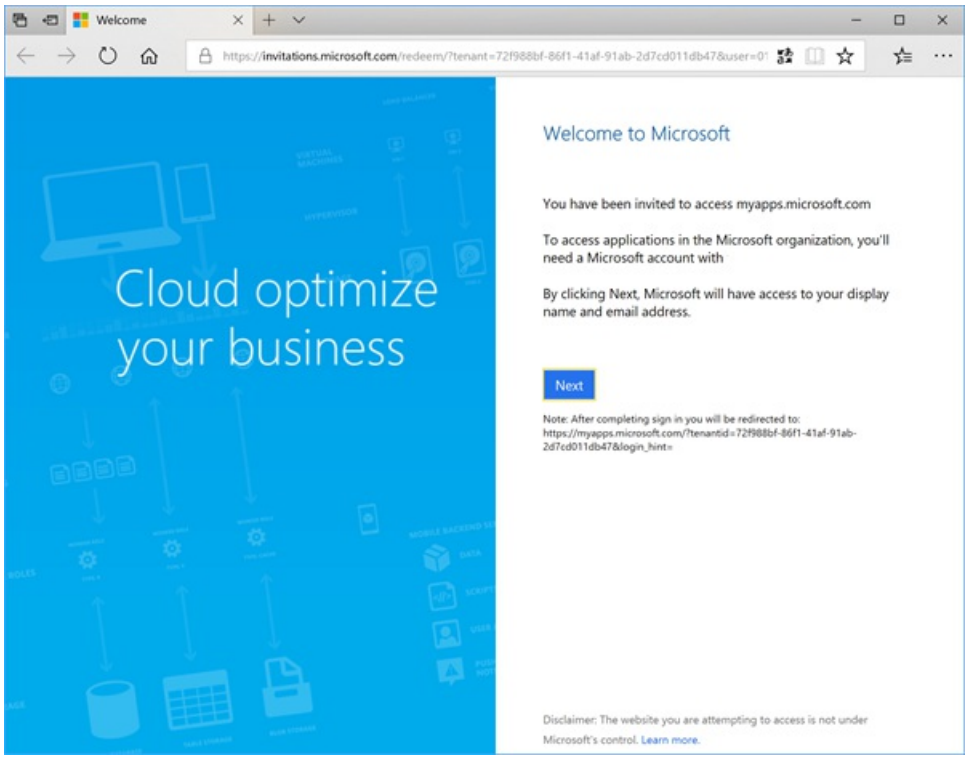
```
From: Microsoft Speech Team from Microsoft (via Microsoft) <invites@microsoft.com>
Subject: You're invited to the Microsoft organization
```



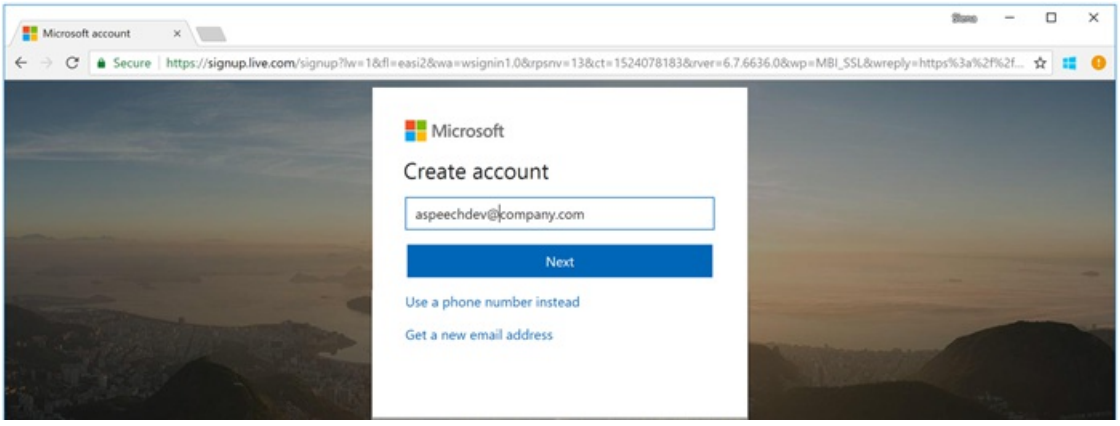
Accept access

Complete the following steps to join Azure AD with the email address you provided during registration. This process grants you access to the Speech Devices SDK download site.

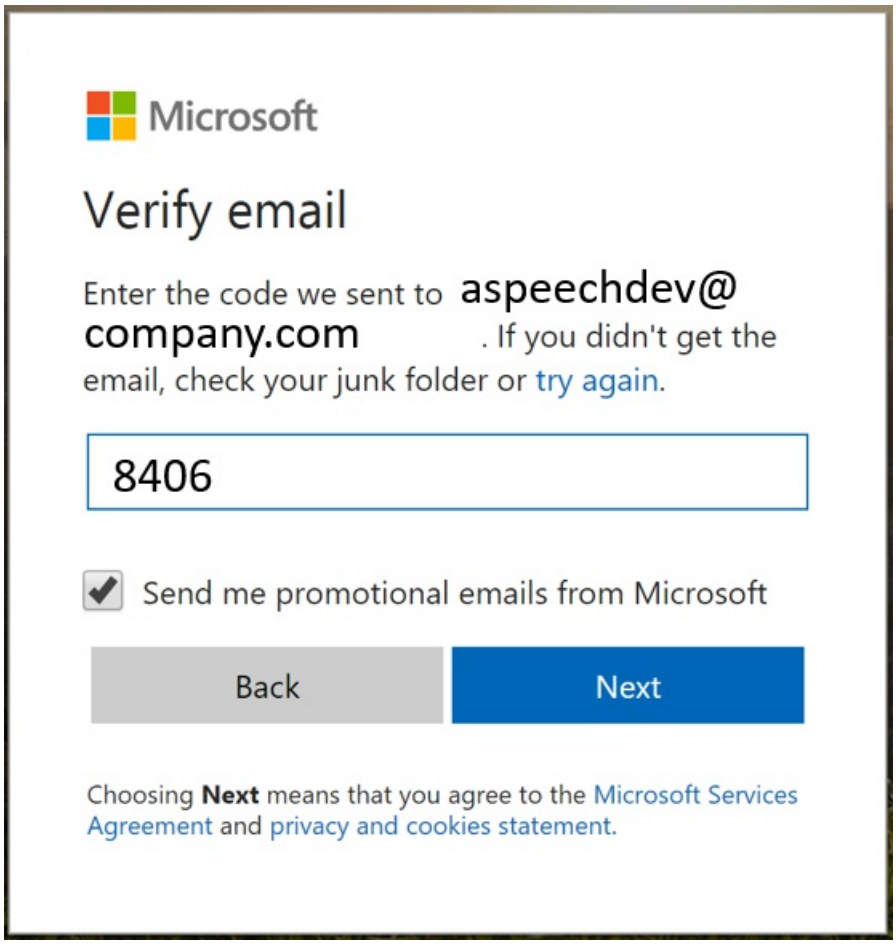
1. In the email message you received, select **Get Started**. If your organization is already an Office 365 customer, you are prompted to sign in and you can skip ahead to step 8.
2. In the browser window that opens, select **Next**.



3. Create a Microsoft account if you don't already have one. Enter the same email address at which you received the invitation email.



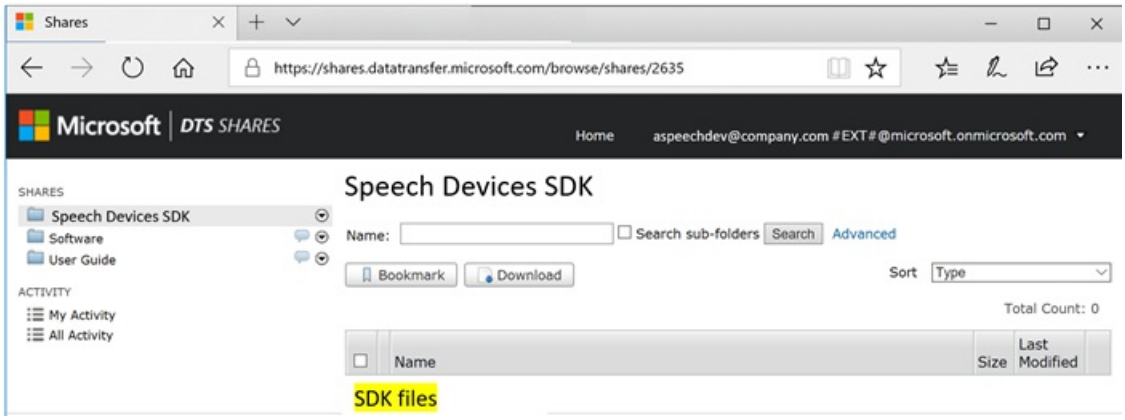
4. Select **Next** to create a password.
5. When prompted to verify your e-mail, get the verification code from the invitation email you received.
6. Paste or type the security code from the email message in the dialog box. In this example, the security code is **8406**. Select **Next**.



7. When you see the Access Panel Application in the browser, you have confirmed that your email address is part of Azure AD. You now have access to the Speech Devices SDK download site.

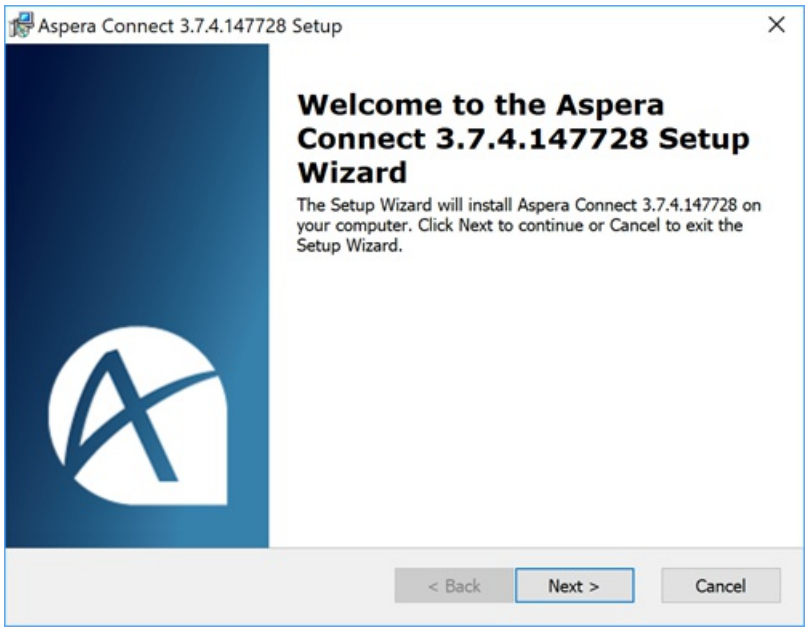
Download the Speech Devices SDK

Go to the Speech Devices SDK download site. Sign in with the Microsoft account you created earlier.

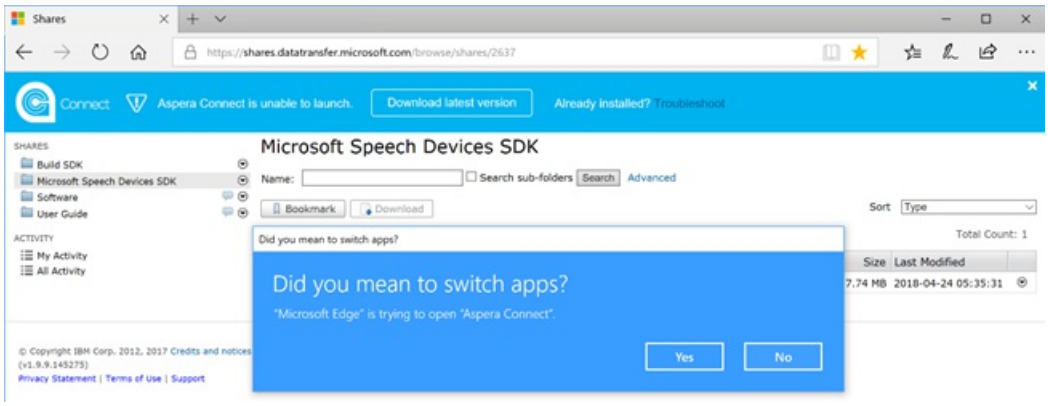


To download the Speech Devices SDK, associated sample code, and reference material:

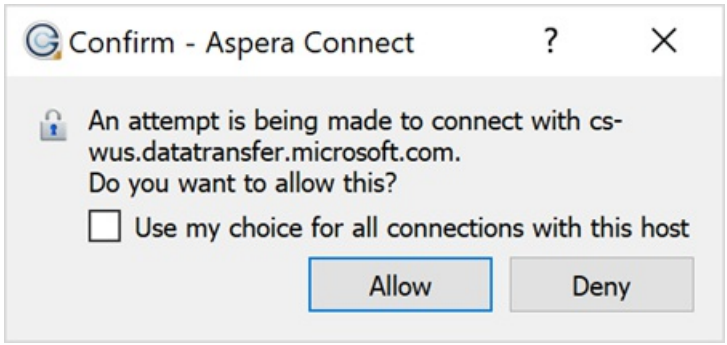
1. Download and install the Aspera Connect tool when prompted in the browser.



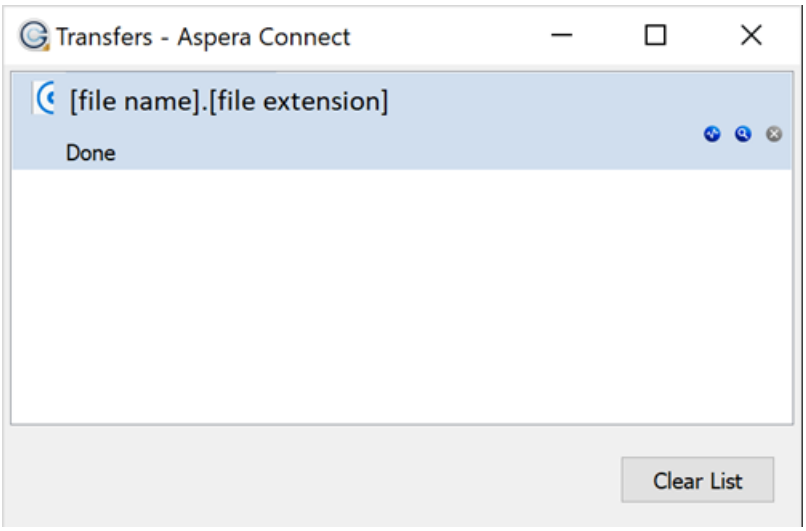
2. Select **Yes** to switch apps to Aspera Connect.



3. Select **Allow** to confirm downloading the files by using Aspera Connect.



4. Close the Aspera Connect Transfers window after the files are downloaded.



By default, the files are downloaded to your **Downloads** folder. You can sign out of this site now.

Next steps

Get started with the Speech Devices SDK

Tutorial: Create a custom acoustic model

10/29/2018 • 8 minutes to read • Edit Online

Creating a custom acoustic model is helpful if your application is designed for use in a particular environment, such as a car, with specific recording devices or conditions, or by a particular user population. Examples involve accented speech, specific background noises, or using a specific microphone for recording.

In this article, you learn how to:

- Prepare the data
- Import the acoustic dataset
- Create the custom acoustic model

If you don't have an Azure Cognitive Services account, create a free account before you begin.

Prerequisites

Ensure that your Cognitive Services account is connected to a subscription by opening the Cognitive Services Subscriptions page.

You can connect to a Speech Service subscription that was created in the Azure portal by selecting **Connect existing subscription**.

For information about creating a Speech Service subscription in the Azure portal, see Try the Speech Service for free.

Prepare the data

To customize an acoustic model for a particular domain, a collection of speech data is required. This collection can range from a couple of utterances to several hundred hours of speech. The collection consists of a set of audio files of speech data, and a text file of transcriptions of each audio file. The audio data should be representative of the scenario in which you want to use the recognizer.

For example:

- If you want to better recognize speech in a noisy factory environment, the audio files should consist of people speaking in a noisy factory.
- If you are interested in optimizing performance for a single speaker—for example, you want to transcribe all of FDR's fireside chats—the audio files should consist of many examples of that speaker only.

An acoustic dataset for customizing the acoustic model consists of two parts: (1) a set of audio files containing the speech data and (2) a file containing the transcriptions of all audio files.

Audio data recommendations

- All audio files in the dataset should be stored in the WAV (RIFF) audio format.
- The audio must have a sampling rate of 8 kilohertz (KHz) or 16 KHz, and the sample values should be stored as uncompressed, pulse-code modulation (PCM) 16-bit signed integers (shorts).
- Only single-channel (mono) audio files are supported.
- The audio files can be between 100 microseconds and 1 minute in length, although ideally they should be around 10-12 seconds. Each audio file should ideally start and end with at least 100 microseconds of silence, and somewhere between 500 microseconds and 1 second is common.
- If you have background noise in your data, we recommend that you also have some examples with longer

segments of silence in your data—for example, a few seconds—before and/or after the speech content.

- Each audio file should consist of a single utterance—for example, a single sentence for dictation, a single query, or a single turn of a dialog system.
- Each audio file in the dataset should have a unique file name and a .wav extension.
- The set of audio files should be placed in a single folder without subdirectories, and the entire set of audio files should be packaged as a single .zip-file archive.

> **NOTE**
> Data imports via the web portal are currently limited to 2 GB, so this is the maximum size of an acoustic dataset. This size corresponds to approximately 17 hours of audio that's recorded at 16 KHz or 34 hours of audio that's recorded at 8 KHz. The main requirements for the audio data are summarized in the following table:

PROPERTY	VALUE
File Format	RIFF (WAV)
Sampling Rate	8,000 Hertz (Hz) or 16,000 Hz
Channels	1 (mono)
Sample Format	PCM, 16-bit integers
File Duration	0.1 seconds < duration < 12 seconds
Silence Collar	> 0.1 seconds
Archive Format	.zip
Maximum Archive Size	2 GB

> **NOTE**
> File names should use only latin characters and follow the format 'filename.extention'

Language support

For a full list of languages that are supported for custom **Speech to Text** language models, see Supported languages for the Speech Service.

Transcriptions for the audio dataset

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
```

> **NOTE**
>
> 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 by the user *prior* to uploading the data to the Custom Speech Service. For the appropriate language to use when you prepare your transcriptions, see Transcription guidelines for using the Speech Service.

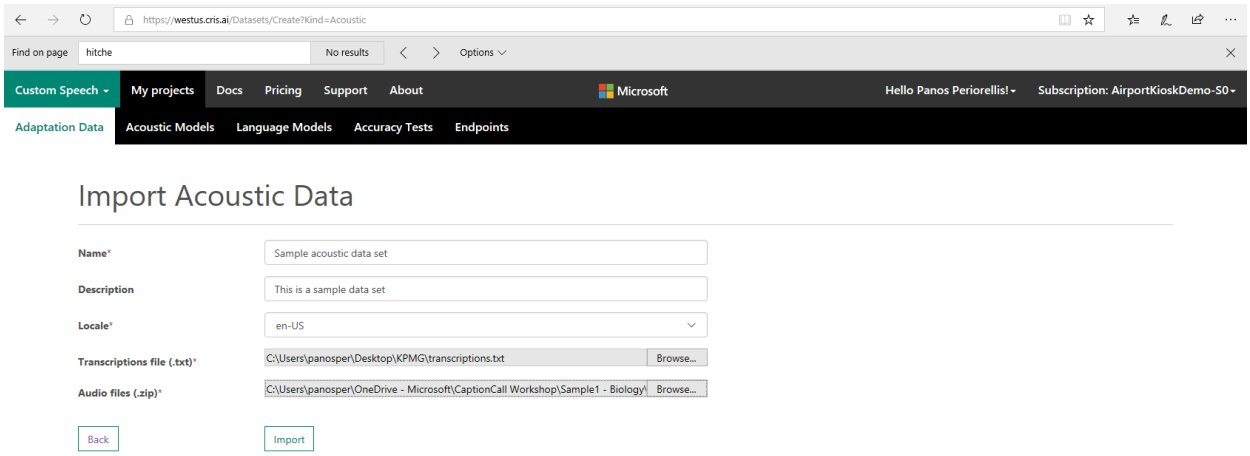
Perform the steps in the next sections by using the Speech Service portal.

Import the acoustic dataset

After you've prepared the audio files and transcriptions, they're ready to be imported to the service web portal.

To import them, first ensure that you're signed in to the Speech Service portal. Then, in the **Custom Speech** drop-down list in the ribbon, select **Adaptation Data**. If this is your first time uploading data to the Custom Speech Service, an empty table labeled **Datasets** is displayed.

In the **Acoustic Datasets** row, select the **Import** button, and the site displays a page for uploading a new dataset.



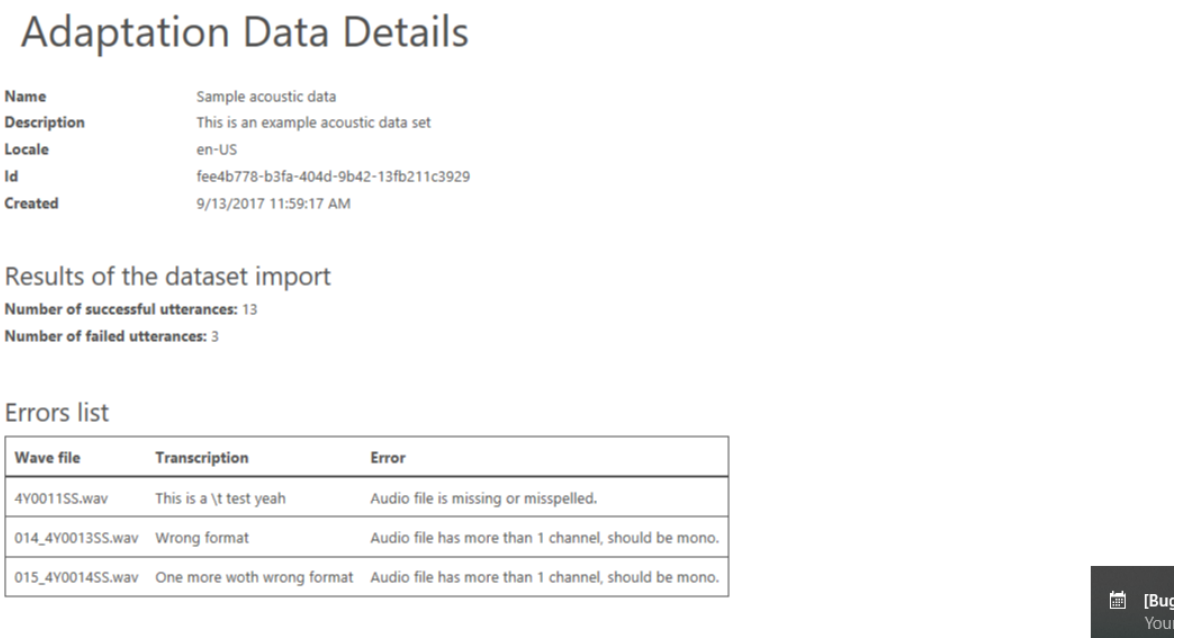
In the **Name** and **Description** boxes, enter the appropriate information. Friendly descriptions are useful for keeping track of the various datasets that you upload.

In the **Transcriptions file (.txt)** and **Audio files (.zip)** boxes, select **Browse**, and then select your plain-text transcription file and the zip archive of WAV files. When the preparation is complete, select **Import** to upload your data. Your data will be uploaded. For larger datasets, the import process might take several minutes.

When the upload is complete, return to the **Acoustic Datasets** table. An entry is displayed that corresponds to your acoustic dataset. Note that it has been assigned a unique ID (GUID). The data displays its current status: *NotStarted* while it is being queued for processing, *Running* while it is undergoing validation, and *Complete* when the data is ready for use.

Data validation includes a series of checks on the audio files to verify the file format, length, and sampling rate, and on the transcription files to verify the file format and perform some text normalization.

When the status is *Succeeded*, you can select **Details** to view the acoustic data verification report. The number of utterances that passed and failed verification are shown, along with details about the failed utterances. In the example in the following image, two WAV files failed verification because of improper audio format. In this dataset, one file has an incorrect sampling rate, and the other has an incorrect file format.



Adaptation Data Details

Name	Sample acoustic data
Description	This is an example acoustic data set
Locale	en-US
Id	fee4b778-b3fa-404d-9b42-13fb211c3929
Created	9/13/2017 11:59:17 AM

Results of the dataset import

Number of successful utterances: 13

Number of failed utterances: 3

Errors list

Wave file	Transcription	Error
4Y0011SS.wav	This is a \t test yeah	Audio file is missing or misspelled.
014_4Y0013SS.wav	Wrong format	Audio file has more than 1 channel, should be mono.
015_4Y0014SS.wav	One more woth wrong format	Audio file has more than 1 channel, should be mono.

If you want to change the name or description of the dataset, you can select the **Edit** link and change their entries. You cannot modify the transcription or audio file entries.

Create a custom acoustic model

After the status of your acoustic dataset is *Complete*, you can use the dataset to create a custom acoustic model. To do so, select **Acoustic Models** in the **Custom Speech** drop-down list. A table labeled **Your models** lists all your custom acoustic models. The table is empty if this is your first use. The table title displays the current locale. Currently, you can create acoustic models for US English only.

To create a new model, select **Create New** under the table title. As before, enter a name and description to help you identify this model. For example, you can use the **Description** field to record which starting model and acoustic dataset you used to create the model.

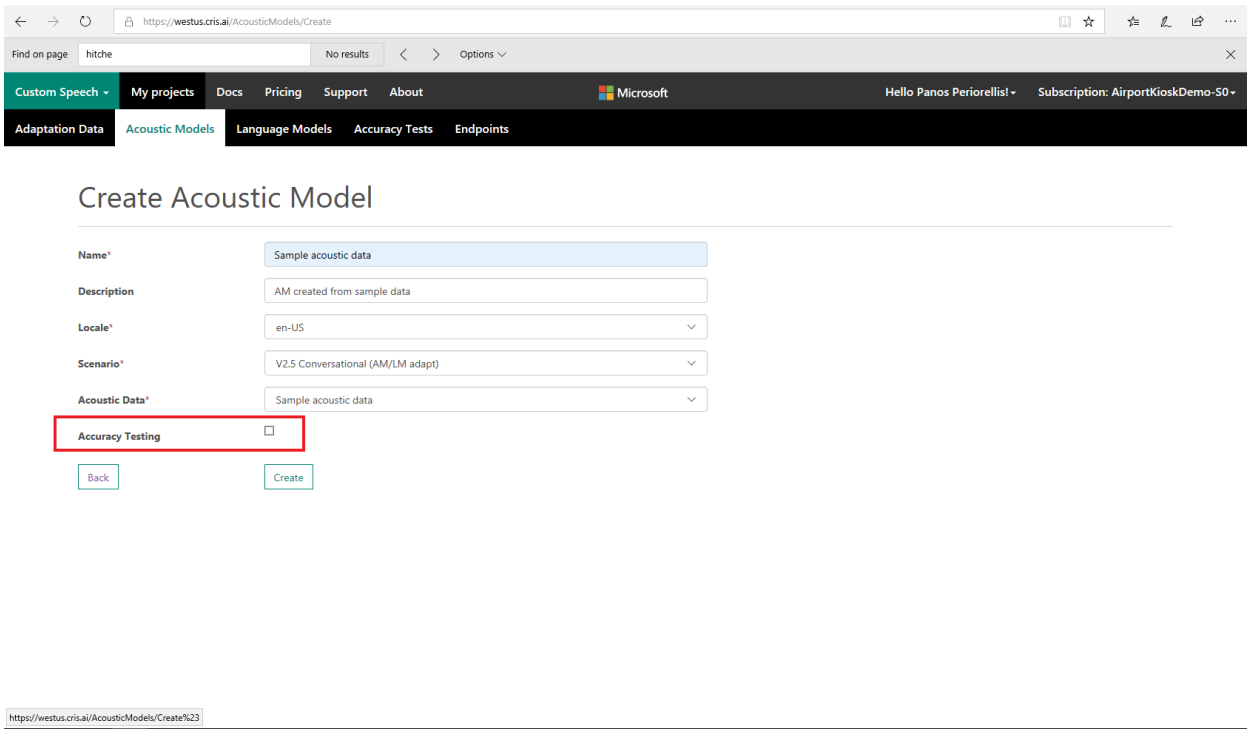
Next, in the **Base Acoustic Model** drop-down list, select a base model. The base model is the starting point for your customization. There are two base acoustic models to choose from:

- The **Microsoft Search and Dictation AM** model is appropriate for speech that's directed at an application, such as commands, search queries, or dictation.
- The **Microsoft Conversational Model** is appropriate for recognizing speech that's spoken in a conversational style. This type of speech is usually directed at another person and occurs in a call center or meetings.

Latency for partial results in Conversational models is higher than in Search and Dictation models.

> **NOTE**
>
> We are currently rolling out our new **Universal** model, which aims to address all scenarios. The aforementioned models will also remain publicly available.

Next, in the **Acoustic Data** drop-down list, select the acoustic data that you want to use to perform the customization.

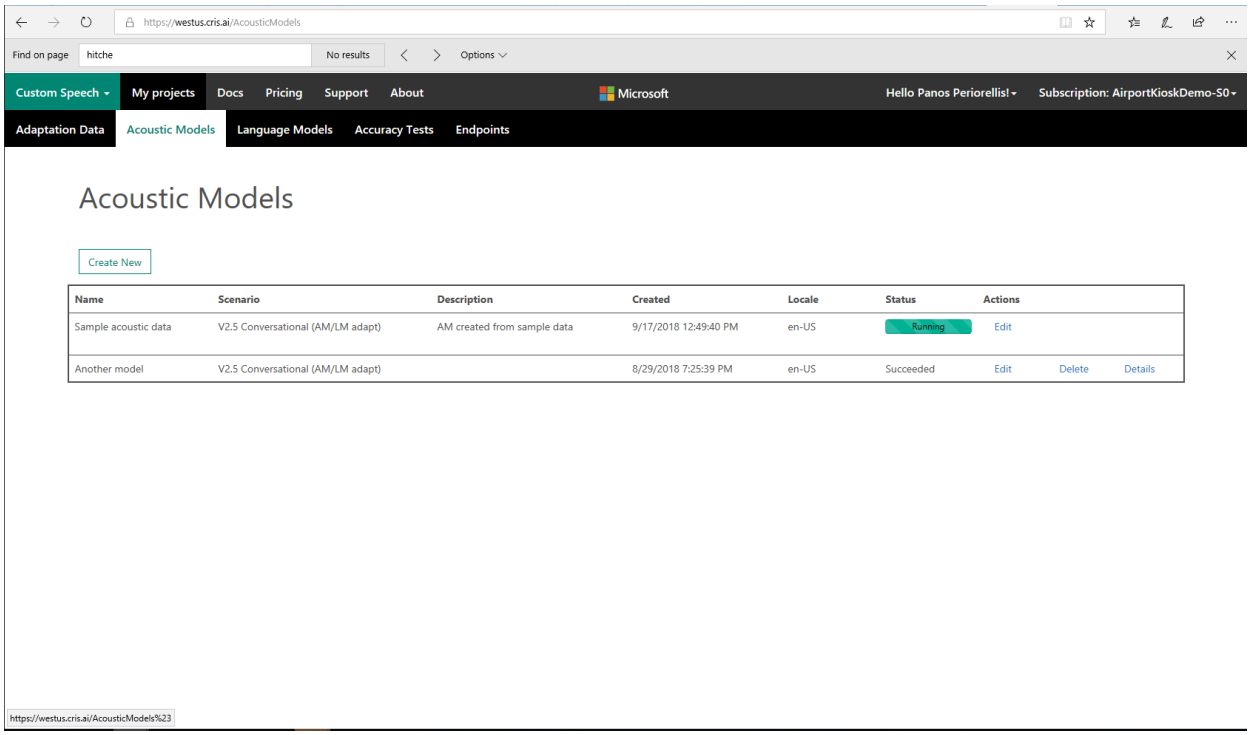


When the processing is complete, you can optionally choose to perform accuracy testing of your new model. This test runs a speech-to-text evaluation on a specified acoustic dataset by using the customized acoustic model and then reports the results. To perform this testing, select the **Accuracy Testing** check box. Then, in the drop-down list, select a language model. If you have not created any custom language models, only the base language models will be displayed in the drop-down list. To select the most appropriate language model, see Tutorial: Create a custom language model.

Finally, select the acoustic dataset that you want to use to evaluate the custom model. If you perform accuracy testing, to get a realistic sense of the model's performance, it's important to select an acoustic dataset that's different from the one you used for the model creation. Testing the accuracy on the training data doesn't allow you to evaluate how the adapted model performs under real conditions. The result will be too optimistic. Also note that accuracy testing is limited to 1,000 utterances. If the acoustic dataset for testing is larger, only the first 1,000 utterances is evaluated.

When you're ready to start running the customization process, select **Create**.

The acoustic models table displays a new entry that corresponds to this new model. The table also displays the status of the process: *Waiting*, *Processing*, or *Complete*.



Next steps

- Get your Speech Service trial subscription
- Recognize speech in C#
- Git Sample Data

Tutorial: Create a custom language model

10/19/2018 • 6 minutes to read • Edit Online

In this document, you create a custom language model. You can then use this custom language model in conjunction with existing state-of-the-art speech models from Microsoft to add voice interaction to your application.

The document discusses how to:

- Prepare the data
- Import the language data set
- Create the custom language model

If you don't have a Cognitive Services account, create a free account before you begin.

Prerequisites

To ensure that your Cognitive Services account is connected to a subscription, open the Cognitive Services subscriptions page.

To connect to a Speech Services subscription that was created in the Azure portal, select the **Connect existing subscription** button.

For information about creating a Speech Services subscription in the Azure portal, see the Get started page.

Prepare the data

To create a custom language model for your application, you need to provide a list of example utterances to the system, for example:

- "The patient has had urticaria for the past week."
- "The patient had a well-healed herniorrhaphy scar."

The sentences do not need to be complete or grammatically correct, but they should accurately reflect the spoken input the system is expected to encounter in deployment. These examples should reflect both the style and content of the task that the users will perform with your application.

The language model data should be written in UTF-8 BOM. The text file should contain one example (sentence, utterance, or query) per line.

If you want certain terms to have a higher weight (importance), you can add several utterances to your data that include those terms.

The main requirements for the language data are summarized in the following table.

PROPERTY	VALUE
Text encoding	UTF-8 BOM
# of utterances per line	1
Maximum file size	1.5 GB

PROPERTY	VALUE
Remarks	Avoid repeating characters more often than four times, for example 'aaaaa'
Remarks	No special characters such as '\t', or any other UTF-8 character above U+00A1 in the Unicode characters table
Remarks	URIs will also be rejected since there is no unique way to pronounce a URI

When the text is imported, it is text-normalized so it can be processed by the system. However, there are some important normalizations that must be done by the user *prior* to uploading the data. See the transcription guidelines to determine the appropriate language to use when preparing your language data.

Language support

See the full list of supported languages for custom **Speech to Text** language models.

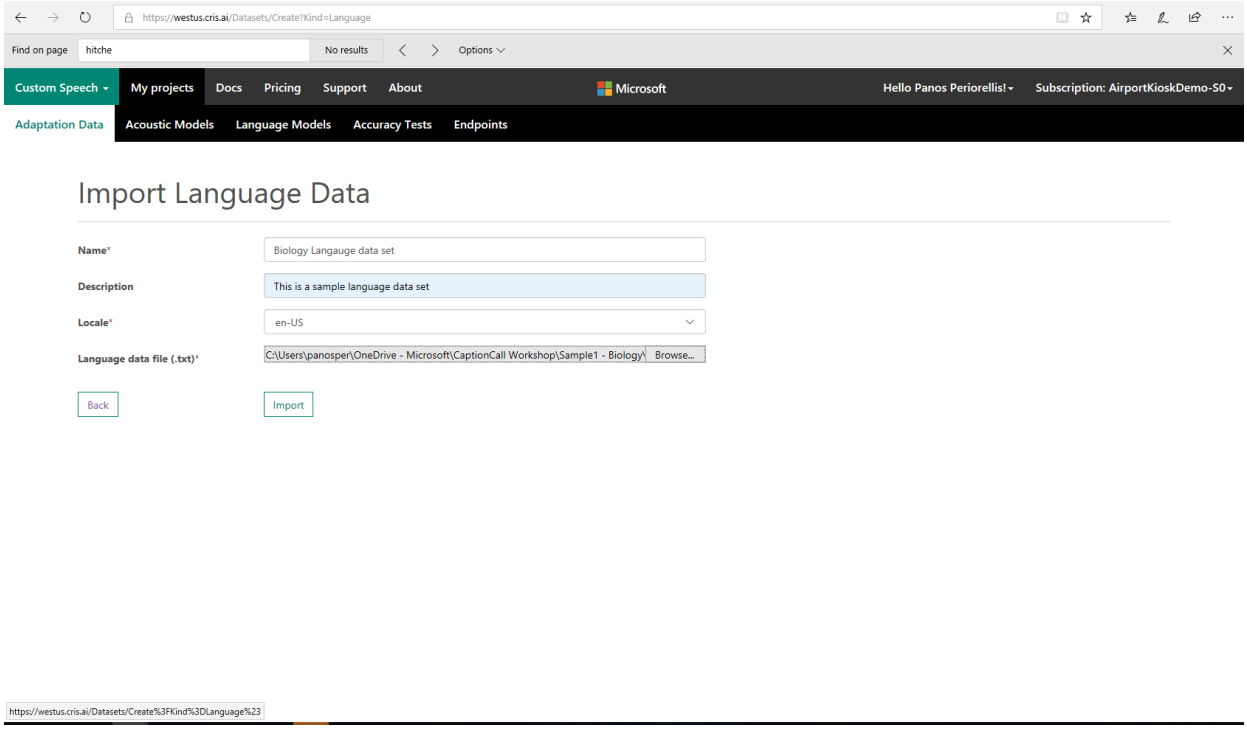
Import the language data set

Select the **Import** button in the **Language Datasets** row, and the site displays a page for uploading a new data set.

When you're ready to import your language data set, sign into the Speech Services portal. First, select the **Custom Speech** drop-down menu on the top ribbon. Then select **Adaptation Data**. The first time you attempt to upload data to Speech Services, you'll see an empty table called **Datasets**.

To import a new data set, select the **Import** button in the **Language Datasets** row. Then the site displays a page for uploading a new data set. Enter a **Name** and **Description** to help you identify the data set in the future, and then choose the locale.

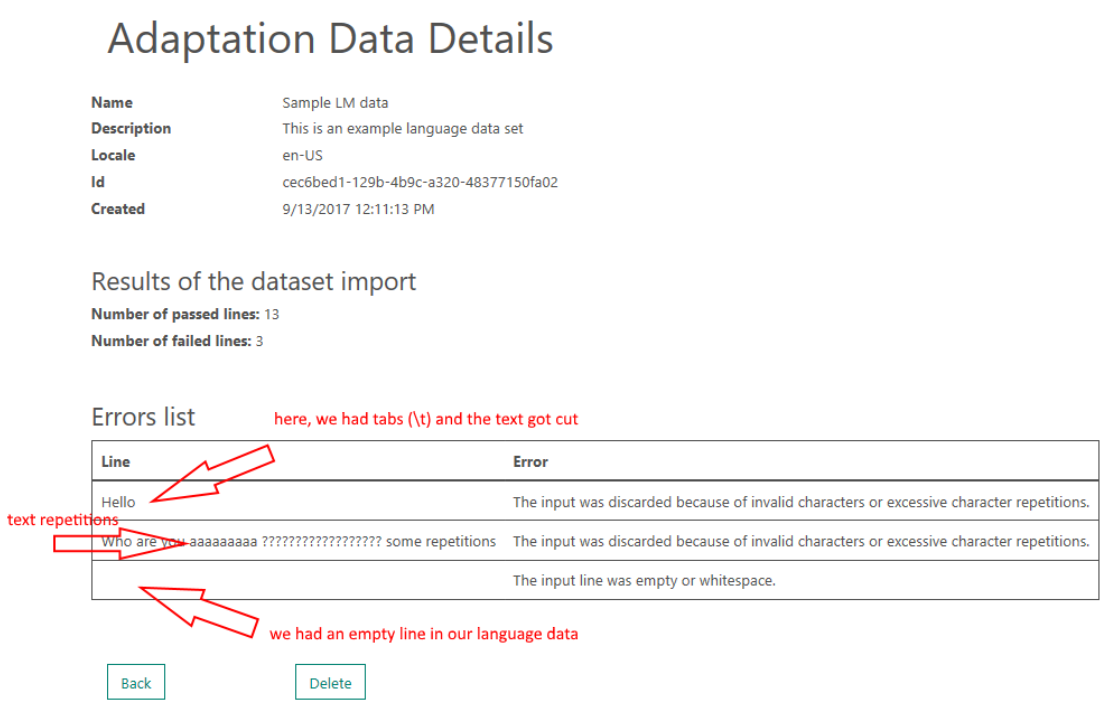
Next, use the **Choose File** button to locate the language data text file. After that, select **Import**, and the data set will be uploaded. Depending on the size of the data set, import might take several minutes.



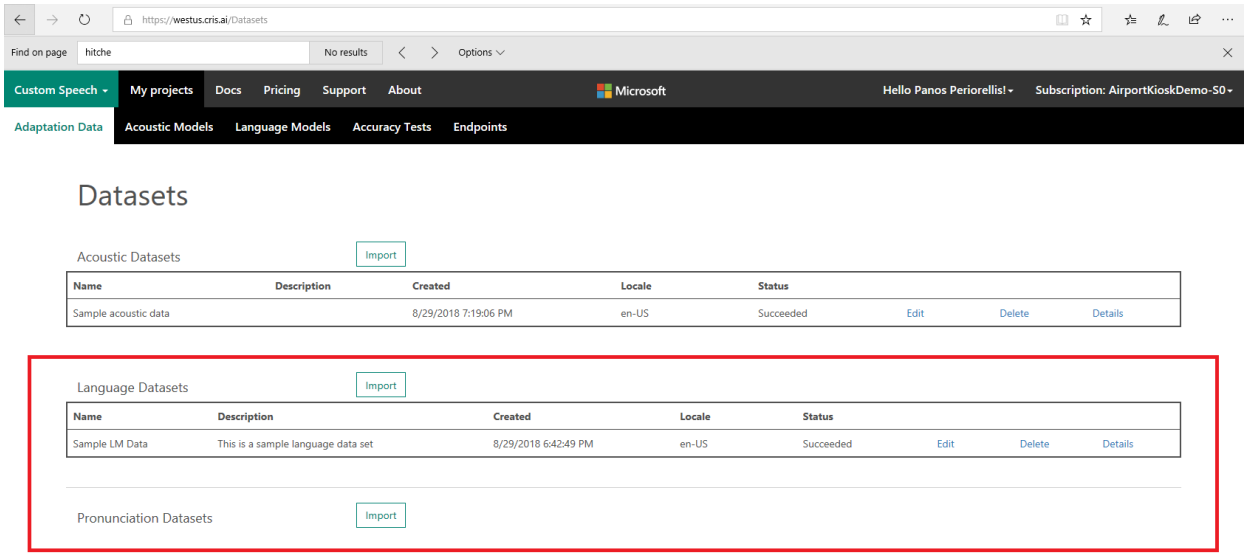
After the import is complete, the language data has an entry that corresponds to your language data set. Notice

that it has been assigned a unique ID (GUID). The data also has a status that reflects its current state. Its status is **Waiting** while it's being queued for processing, **Processing** while it's going through validation, and **Complete** when the data is ready for use. Data validation performs a series of checks on the text in the file. It also does some text normalization of the data.

When the status is **Complete**, you can select **View Report** to see the language data verification report. The number of utterances that passed and failed verification are shown, along with details about the failed utterances. In the following example, two examples failed verification because of incorrect characters. (In this data set, the first line had two tab characters, the second had several characters that aren't part of the ASCII printable character set, and the third line was blank).



When the status of the language data set is **Complete**, it can be used to create a custom language model.



Create a custom language model

After your language data is ready, select **Language Models** from the **Menu** drop-down menu to start the

process of custom language model creation. This page contains a table called **Language Models** with your current custom language models. If you haven't yet created any custom language models, the table will be empty. The current locale is shown in the table next to the relevant data entry.

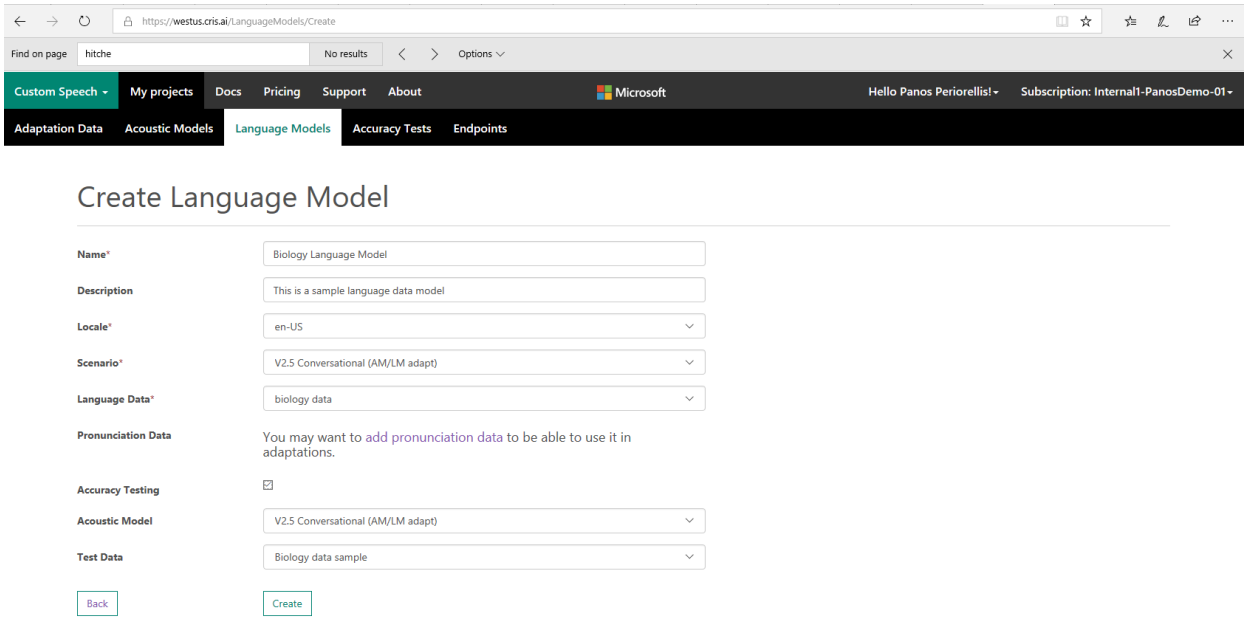
The appropriate locale must be selected before taking any action. The current locale is indicated in the table title on all data, model, and deployment pages. To change the locale, select the **Change Locale** button that's located under the table title. This takes you to a locale confirmation page. Select **OK** to return to the table.

On the Create Language Model page, enter a **Name** and **Description** to help you keep track of pertinent information about this model, such as the data set that's used. Next, select the **Base Language Model** from the drop-down menu. This model is the starting point for your customization.

There are two base language models to choose from. The Search and Dictation model is appropriate for speech that's directed at an application, such as commands, search queries, or dictation. The Conversational model is appropriate for recognizing speech that's spoken in a conversational style. This type of speech is typically directed at another person and occurs in call centers or meetings.

The Search and Dictation model is appropriate for speech that's directed at an application, such as commands, search queries, or dictation. The Conversational model is appropriate for recognizing speech that's spoken in a conversational style. This type of speech is typically directed at another person and occurs in call centers or meetings. A new model called "Universal" is also publicly available. Universal aims to tackle all scenarios and eventually replace the Search and Dictation and the Conversational models.

As shown in the following example, after you've specified the base language model, use the **Language Data** drop-down menu to select the language data set that you want to use for the customization.



As with the acoustic model creation, you can optionally choose to do offline testing of your new model when the processing is complete. Model evaluations require an acoustic data set.

To do offline testing of your language model, select the check box next to **Offline Testing**. Then select an acoustic model from the drop-down menu. If you haven't created any custom acoustic models, the Microsoft base acoustic models will be the only model in the menu. If you've picked a conversational LM base model, you need to use a conversational AM here. If you use a search and dictate LM model, you have to select a search and dictate AM model.

Finally, select the acoustic data set you want to use to do the evaluation.

When you're ready to start processing, select **Create**. Next you'll see the table of language models. There will be a new entry in the table corresponding to this model. The status reflects the model's state and will go through several states including **Waiting**, **Processing**, and **Complete**.

When the model has reached the **Complete** state, it can be deployed to an endpoint. Selecting **View Result** shows the results of offline testing, if you performed it.

If you want to change the **Name** or **Description** of the model at some point, you can use the **Edit** link in the appropriate row of the language models table.

Next steps

- Get your Speech Services trial subscription
- How to recognize speech in C#
- Git sample data

Enable custom pronunciation

10/19/2018 • 2 minutes to read • Edit Online

By using custom pronunciation, you can define the phonetic form and display of a word or term. It is useful for handling customized terms, such as product names or acronyms. All you need is a pronunciation file (a simple .txt file).

Here's how it works. In a single .txt file, you can enter several custom pronunciation entries. The structure is as follows:

```
Display form <Tab> Spoken form <Newline>
```

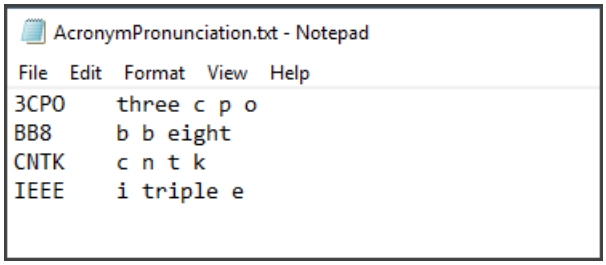
Several examples are shown in the following table:

DISPLAY FORM	SPOKEN FORM
C3PO	see three pea o
L8R	late are
CNTK	see n tea k

Requirements for the spoken form

The spoken form must be lowercase, which you can force during the import. You also need to provide checks in the data importer. No tab in either the spoken form or the display form is permitted. However, there might be more forbidden characters in the display form (for example, ~ and ^).

Each .txt file can have several entries, as shown in the following image:



The spoken form is the phonetic sequence of the display form. It's composed of letters, words, or syllables. Currently, there's no further guidance or set of standards to help you formulate the spoken form.

Supported pronunciation characters

Custom pronunciation is currently supported for English (en-US) and German (de-de). The character sets that you can use to express the spoken form of a term (in the custom pronunciation file) are shown in the following table:

LANGUAGE	CHARACTERS
English (en-US)	a, b, c, d, e, f, g, h, i, j, k, l, o, p, q, r, s, t, u, v, w, x, y, z

LANGUAGE	CHARACTERS
German (de-de)	ä, ö, ü, ?, a, b, c, d, e, f, g, h, i, j, k, l, o, p, q, r, s, t, u, v, w, x, y, z

> **NOTE**
> A term's display form (in a pronunciation file) should be written the same way in a language adaptation dataset.

Requirements for the display form

A display form can be only a custom word, a term, an acronym, or compound words that combine existing words. You can also enter alternative pronunciations for common words.

> **NOTE**
> We don't recommend using this feature to reformulate common words or to modify the spoken form. It is better to run the decoder to see whether some unusual words (such as abbreviations, technical words, or foreign words) are incorrectly decoded. If they are, add them to the custom pronunciation file. In the language model, you should always and only use the display form of a word.

Requirements for the file size

The size of the .txt file that contains the pronunciation entries is limited to 1 megabyte (MB). Usually, you don't need to upload large amounts of data through this file. Most custom pronunciation files are likely to be just a few kilobytes (KBs) in size. The encoding of the .txt file for all locales should be UTF-8 BOM. For the English locale, ANSI is also acceptable.

Next steps

- Improve recognition accuracy by creating a custom acoustic model.
- Improve recognition accuracy by creating a custom language model.

Creating custom voice fonts

10/19/2018 • 10 minutes to read • Edit Online

Text-to-Speech (TTS) voice customization enables you to create a recognizable, one-of-a-kind voice for your brand: a *voice font*.

To create your voice font, you make a studio recording and upload the associated scripts as the training data. The service then creates a unique voice model tuned to your recording. You can use this voice font to synthesize speech.

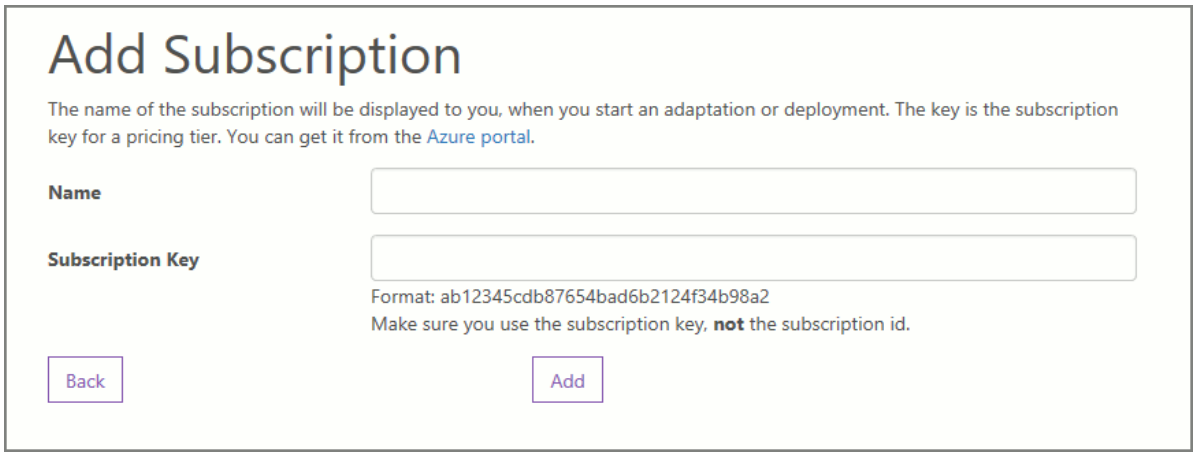
You can get started with a small amount of data for a proof of concept. But the more data you provide, the more natural and professional your voice sounds.

Prerequisites

You need an Azure account and a subscription to the Speech service. Create one if you haven't already. Connect your subscription to the Custom Voice portal as shown here.

1. Sign in to the Custom Voice portal using the same Microsoft account that you used to apply for access.
2. Under your account name on the top right, go to **Subscriptions**.

3. On the Subscriptions page, choose **Connect existing subscription**. Note that Speech Services supports different regions. Check the region where your subscription key was created, and make sure that you connect your key to the correct sub-portal.
4. Paste your subscription key into the table, as shown in the following example. Each subscription has two keys, and you can use either of them.



You're ready to go!

Prepare recordings and transcripts

A voice training dataset consists of a set of audio files, along with a text file that contains the transcripts of the audio files.

You can prepare these files in two ways. Either write a script and have it read by voice talent or use publicly available audio and transcribe it to text. If you do the latter, edit disfluencies from the audio files, such as "um" and other filler sounds, stutters, mumbled words, or mispronunciations.

To produce a good voice font, make the recordings in a quiet room with a high-quality microphone. Consistent volume, speaking rate, speaking pitch, and expressive mannerisms of speech are essential for building a great digital voice.

To create a voice for production use, we recommend you use a professional recording studio and voice talent. For more information, see How to record voice samples for a custom voice.

Audio files

Each audio file should contain a single utterance (for example, a single sentence or a single turn of a dialog system). All files must be in the same language. (Multilanguage custom voices are not supported.) The audio files must also each have a unique numeric filename with the filename extension `.wav`.

Audio files should be prepared as follows. Other formats are unsupported and will be rejected.

PROPERTY	VALUE
File format	RIFF (.wav)
Sampling rate	at least 16,000 Hz
Sample format	PCM, 16-bit
File name	Numeric, with `.wav` extension
Archive format	.zip
Maximum archive size	200 MB

> **NOTE**
>
> .wav files with a sampling rate lower than 16,000 Hertz will be rejected. If a .zip file contains waves with different sampling rates, only those equal to or higher than 16,000 Hz will be imported. The portal currently imports .zip archives up to 200 MB. However, multiple archives can be uploaded. The maximum number of datasets allowed is 10 .zip files for free subscription users and 50 for standard subscription users.

Transcripts

The transcription file is a plain text file (ANSI, UTF-8, UTF-8-BOM, UTF-16-LE, or UTF-16-BE). Each line of the transcription file must have the name of an audio file, followed by a tab (code point 9) character, and finally its transcript. No blank lines are allowed.

For example:

```
0000000001  This is the waistline, and it's falling.
0000000002  We have trouble scoring.
0000000003  It was Janet Maslin.
```

The custom voice system normalizes transcripts by converting the text to lowercase and removing extraneous punctuation. It's important that the transcripts are 100% accurate transcriptions of the corresponding audio recordings.

> **TIP**
>
> When building production Text-to-Speech voices, select utterances (or write scripts) that take into account both phonetic coverage and efficiency. Having trouble getting the results you want? Contact the Custom Voice team to find out more about having us consult.

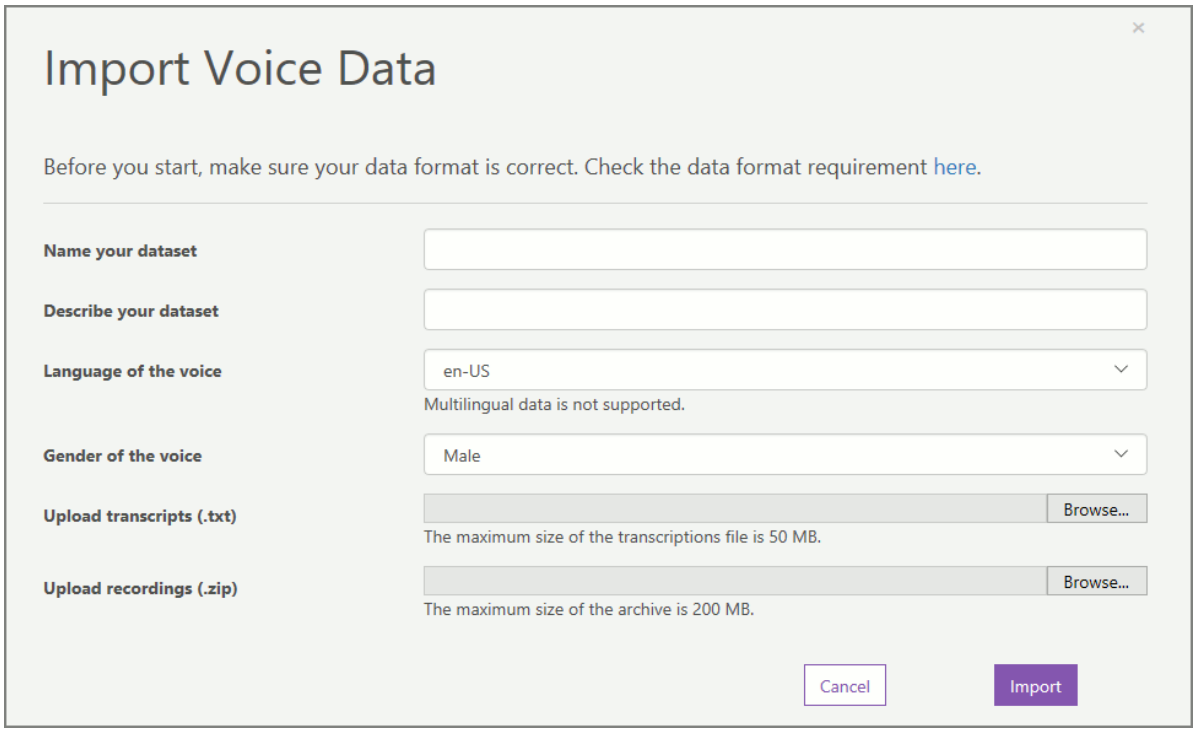
Upload your datasets

After you prepare your audio file archive and transcripts, upload them via the Custom Voice service portal.

> **NOTE**
>
> Datasets cannot be edited after they have been uploaded. If you forget to include transcripts of some of the audio files, for example, or accidentally choose the wrong gender, you must upload the entire dataset again. Check your dataset and settings thoroughly before starting the upload.

1. Sign in to the portal.
2. On the main page, under **Custom Voice**, select **Data**.

   The **My Voice** table appears. It is empty if you haven't uploaded any voice datasets yet.
3. To open the page for uploading a new dataset, select **Import data**.

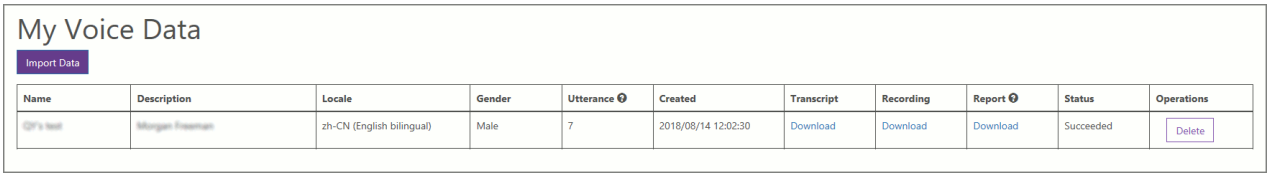


4. Enter a name and description in the fields that are provided.
5. Select the locale for your voice fonts. Make sure the locale information matches the language of the recording data and the scripts.
6. Select the gender of the speaker whose voice you're using.
7. Select the script and audio files to upload.
8. Select **Import** to upload your data. For larger datasets, importing may take several minutes.

> **NOTE**
> Free subscription users can upload two datasets at a time. Standard subscription users can upload five datasets simultaneously. If you reach the limit, wait until at least one of your datasets finishes importing. Then try again.

When the upload is complete, the **My Voice Data** table appears again. You should see an entry that corresponds to the dataset you just uploaded.

Datasets are automatically validated after upload. Data validation includes a series of checks on the audio files to verify their file format, size, and sampling rate. Checks on the transcription files verify the file format and do some text normalization. The utterances are transcribed using speech recognition. Then the resulting text is compared with the transcript you provided.



The following table shows the processing states for imported datasets:

STATE	MEANING
`NotStarted`	Your dataset has been received and is queued for processing.

STATE	MEANING
`Running`	Your dataset is being validated.
`Succeeded`	Your dataset has been validated and may now be used to build a voice font.

After validation is complete, you can see the total number of matched utterances for each of your datasets in the **Utterance** column.

You can download a report to check the pronunciation scores and the noise level for each of your recordings. The pronunciation score ranges from 0 to 100. A score below 70 normally indicates a speech error or script mismatch. A heavy accent can reduce your pronunciation score and impact the generated digital voice.

A higher signal-to-noise ratio (SNR) indicates lower noise in your audio. You can typically reach a 50+ SNR by recording at professional studios. Audio with an SNR below 20 can result in obvious noise in your generated voice.

Consider re-recording any utterances with low pronunciation scores or poor signal-to-noise ratios. If you can't re-record, you might exclude those utterances from your dataset.

Build your voice font

After your dataset has been validated, you can use it to build your custom voice font.

1. In the **Custom Voice** drop-down menu, choose **Models**.

   The **My Voice Fonts** table appears, listing any custom voice fonts you've already created.

2. Under the table title, select **Create voices**.

   The page for creating a voice font appears. The current locale is shown in the first row of the table. Change the locale to create a voice in another language. The locale must be the same as it is for the datasets that are being used to create the voice.

3. As you did when you uploaded your dataset, enter a name and description to help you identify this model.

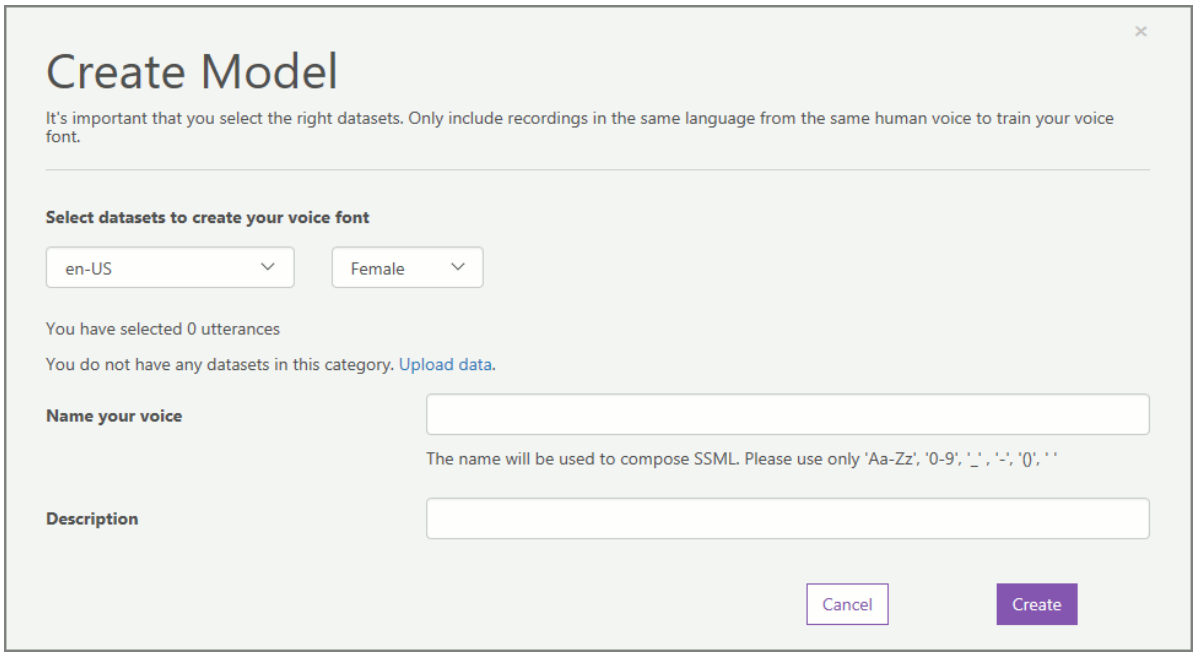
   Choose a name carefully. The name you enter here will be the name you use to specify the voice in your request for speech synthesis as part of the SSML input. Only letters, numbers, and a few punctuation characters such as '-', '_', and '(', ')' are allowed.

   A common use of the **Description** field is to record the names of the datasets that were used to create the model.

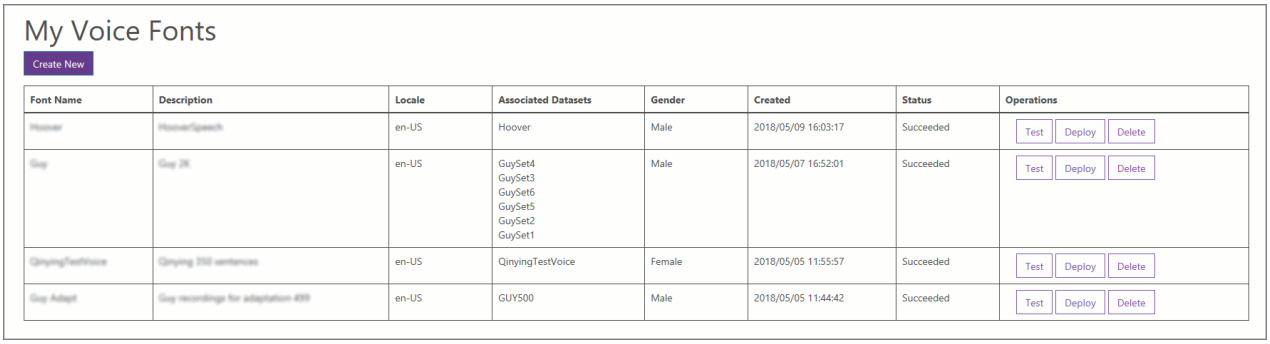
4. Choose the gender of your voice font. It must match the gender of the dataset.

5. Select the dataset(s) that you want to use for training the voice font. All datasets that you use must be from the same speaker.

6. Click **Create** to begin creating your voice font.



Your new model appears in the **My Voice Fonts** table.



The status that's shown reflects the process of converting your dataset to a voice font, as shown here.

STATE	MEANING
`NotStarted`	Your request for voice font creation is queued for processing.
`Running`	Your voice font is being created.
`Succeeded`	Your voice font has been created and may be deployed.

Training time varies depending on the volume of audio data processed. Typical times range from about 30 minutes for hundreds of utterances to 40 hours for 20,000 utterances.

> **NOTE**
>
> Free subscription users can train one voice font at a time. Standard subscription users can train three voices simultaneously. If you reach the limit, wait until at least one of your voice fonts finishes training, and then try again.

Test your voice font

After your voice font is successfully built, you can test it before deploying it for use. In the **Operations** column, select **Test**. The test page appears for the selected voice font. The table is empty if you haven't yet submitted any test requests for the voice.

To display a pop-up menu for submitting text requests, select the **Test with text** button under the table title. You can submit your test request in either plain text or SSML. The maximum input size is 1,024 characters, including all tags for the SSML request. The language of your text must be the same as the language of your voice font.

After filling in the text box and confirming the input mode, select **Yes** to submit your test request and return to the test page. The table now includes an entry that corresponds to your new request and the status column. It can take a few minutes to synthesize speech. When the status column says **Succeeded**, you can download the text input (a `.txt` file) and audio output (a `.wav` file), and audition the latter for quality.

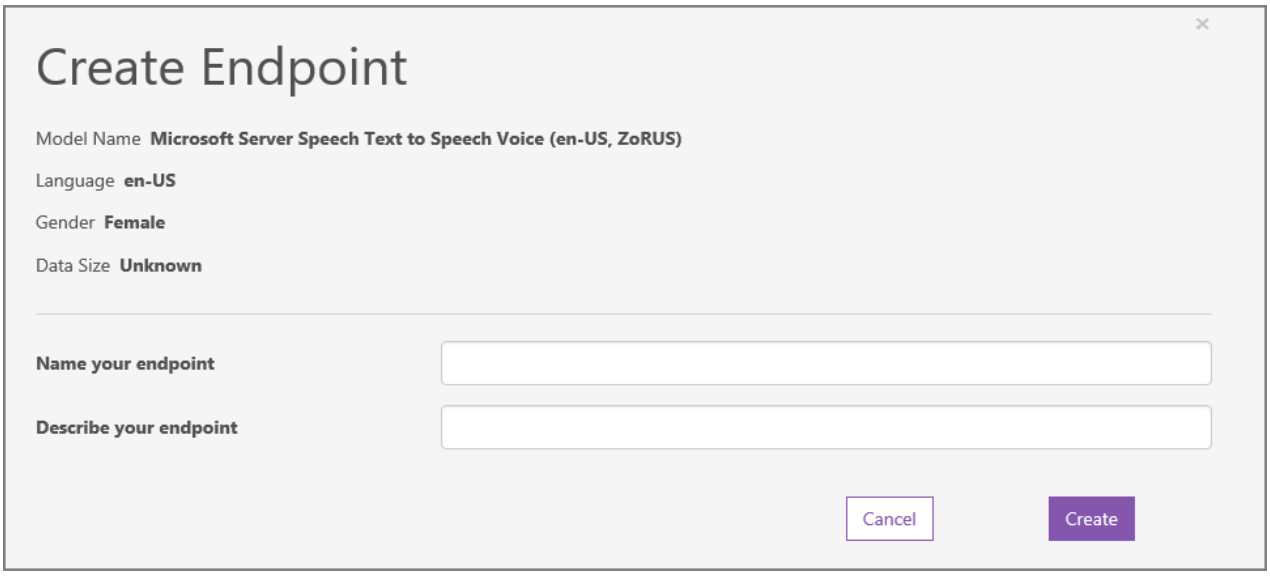
Create and use a custom endpoint

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 used to deploy the font.

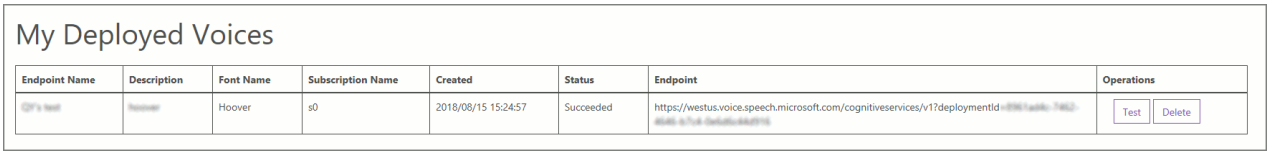
To create a new custom endpoint, choose **Endpoints** from the **Custom Voice** menu at the top of the page. The **My Deployed Voices** page appears, with its table of current custom voice endpoints, if any. The current locale is reflected in the first row of the table. To create a deployment for a different language, change the displayed locale. (It must match the voice you're deploying.)

To create a new endpoint, select the **Deploy voices** button. Enter the name and description of your custom endpoint.

From the **Subscription** menu, choose the subscription that you want to use. Free subscription users can have only one model deployed at a time. Standard subscription users can create up to 20 endpoints, each with its own custom voice.



After selecting the model to be deployed, select **Create**. The **My Deployed Voices** page reappears, now with 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.



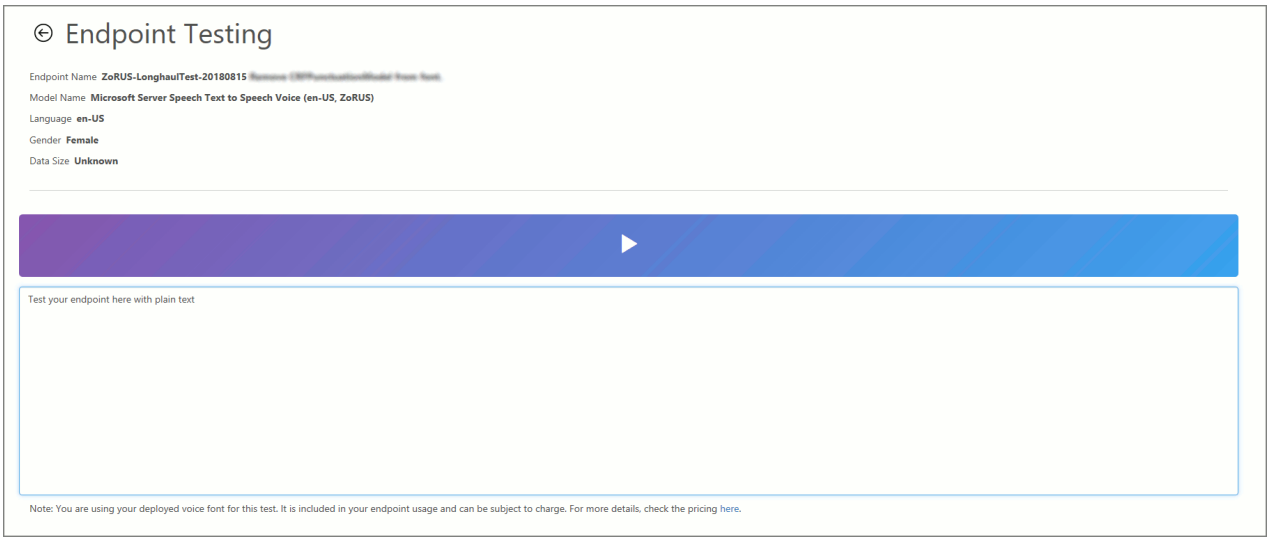
When the deployment status is **Succeeded**, the endpoint of your deployed voice font appears in the **My Deployed Voices** table. You can use this URI directly in an HTTP request.

Online testing of the endpoint is also available via the custom voice portal. To test your endpoint, choose **Endpoints testing** from the **Custom Voice** drop-down menu. The endpoint testing page appears. Choose a

deployed custom voice and enter the text to be spoken (in either plain text or SSML format) in the text box.

> **NOTE**
>
> When using SSML, the `<voice>` tag must specify the name that you gave your custom voice when you created it. If you submit plain text, the custom voice is always used.

To hear the text spoken in your custom voice font, select **Play**.



The custom endpoint is functionally identical to the standard endpoint that's used for text-to-speech requests. See REST API for more information.

Language support

Voice customization is available for US English (en-US), mainland Chinese (zh-CN) and Italian (it-IT).

> **NOTE**
>
> Italian voice training starts with a dataset of 2,000+ utterances. Chinese-English bilingual models are also supported with a dataset of 2,000+ utterances.

Next steps

- Get your Speech trial subscription
- Record your voice samples

Create a custom wake word by using the Speech service

10/19/2018 • 3 minutes to read • Edit Online

Your device is always listening for a wake word (or phrase). For example, "Hey Cortana" is a wake word for the Cortana assistant. When the user says the wake word, the device sends all subsequent audio to the cloud, until the user stops speaking. Customizing your wake word is an effective way to differentiate your device and strengthen your branding.

In this article, you learn how to create a custom wake word for your device.

Choose an effective wake word

Consider the following guidelines when you choose a wake word:

- Your wake word should be an English word or a phrase. It should take no longer than two seconds to say.
- Words of 4 to 7 syllables work best. For example, "Hey, Computer" is a good wake word. Just "Hey" is a poor one.
- Wake words should follow common English pronunciation rules.
- A unique or even a made-up word that follows common English pronunciation rules might reduce false positives. For example, "computerama" might be a good wake word.
- Do not choose a common word. For example, "eat" and "go" are words that people say frequently in ordinary conversation. They might be false triggers for your device.
- Avoid using a wake word that might have alternative pronunciations. Users would have to know the "right" pronunciation to get their device to respond. For example, "509" can be pronounced "five zero nine," "five oh nine," or "five hundred and nine." "R.E.I." can be pronounced "r-e-i" or "ray." "Live" can be pronounced "/līv/" or "/liv/".
- Do not use special characters, symbols, or digits. For example, "Go#" and "20 + cats" would not be good wake words. However, "go sharp" or "twenty plus cats" might work. You can still use the symbols in your branding and use marketing and documentation to reinforce the proper pronunciation.

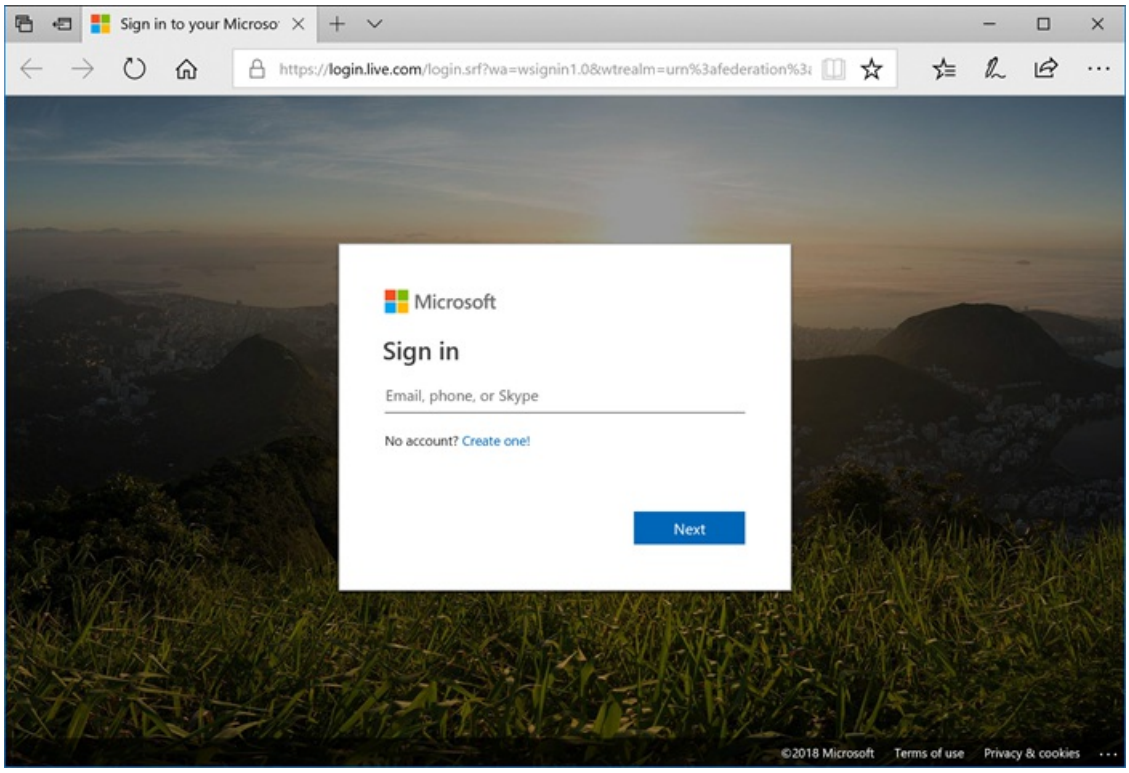
> **NOTE**
>
> If you choose a trademarked word as your wake word, be sure that you own that trademark or that you have permission from the trademark owner to use the word. Microsoft is not liable for any legal issues that might arise from your choice of wake word.

Create your wake word

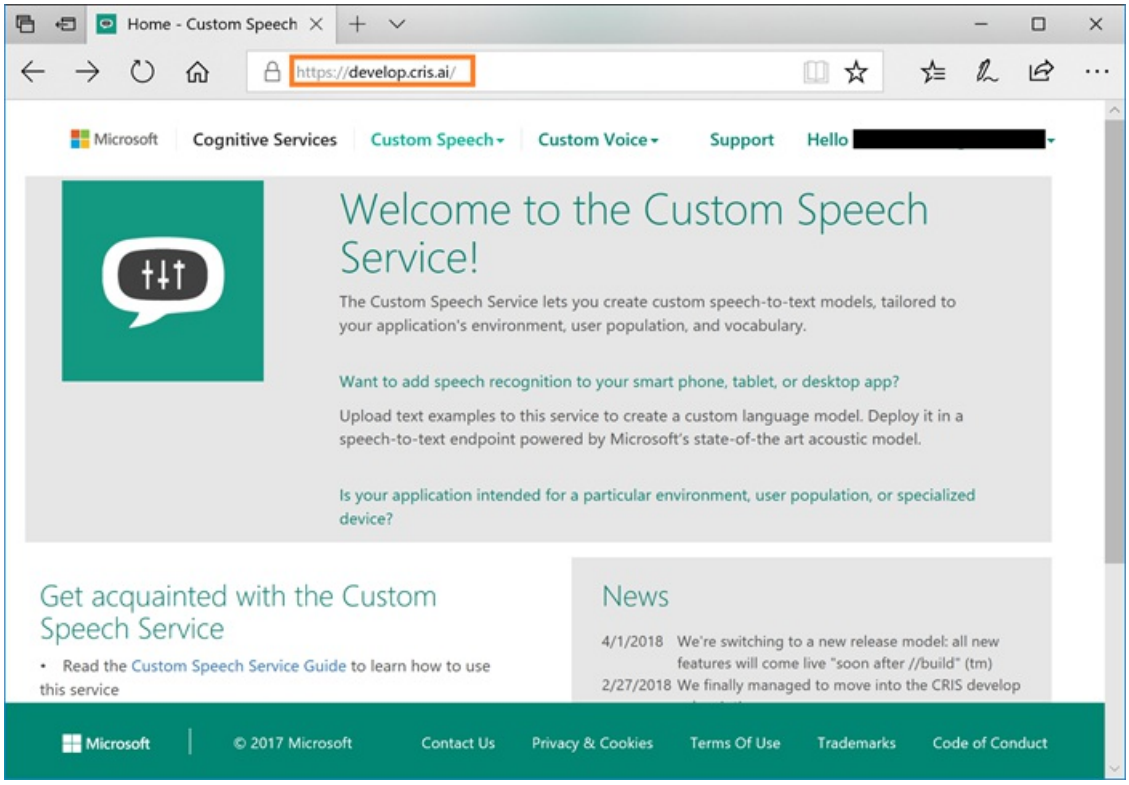
Before you can use a custom wake word with your device, you must create the wake word by using the Microsoft Custom Wake Word Generation service. After you provide a wake word, the service produces a file that you deploy to your development kit to enable your wake word on your device.

1. Go to the Custom Speech service portal.
2. Create a new account with the email address at which you received the invitation for Azure Active

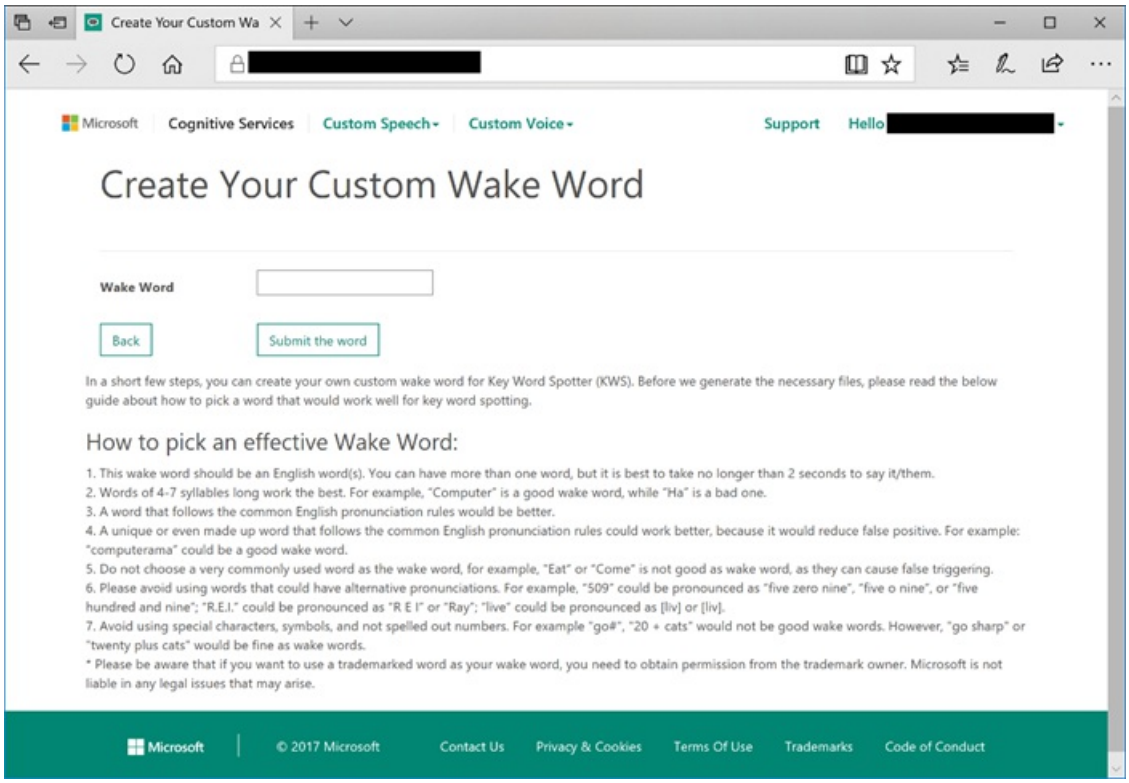
Directory.



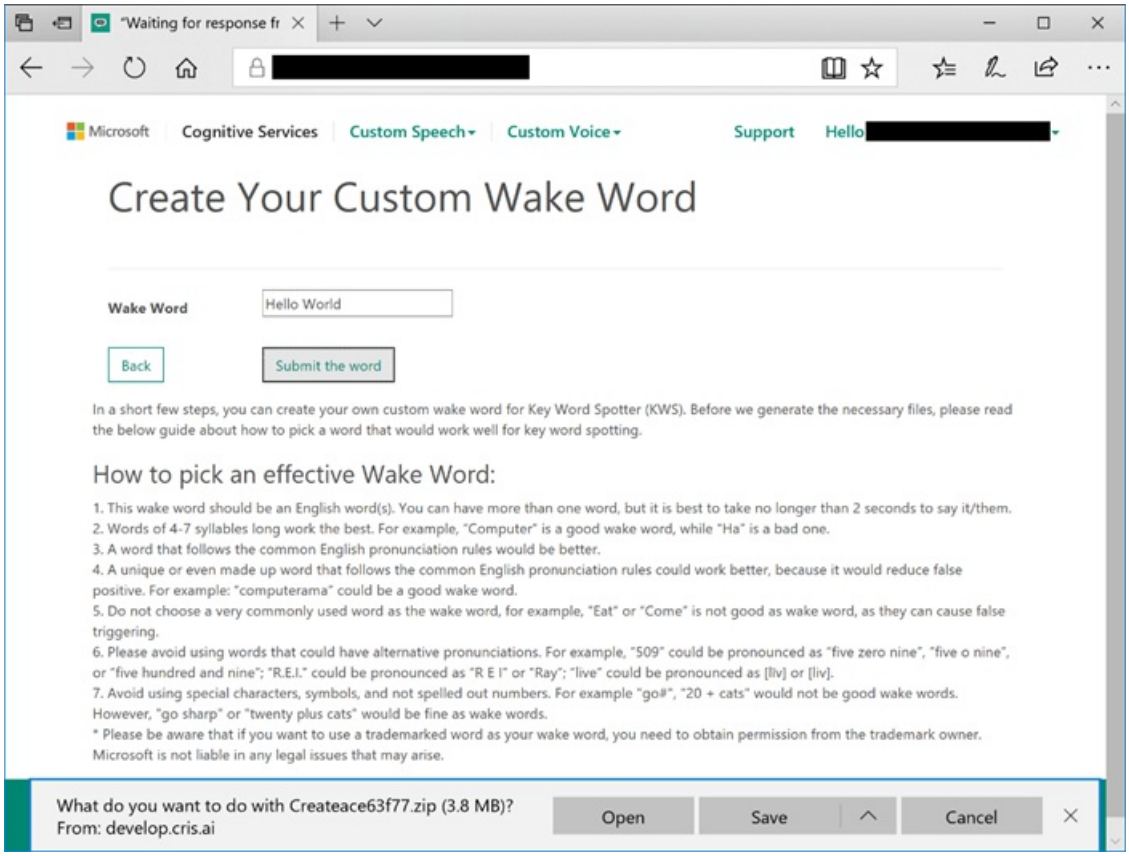
3. The **Custom Wake Word** page is not available to the public, so there is no direct link that takes you there. The Custom Speech feature requires an Azure subscriptions, but the Custom Wake Word feature doesn't. If you got the **No Subscriptions found.** error page, just replace the **"Subscriptions?errorMessage=No%20Subscriptions%20found"** with "**customkws**" in the URL, and hit ENTER. The URL should be one of these: https://westus.cris.ai/customkws, https://eastasia.cris.ai/customkws or https://northeurope.cris.ai/customkws, depending on where your region is.



4. Type in the wake word of your choice, and then select **Submit the word**.



5. It might take a few minutes for the files to be generated. You should see a spinning circle in your browser window. After a moment, an information bar appears, asking you to download a .zip file.



6. Save the .zip file to your computer. You need this file to deploy the custom wake word to the development kit. To deploy the custom wake word, follow the instructions in Get started with the Speech Devices SDK.

7. Select **Sign out.**

Next steps

To get started, get a free Azure account and sign up for the Speech Devices SDK.

Sign up for the Speech Devices SDK

Record voice samples to create a custom voice

10/19/2018 • 17 minutes to read • Edit Online

Creating a high-quality production custom voice from scratch is not a casual undertaking. The central component of a custom voice is a large collection of audio samples of human speech. It's vital that these audio recordings be of high quality. Choose a voice talent who has experience making these kinds of recordings, and have them recorded by a competent recording engineer using professional equipment.

Before you can make these recordings, though, you need a script: the words that will be spoken by your voice talent to create the audio samples. For best results, your script must have good phonetic coverage and sufficient variety to train the custom voice model.

Many small but important details go into creating a professional voice recording. This guide is a roadmap for a process that will help you get good, consistent results.

> **TIP**
> For the highest quality results, consider engaging Microsoft to help develop your custom voice. Microsoft has extensive experience producing high-quality voices for its own products, including Cortana and Office.

Voice recording roles

There are four basic roles in a custom voice recording project:

ROLE	PURPOSE
Voice talent	This person's voice will form the basis of the custom voice.
Recording engineer	Oversees the technical aspects of the recording and operates the recording equipment.
Director	Prepares the script and coaches the voice talent's performance.
Editor	Finalizes the audio files and prepares them for upload to the Custom Voice portal.

An individual may fill more than one role. This guide assumes that you will be primarily filling the director role and hiring both a voice talent and a recording engineer. If you want to make the recordings yourself, this article includes some information about the recording engineer role. The editor role isn't needed until after the session, so can be performed by the director or the recording engineer.

Choose your voice talent

Actors with experience in voiceover or voice character work make good custom voice talent. You can also often find suitable talent among announcers and newsreaders.

Choose voice talent whose natural voice you like. It is possible to create unique "character" voices, but it's much harder for most talent to perform them consistently, and the effort can cause voice strain.

> **TIP**
> Generally, avoid using recognizable voices to create a custom voice—unless, of course, your goal is to produce a celebrity voice. Lesser-known voices are usually less distracting to users.

The single most important factor for choosing voice talent is consistency. Your recordings should all sound like they were made on the same day in the same room. You can approach this ideal through good recording practices and engineering.

Your voice talent is the other half of the equation. He or she 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 his or her pitch variation, emotional affect, and speech mannerisms.

Recording custom voice samples can be more fatiguing than other kinds of voice work. Most voice talent can record for two or three hours a day. Limit sessions to three or four a week, with a day off in-between if possible.

Recordings made for a voice model should be emotionally neutral. That is, a sad utterance should not be read in a sad way. Mood can be added to the synthesized speech later through prosody controls. Work with your voice talent to develop a "persona" that defines the overall sound and emotional tone of the custom voice. In the process, you'll pinpoint what "neutral" sounds like for that persona.

A persona might have, for example, a naturally upbeat personality. So "their" voice might carry a note of optimism even when they speak neutrally. However, such a personality trait should be subtle and consistent. Listen to readings by existing voices to get an idea of what you're aiming for.

> **TIP**
> Usually, you'll want to own the voice recordings you make. Your voice talent should be amenable to a work-for-hire contract for the project.

Create a script

The starting point of any custom voice recording session is the script, which contains the utterances to be spoken by your voice talent. (The term "utterances" encompasses both full sentences and shorter phrases.)

The utterances in your script can come from anywhere: fiction, non-fiction, transcripts of speeches, news reports, and anything else available in printed form. If you want to make sure your voice does well on specific kinds of words (such as medical terminology or programming jargon), you might want to include sentences from scholarly papers or technical documents. For a brief discussion of potential legal issues, see the "Legalities" section. You can also write your own text.

Your utterances don't need to come from the same source, or the same kind of source. They don't even need to have anything to do with each other. However, if you will use set phrases (for example, "You have successfully logged in") in your speech application, make sure to include them in your script. This will give your custom voice a better chance of pronouncing those phrases well. And if you should decide to use a recording in place of synthesized speech, you'll already have it in the same voice.

While consistency is key in choosing voice talent, variety is the hallmark of a good script. Your script should include many different words and sentences with a variety of sentence lengths, structures, and moods. Every sound in the language should be represented multiple times and in numerous contexts (called *phonetic coverage*).

Furthermore, the text should incorporate all the ways that a particular sound can be represented in writing, and place each sound at varying places in the sentences. Both declarative sentences and questions should be included and read with appropriate intonation.

It's difficult to write a script that provides *just enough* data to allow the Custom Speech portal to build a good voice. In practice, the simplest way to make a script that achieves robust phonetic coverage is to include a large number of samples. The standard voices that Microsoft provides were built from tens of thousands of utterances. You should be prepared to record a few to several thousand utterances at minimum to build a production-quality custom voice.

Check the script carefully for errors. If possible, have someone else check it too. When you run through the script with your talent, you'll probably catch a few more mistakes.

Script format

You can write your script in Microsoft Word. The script is for use during the recording session, so you can set it up any way you find easy to work with. Create the text file that's required by the Custom Voice portal separately.

A basic script format contains three columns:

- The number of the utterance, starting at 1. Numbering makes it easy for everyone in the studio to refer to a particular utterance ("let's try number 356 again"). You can use the Word paragraph numbering feature to number the rows of the table automatically.
- A blank column where you'll write the take number or time code of each utterance to help you find it in the finished recording.
- The text of the utterance itself.

Custom Voice Script Session Date: 6/1/65 Voice Talent: A. Lincoln

	Time Code	Utterance
1		Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.
2		Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure.
3		We are met on a great battlefield of that war.
4		We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that that nation might live.

> **NOTE**
>
> Most studios record in short segments known as *takes*. Each take typically contains 10 to 24 utterances. Just noting the take number is sufficient to find an utterance later. If you're recording in a studio that prefers to make longer recordings, you'll want to note the time code instead. The studio will have a prominent time display.

Leave enough space after each row to write notes. Be sure that no utterance is split between pages. Number the pages, and print your script on one side of the paper.

Print three copies of the script: one for the talent, one for the engineer, and one for the director (you). Use a paper clip instead of staples: an experienced voice artist will separate the pages to avoid making noise as the pages are turned.

Legalities

Under copyright law, an actor's reading of copyrighted text might be a performance for which the author of the work should be compensated. This performance will not be recognizable in the final product, the custom voice.

Even so, the legality of using a copyrighted work for this purpose is not well established. Microsoft cannot provide legal advice on this issue; consult your own counsel.

Fortunately, it is possible to avoid these issues entirely. There are many sources of text you can use without permission or license.

TEXT SOURCE	DESCRIPTION
CMU Arctic corpus	About 1100 sentences selected from out-of-copyright works specifically for use in speech synthesis projects. An excellent starting point.
Works no longer under copyright	Typically works published prior to 1923. For English, 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.
Public domain	Works for which copyright has been explicitly disclaimed or that have been dedicated to the public domain. It may not be possible to waive copyright entirely in some jurisdictions.
Permissively-licensed works	Works distributed under a license like Creative Commons or the GNU Free Documentation License (GFDL). Wikipedia uses the GFDL. Some licenses, however, may impose restrictions on performance of the licensed content that may impact the creation of a custom voice model, so read the license carefully.

Recording your script

Record your script at a professional recording studio that specializes in voice work. They'll have a recording booth, the right equipment, and the right people to operate it. It pays not to skimp on recording.

Discuss your project with the studio's recording engineer and listen to his or her advice. The recording should have little or no dynamic range compression (maximum of 4:1). It is critical that the audio have consistent volume and a high signal-to-noise ratio, while being free of unwanted sounds.

Do it yourself

If you want to make the recording yourself, rather than going into a recording studio, here's a short primer. Thanks to the rise of home recording and podcasting, it's easier than ever to find good recording advice and resources online.

Your "recording booth" should be a small room with no noticeable echo or "room tone." It should be as quiet and soundproof as possible. Drapes on the walls can be used to reduce echo and neutralize or "deaden" the sound of the room.

Use a high-quality studio condenser microphone ("mic" for short) intended for recording voice. Sennheiser, AKG, and even newer Zoom mics can yield good results. You can buy a mic, or rent one from a local audio-visual rental firm. Look for one with a USB interface. This type of mic conveniently combines the microphone element, preamp, and analog-to-digital converter into one package, simplifying hookup.

You may also use an analog microphone. Many rental houses offer "vintage" microphones renowned for their voice character. Note that professional analog gear uses balanced XLR connectors, rather than the 1/4-inch plug that's used in consumer equipment. If you go analog, you'll also need a preamp and a computer audio interface

with these connectors.

Install the microphone on a stand or boom, and install a pop filter in front of the microphone to eliminate noise from "plosive" consonants like "p" and "b." Some microphones come with a suspension mount that isolates them from vibrations in the stand, which is helpful.

The voice talent must stay at a consistent distance from the microphone. Use tape on the floor to mark where he or she should stand. If the talent prefers to sit, take special care to monitor mic distance and avoid chair noise.

Use a stand to hold the script. Avoid angling the stand so that it can reflect sound toward the microphone.

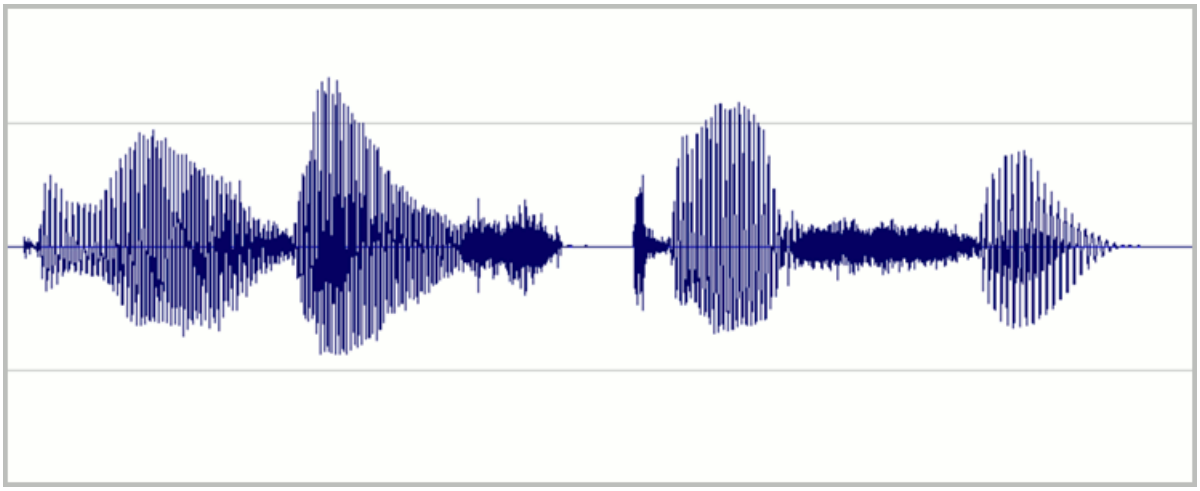
The person operating the recording equipment—the engineer—should be in a separate room from the talent, with some way to talk to the talent in the recording booth (a *talkback circuit)*.

The recording should contain as little noise as possible, with a goal of an 80-db signal-to-noise ratio or better.

Listen closely to a recording of silence in your "booth," figure out where any noise is coming from, and eliminate the cause. Common sources of noise are air vents, fluorescent light ballasts, traffic on nearby roads, and equipment fans (even notebook PCs might have fans). Microphones and cables can pick up electrical noise from nearby AC wiring, usually a hum or buzz. A buzz can also be caused by a *ground loop*, which is caused by having equipment plugged into more than one electrical circuit.

> **TIP**
> In some cases, you might be able to use an equalizer or a noise reduction software plug-in to help remove noise from your recordings, although it is always best to stop it at its source.

Set levels so that most of the available dynamic range of digital recording is used without overdriving. That means set the audio loud, but not so loud that it becomes distorted. An example of the waveform of a good recording is shown in the following image:



Here, most of the range (height) is used, but the highest peaks of the signal do not reach the top or bottom of the window. You can also see that the silence in the recording approximates a thin horizontal line, indicating a low noise floor. This recording has acceptable dynamic range and signal-to-noise ratio.

Record directly into the computer via a high-quality audio interface or a USB port, depending on the mic you're using. For analog, keep the audio chain simple: mic, preamp, audio interface, computer. You can license both Avid Pro Tools and Adobe Audition monthly at a reasonable cost. If your budget is extremely tight, try the free Audacity.

Record at 44.1 kHz 16 bit monophonic (CD quality) or better. Current state-of-the-art is 48 kHz 24-bit, if your equipment supports it. You will down-sample your audio to 16 kHz 16-bit before you submit it to the Custom Voice portal. Still, it pays to have a high-quality original recording in the event edits are needed.

Ideally, have different people serve in the roles of director, engineer, and talent. Don't try to do it all yourself. In a pinch, one person can be both the director and the engineer.

Before the session

To avoid wasting studio time, run through the script with your voice talent before the recording session. While the voice talent becomes familiar with the text, he or she 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 him or her that you want a natural reading with no particular emphasis. Emphasis can be added when speech is synthesized; it should not be a part of the original recording.

Direct the talent to pronounce words distinctly. Every word of the script should be pronounced as written. Sounds should not be omitted or slurred together, as is common in casual speech, *unless they have been written that way in the script*.

WRITTEN TEXT	UNWANTED CASUAL PRONUNCIATION
never going to give you up	never gonna give you up
there are four lights	there're four lights
how's the weather today	how's th' weather today
say hello to my little friend	say hello to my lil' friend

The talent should *not* add distinct pauses between words. The sentence should still flow naturally, even while sounding a little formal. This fine distinction might take practice to get right.

The recording session

Create a reference recording, or *match file*, of a typical utterance at the beginning of the session. Ask the talent to repeat this line every page or so. Each time, compare the new recording to the reference. This practice helps the talent remain consistent in volume, tempo, pitch, and intonation. Meanwhile, the engineer can use the match file as a reference for levels and overall consistency of sound.

The match file is especially important when you resume recording after a break or on another day. You'll want to play it a few times for the talent and have them repeat it each time until they are matching well.

Coach your talent to take a deep breath and pause for a moment before each utterance. Record a couple of seconds of silence between utterances. Words should be pronounced the same way each time they appear, considering context. For example, "record" as a verb is pronounced differently from "record" as a noun.

Record a good five seconds of silence before the first recording to capture the "room tone." This practice helps the Custom Voice portal compensate for any remaining noise in the recordings.

> **TIP**
> All you really need to capture is the voice talent, so you can make a monophonic (single-channel) recording of just their lines. However, if you record in stereo, you can use the second channel to record the chatter in the control room to capture discussion of particular lines or takes. Remove this track from the version that's uploaded to the Custom Voice portal.

Listen closely, using headphones, to the voice talent's performance. You're looking for good but natural diction, correct pronunciation, and a lack of unwanted sounds. Don't hesitate to ask your talent to re-record an utterance that doesn't meet these standards.

> **TIP**
> If you are using a large number of utterances, a single utterance might not have a noticeable effect on the resultant custom voice. It might be more expedient to simply note any utterances with issues, exclude them from your dataset, and see how your custom voice turns out. You can always go back to the studio and record the missed samples later.

Note the take number or time code on your script for each utterance. Ask the engineer to mark each utterance in the recording's metadata or cue sheet as well.

Take regular breaks and provide a beverage to help your voice talent keep his or her voice in good shape.

After the session

Modern recording studios run on computers. At the end of the session, you receive one or more audio files, not a tape. These files will probably be WAV or AIFF format in CD quality (44.1 kHz 16-bit) or better. 48 kHz 24-bit is common and desirable. Higher sampling rates, such as 96 kHz, are generally not needed.

The Custom Voice portal requires each provided utterance to be in its own file. Each audio file delivered by the studio contains multiple utterances. So the primary post-production task is to split up the recordings and prepare them for submission. The recording engineer might have placed markers in the file (or provided a separate cue sheet) to indicate where each utterance starts.

Use your notes to find the exact takes you want, and then use a sound editing utility, such as Avid Pro Tools, Adobe Audition, or the free Audacity, to copy each utterance into a new file.

Leave only about 0.2 seconds of silence at the beginning and end of each clip, except for the first. That file should start with a full five seconds of silence. Do not use an audio editor to "zero out" silent parts of the file. Including the "room tone" will help the Custom Voice algorithms compensate for any residual background noise.

Listen to each file carefully. At this stage, you can edit out small unwanted sounds that you missed during recording, like a slight lip smack before a line, but be careful not to remove any actual speech. If you can't fix a file, remove it from your dataset and note that you have done so.

Convert each file to 16 bits and a sample rate of 16 kHz before saving and, if you recorded the studio chatter, remove the second channel. Save each file in WAV format, naming the files with the utterance number from your script.

Finally, create the *transcript* that associates each WAV file with a text version of the corresponding utterance. Creating custom voice fonts includes details of the required format. You can copy the text directly from your script. Then create a Zip file of the WAV files and the text transcript.

Archive the original recordings in a safe place in case you need them later. Preserve your script and notes, too.

Next steps

You're ready to upload your recordings and create your custom voice.

Create custom voice fonts

Transcription guidelines for using the Speech Service

10/19/2018 • 5 minutes to read • Edit Online

To customize **Speech to Text** or **Text to Speech**, you must provide text along with speech. Each line in the text corresponds to a single utterance. The text should match the speech as closely as possible. The text is called a *transcript*, and you must create it in a specific format.

The Speech Service normalizes the input to keep text consistent.

This article describes both types of normalizations. The guidelines vary slightly for various languages.

US English (en-us)

Text data should be written one utterance per line, in plain text, using only the ASCII character set.

Avoid the use of extended (Latin-1) or Unicode punctuation characters. These characters can be included inadvertently when you prepare the data in a word-processing program or scrape data from webpages. Replace the characters with appropriate ASCII substitutions. For example:

CHARACTERS TO AVOID	SUBSTITUTION
“Hello world” (opening and closing double quotation marks)	"Hello world" (double quotation marks)
John’s day (right single quotation mark)	John's day (apostrophe)
it was good—no, it was great! (em dash)	it was good--no, it was great! (hyphens)

Text normalization rules for English

The Speech Service carries out the following normalization rules:

- Using lowercase letters for all text
- Removing all punctuation except word-internal apostrophes
- Expanding numbers to spoken form, including dollar amounts

Here are some examples:

ORIGINAL TEXT	AFTER NORMALIZATION
"Holy cow!" said Batman.	holy cow said batman
"What?" said Batman's sidekick, Robin.	what said batman's sidekick robin
Go get -em!	go get em
I'm double-jointed	I'm double jointed
104 Elm Street	one oh four Elm street
Tune to 102.7	tune to one oh two seven

ORIGINAL TEXT	AFTER NORMALIZATION
Pi is about 3.14	pi is about three point one four
It costs $3.14	it costs three fourteen

Apply the following normalization to your text transcripts:

- Abbreviations should be written out in words.
- Non-standard numeric strings (such as some date or accounting forms) should be written out in words.
- Words with non-alphabetic characters or mixed alphanumeric characters should be transcribed as pronounced.
- Leave abbreviations that are pronounced as words untouched (for example, "radar," "laser," "RAM," or "NATO").
- Write abbreviations that are pronounced as separate letters, with letters separated by spaces (for example, "IBM," "CPU," "FBI," "TBD," or "NaN").

Here are some examples:

ORIGINAL TEXT	AFTER NORMALIZATION
14 NE 3rd Dr.	fourteen northeast third drive
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

Chinese (zh-cn)

Text data that's uploaded to the Custom Speech Service should use UTF-8 encoding with a byte-order marker. The file should be written one utterance per line.

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 webpages. Replace them with appropriate full-width substitutions. For example:

CHARACTERS TO AVOID	SUBSTITUTION
"你好" (opening and closing double quotation marks)	"你好" (double quotation marks)

CHARACTERS TO AVOID	SUBSTITUTION
需要什么帮助? (question mark)	需要什么帮助？

Text normalization rules for Chinese

The Speech Service carries out the following normalization rules:

- Removing all punctuation
- Expanding numbers to spoken form
- Converting full-width letters to half-width letters
- Using uppercase letters for all English words

Here are some examples:

ORIGINAL 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

Before you import your text, apply the following normalization to it:

- Abbreviations should be written out in words (as in spoken form).
- Write out numeric strings in spoken form.

Here are some examples:

ORIGINAL TEXT	AFTER NORMALIZATION
我今年21	我今年二十一
3号楼504	三号 楼 五 零 四

Other languages

Text data uploaded to the **Speech to Text** service must use UTF-8 encoding with a byte-order marker. The file should be written one utterance per line.

> **NOTE**
> The following examples use German. However, the guidelines apply to all languages that are not US English or Chinese.

Text normalization rules for German

The Speech Service carries out the following normalization rules:

- Using lowercase letters for all text
- Removing all punctuation, including various types of quotation marks ("test", 'test', "test„ and «test» are OK)
- Discarding rows with any special character from the set ¢ ¤ ¥ ¦ § © ª ¬ ® ° ± ² µ × ÿ Ø ¬ ¬
- Expanding numbers to word form, including dollar or Euro amounts
- Accepting umlauts only for a, o, and u; others will be replaced by "th" or be discarded

Here are some examples:

ORIGINAL TEXT	AFTER NORMALIZATION
Frankfurter Ring	frankfurter ring
¡Eine Frage!	eine frage
wir, haben	wir haben

Before you import your text, apply the following normalization to it:

- Decimal points should be "," and not ".".
- Time separators between hours and minutes should be ":" and not "." (for example, 12:00 Uhr).
- Abbreviations such as "ca." aren't replaced. We recommend that you use the full form.
- The four main mathematical operators (+, -, *, and /) are removed. We recommend replacing them with their literal form: "plus," "minus," "mal," and "geteilt."
- The same rule applies to comparison operators (=, <, and >). We recommend replacing them with "gleich," "kleiner als," and "grösser als."
- Use fractions, such as 3/4, in word form (for example, "drei viertel" instead of ¾).
- Replace the € symbol with the word form "Euro."

Here are some examples:

ORIGINAL TEXT	AFTER USER'S NORMALIZATION	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

Next steps

- Get your Speech Service trial subscription
- Recognize speech in C#

About the Speech SDK audio input stream API

10/19/2018 • 2 minutes to read • Edit Online

The Speech SDK's **Audio Input Stream** API provides a way to stream audio streams into the recognizers instead of using either the microphone or the input file APIs.

The following steps are required when using audio input streams:

- Identify the format of the audio stream. The format must be supported by the Speech SDK and the Speech service. Currently, only the following configuration is supported:

  Audio samples in PCM format, one channel, 16000 samples per second, 32000 bytes per second, two block align (16 bit including padding for a sample), 16 bits per sample.

  The corresponding code in the SDK to create the audio format looks like this:

  ```
  byte channels = 1;
  byte bitsPerSample = 16;
  int samplesPerSecond = 16000;
  var audioFormat = AudioStreamFormat.GetWaveFormatPCM(samplesPerSecond, bitsPerSample, channels);
  ```

- Make sure your code can provide the RAW audio data according to these specifications. If your audio source data doesn't match the supported formats, the audio must be transcoded into the required format.

- Create your own audio input stream class derived from `PullAudioInputStreamCallback`. Implement the `Read()` and `Close()` members. The exact function signature is language-dependent, but the code will look similar to this code sample:

  ```
  public class ContosoAudioStream : PullAudioInputStreamCallback {
     ContosoConfig config;

     public ContosoAudioStream(const ContosoConfig& config) {
         this.config = config;
     }

     public size_t Read(byte *buffer, size_t size) {
         // returns audio data to the caller.
         // e.g. return read(config.YYY, buffer, size);
     }

     public void Close() {
         // close and cleanup resources.
     }
  };
  ```

- Create an audio configuration based on your audio format and input stream. Pass in both your regular speech configuration and the audio input configuration when you create your recognizer. For example:

```
var audioConfig = AudioConfig.FromStreamInput(new ContosoAudioStream(config), audioFormat);

var speechConfig = SpeechConfig.FromSubscription(...);
var recognizer = new SpeechRecognizer(speechConfig, audioConfig);

// run stream through recognizer
var result = await recognizer.RecognizeOnceAsync();

var text = result.GetText();
```

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

Migrate from Bing Speech to the Speech Service

10/31/2018 • 4 minutes to read • Edit Online

Use this article to migrate your applications from the Bing Speech API to the Speech Service.

This article outlines the differences between the Bing Speech APIs and the Speech Service, and suggests strategies for migrating your applications. Your Bing Speech API subscription key won't be accepted by the Speech Service; you'll need a new Speech Service subscription.

A single Speech Service subscription key grants access to the following features. Each is metered separately, so you're charged only for the features you use.

- Speech-to-text
- Custom speech-to-text
- Text-to-speech
- Custom text-to-speech voices
- Speech translation (does not include Text translation)

The Speech SDK is a functional replacement for the Bing Speech client libraries, but uses a different API.

Comparison of features

The Speech Service is largely similar to Bing Speech, with the following differences.

FEATURE	BING SPEECH	SPEECH SERVICE	DETAILS
C++ SDK	□	✔□	Speech Service supports Windows and Linux.
Java SDK	✔□	✔□	Speech Service supports Android and Speech Devices.
C# SDK	✔□	✔□	Speech Service supports Windows 10, Universal Windows Platform (UWP), and .NET Standard 2.0.
Continuous speech recognition	10 minutes	Unlimited (with SDK)	Both Bing Speech and Speech Service WebSockets protocols support up to 10 minutes per call. However, the Speech SDK automatically reconnects on timeout or disconnect.
Partial or interim results	✔□	✔□	With WebSockets protocol or SDK.
Custom speech models	✔□	✔□	Bing Speech requires a separate Custom Speech subscription.

FEATURE	BING SPEECH	SPEECH SERVICE	DETAILS
Custom voice fonts	✔	✔	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 Speech Service.
Endpoint locality	Global	Regional	Regional endpoints improve latency.
REST APIs	✔	✔	Speech Service REST API is compatible with Bing Speech (different endpoint). REST APIs support text-to-speech and limited speech-to-text functionality.
WebSockets protocols	✔	✔	Speech Service WebSockets API is compatible with Bing Speech (different endpoint). Migrate to the Speech SDK if possible, to simplify your code.
Service-to-service API calls	✔		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.

The Speech Service WebSockets protocols are also compatible with those used by Bing Speech. We recommend that for new development, you use the Speech Service SDK rather than WebSockets. It's a good idea to migrate existing code to the SDK as well. However, as with the REST APIs, existing code that uses Bing Speech via WebSockets requires only a change in endpoint and an updated key.

If you're using a Bing Speech client library for a specific programming language, migrating to the Speech SDK requires changes to your application, because the API is different. The Speech SDK can make your code simpler, while also giving you access to new features.

Currently, the Speech SDK supports C# (Windows 10, UWP, .NET Standard), Java (Android and custom devices), Objective C (iOS), C++ (Windows and Linux), and JavaScript. APIs on all platforms are similar, easing multi-platform development.

The Speech Service doesn't currently offer a global endpoint. Determine if your application functions efficiently when it uses a single regional endpoint for all of its traffic. If not, use geolocation to determine the most efficient endpoint. You need a separate Speech Service subscription in each region you use.

If your application uses long-lived connections and can't use an available SDK, you can use a WebSockets connection. Manage the 10-minute timeout limit by reconnecting at the appropriate times.

To get started with the Speech SDK:

1. Download the Speech SDK.
2. Work through the Speech Service quickstart guides and tutorials. Also look at the code samples to get experience with the new APIs.
3. Update your application to use the Speech Service and APIs.

Support

Bing Speech customers should contact customer support by opening a support ticket. You can also contact us if your support need requires a technical support plan.

For Speech Service, SDK, and API support, visit the Speech Service support page.

Next steps

- Try out Speech Service for free
- Quickstart: Recognize speech in a UWP app using the Speech SDK

See also

- Speech Service release notes
- What is the Speech Service
- Speech Service and SDK documentation

Migrate from the Custom Speech Service to the Speech Service

10/31/2018 • 2 minutes to read • Edit Online

Use this article to migrate your applications from the Custom Speech Service to the Speech Service.

The Custom Speech Service is now part of the Speech Service. Switch to the Speech Service to benefit from the latest quality and feature updates.

Migration for new customers

The pricing model is simpler, using an hour-based pricing model for the Speech Service.

1. Create an Azure resource in each region where your application is available. The Azure resource name is **Speech**. You can use a single Azure resource for the following services in the same region, instead of creating separate resources:
   - Speech-to-text
   - Custom speech-to-text
   - Text-to-speech
   - Speech translation
2. Download the Speech SDK.
3. Follow the quickstart guides and SDK samples to use the correct APIs. If you use the REST APIs, you also need to use the correct endpoints and resource keys.
4. Update the client application to use the Speech Service and APIs.

> **NOTE**
> - If you enabled speech in Language Understanding (LUIS), a single LUIS resource in the same region will work for LUIS as well as all the speech services. For more information, see Recognize intents from speech.
> - Text-to-text translation is not part of the Speech Service. This functionality requires its own Azure resource subscription.

Migration for existing customers

Migrate your existing resource keys to the Speech Service on the Speech Service portal. Use the following steps:

> **NOTE**
> Resource keys can only be migrated within the same region.

1. Sign in to the cris.ai portal, and select the subscription in the top right menu.
2. Select **Migrate selected subscription**.
3. Enter the subscription key in the text box, and select **Migrate**.

Next steps

- Try out Speech Service for free.
- Learn speech to text concepts.

See also

- What is the Speech Service
- Speech Service and SDK documentation

Migrate from the Translator Speech API to the Speech Service

10/31/2018 • 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 will need to start 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 currently 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 API 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 (currently in preview) 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 currently offer a global endpoint. Determine if your application functions efficiently when it uses a single regional endpoint for all of its traffic. If not, use geolocation to determine the most efficient endpoint.
- If your application uses long-lived connections and can't use the available SDKs, you can use a WebSockets connection. Manage the 10-minute timeout limit by reconnecting at the appropriate times.
- If your application uses the Translator Text API and Translator Speech API to enable custom translation models, you can add Category IDs directly by using the Speech Service.
- Unlike the Translator Speech API, the Speech Service can complete translations into multiple languages in a single request.

Next steps

- Try out Speech Service for free

- Quickstart: Recognize speech in a UWP app using the Speech SDK

See also

- What is the Speech Service
- Speech Service and SDK documentation

About the Speech to Text API

10/19/2018 • 2 minutes to read • Edit Online

The **Speech to Text** API *transcribes* audio streams into text that your application can display to the user or act upon as command input. The APIs can be used either with an SDK client library (for supported platforms and languages) or a REST API.

The **Speech to Text** API offers the following features:

- Advanced speech recognition technology from Microsoft—the same used by Cortana, Office, and other Microsoft products.
- Real-time continuous recognition. **Speech to Text** allows users to transcribe audio into text in real time. It also supports receiving intermediate results of the words that have been recognized so far. The service automatically recognizes the end of speech. Users can also choose additional formatting options, including capitalization and punctuation, profanity masking, and inverse text normalization.
- Results returned in both Lexical and Display forms (for Lexical results, see DetailedSpeechRecognitionResult in the examples or API).
- Support for many spoken languages and dialects. For the full list of supported languages in each recognition mode, see Supported languages.
- Customized language and acoustic models, so you can tailor your application to your users' specialized domain vocabulary, speaking environment and way of speaking.
- Natural-language understanding. Through integration with Language Understanding (LUIS), you can derive intents and entities from speech. Users don't have to know your app's vocabulary, but can describe what they want in their own words.
- Confidence score is returned back from the service if you specify a detailed output on the speech configuration object (SpeechConfig.OutputFormat property). Then you can use either Best() method on the result or get it the score directly from JSON returned from the service (something like result.Properties.GetProperty(PropertyId.SpeechServiceResponse_JsonResult)).

API capabilities

Some of the capabilities of the **Speech to Text** API, especially around customization, are available via REST. The following table summarizes the capabilities of each method of accessing the API. For a full list of capabilities and API details, see Swagger reference.

USE CASE	REST	SDKS
Transcribe a short utterance, such as a command (length < 15 s); no interim results	Yes	Yes
Transcribe a longer utterance (> 15 s)	No	Yes
Transcribe streaming audio with optional interim results	No	Yes

USE CASE	REST	SDKS
Understand speaker intents via LUIS	No*	Yes
Create Accuracy Tests	Yes	No
Upload datasets for model adaptation	Yes	No
Create & manage speech models	Yes	No
Create & manage model deployments	Yes	No
Manage Subscriptions	Yes	No
Create & manage model deployments	Yes	No
Create & manage model deployments	Yes	No

> **NOTE**
> The REST API implements throttling that limits the API requests to 25 per 5 second. Message headers will inform of the limits

** LUIS intents and entities can be derived using a separate LUIS subscription. With this subscription, the SDK can call LUIS for you and provide entity and intent results as well as speech transcriptions. With the REST API, you can call LUIS yourself to derive intents and entities with your LUIS subscription.*

Next steps

- Get your Speech trial subscription
- Quickstart: recognize speech in C#
- See how to recognize intents from speech in C#

About the Text to Speech API

10/19/2018 • 2 minutes to read • Edit Online

The **Text to Speech** (TTS) API of the Speech service converts input text into natural-sounding speech (also called *speech synthesis*).

To generate speech, your application sends HTTP POST requests to the Speech service. There, text is synthesized into human-sounding speech and returned as an audio file. A variety of voices and languages are supported.

Scenarios in which speech synthesis is being adopted include:

- *Improving accessibility:* **Text to Speech** technology enables content owners and publishers to respond to the different ways people interact with their content. People with visual impairment or reading difficulties appreciate being able to consume content aurally. Voice output also makes it easier for people to enjoy textual content, such as newspapers or blogs, on mobile devices while commuting or exercising.
- *Responding in multitasking scenarios:* **Text to Speech** enables people to absorb important information quickly and comfortably while driving or otherwise outside a convenient reading environment. Navigation is a common application in this area.
- *Enhancing learning with multiple modes:* Different people learn best in different ways. Online learning experts have shown that providing voice and text together can help make information easier to learn and retain.
- *Delivering intuitive bots or assistants:* The ability to talk can be an integral part of an intelligent chat bot or a virtual assistant. More and more companies are developing chat bots to provide engaging customer service experiences for their customers. Voice adds another dimension by allowing the bot's responses to be received aurally (for example, by phone).

Voice support

The Microsoft **Text-to-Speech** service offers more than 75 voices in more than 45 languages and locales. To use these standard "voice fonts", you only need to specify the voice name with a few other parameters when you call the service's REST API. For the details of the voices supported, see Supported languages.

If you want a unique voice for your application, you can create custom voice fonts from your own speech samples.

API capabilities

A lot of the capabilities of the **Text to Speech** API, especially around customization, are available via REST. The following table summarizes the capabilities of each method of accessing the API. For a full list of capabilities and API details, see Swagger reference.

USE CASE	REST	SDKS
Upload datasets for voice adaptation	Yes	No
Create & manage voice font models	Yes	No
Create & manage voice font deployments	Yes	No

USE CASE	REST	SDKS
Create & manage voice font tests	Yes	No
Manage Subscriptions	Yes	No

> **NOTE**
> The API implements throttling that limits the API requests to 25 per 5 seconds. Message headers will inform of the limits.

Next steps

- Get your Speech trial subscription
- See how to synthesize speech via the REST API

About the Speech Translation API

10/19/2018 • 2 minutes to read • Edit Online

The Speech Service API lets you add end-to-end, real-time, multi-language translation of speech to your applications, tools, and devices. The same API can be used for both speech-to-speech and speech-to-text translation.

With the Translator Speech API, client applications stream speech audio to the service and receive back a stream of results. These results include the recognized text in the source language and its translation in the target language. Interim translations can be provided until an utterance is complete, at which time a final translation is provided.

Optionally, a synthesized audio version of the final translation can be prepared, enabling true speech-to-speech translation.

The Speech Translation API uses a WebSockets protocol to provide a full-duplex communication channel between the client and the server. But you don't need to deal with WebSockets; the Speech SDK handles that for you.

The Speech Translation API employs the same technologies that power various Microsoft products and services. This service is already used by thousands of businesses worldwide in their applications and workflows.

About the technology

Underlying Microsoft's translation engine are two different approaches: statistical machine translation (SMT) and neural machine translation (NMT). The latter, an artificial intelligence approach employing neural networks, is the more modern approach to machine translation. NMT provides better translations — not just more accurate, but also more fluent and natural. The key reason for this fluidity is that NMT uses the full context of a sentence to translate words.

Today, Microsoft has migrated to NMT for the most popular languages, employing SMT only for less-frequently-used languages. All languages available for speech-to-speech translation are powered by NMT. Speech-to-text translation may use SMT or NMT depending on the language pair. If the target language is supported by NMT, the full translation is NMT-powered. If the target language isn't supported by NMT, the translation is a hybrid of NMT and SMT, using English as a "pivot" between the two languages.

The differences between models are internal to the translation engine. End users notice only the improved translation quality, especially for Chinese, Japanese, and Arabic.

> **NOTE**
>
> Interested in learning more about the technology behind Microsoft's translation engine? See Machine Translation.

Next steps

- Get your Speech trial subscription
- See how to translate speech in C#
- See how to translate speech in C++
- See how to translate speech in Java

Language and region support for Speech Service API

10/19/2018 • 7 minutes to read • Edit Online

Different languages are supported for different Speech service functions. The following tables summarize language support.

Speech to Text

The Microsoft speech recognition API supports the following languages. Different levels of customization are available for each language.

CODE	LANGUAGE	ACOUSTIC ADAPTATION	LANGUAGE ADAPTATION	PRONUNCIATION ADAPTATION
ar-EG	Arabic (Egypt), modern standard	No	Yes	No
ca-ES	Catalan (Spain)	No	No	No
da-DK	Danish (Denmark)	No	No	No
de-DE	German (Germany)	Yes	Yes	No
en-AU	English (Australia)	No	Yes	Yes
en-CA	English (Canada)	No	Yes	Yes
en-GB	English (United Kingdom)	No	Yes	Yes
en-IN	English (India)	Yes	Yes	Yes
en-NZ	English (New Zealand)	No	Yes	Yes
en-US	English (United States)	Yes	Yes	Yes
es-ES	Spanish (Spain)	No	Yes	No
es-MX	Spanish (Mexico)	No	Yes	No
fi-FI	Finnish (Finland)	No	No	No
fr-CA	French (Canada)	No	Yes	No
fr-FR	French (France)	Yes	Yes	No
hi-IN	Hindi (India)	No	Yes	No
it-IT	Italian (Italy)	Yes	Yes	No

CODE	LANGUAGE	ACOUSTIC ADAPTATION	LANGUAGE ADAPTATION	PRONUNCIATION ADAPTATION
ja-JP	Japanese (Japan)	No	Yes	No
ko-KR	Korean (Korea)	No	Yes	No
nb-NO	Norwegian (Bokmål) (Norway)	No	No	No
nl-NL	Dutch (Netherlands)	No	Yes	No
pl-PL	Polish (Poland)	No	No	No
pt-BR	Portuguese (Brazil)	No	Yes	No
pt-PT	Portuguese (Portugal)	No	Yes	No
ru-RU	Russian (Russia)	Yes	Yes	No
sv-SE	Swedish (Sweden)	No	No	No
zh-CN	Chinese (Mandarin, simplified)	Yes	Yes	No
zh-HK	Chinese (Mandarin, Traditional)	No	Yes	No
zh-TW	Chinese (Taiwanese Mandarin)	No	Yes	No
th-TH	Thai (Thailand)	No	No	No

Text to Speech

The speech synthesis API offers the following voices, each of which supports a specific language and dialect, identified by locale.

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
ar-EG*	Arabic (Egypt)	Female	"Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)"
ar-SA	Arabic (Saudi Arabia)	Male	"Microsoft Server Speech Text to Speech Voice (ar-SA, Naayf)"
bg-BG	Bulgarian	Male	"Microsoft Server Speech Text to Speech Voice (bg-BG, Ivan)"

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
ca-ES	Catalan (Spain)	Female	"Microsoft Server Speech Text to Speech Voice (ca-ES, HerenaRUS)"
cs-CZ	Czech	Male	"Microsoft Server Speech Text to Speech Voice (cs-CZ, Jakub)"
cs-CZ	Czech	Male	"Microsoft Server Speech Text to Speech Voice (cs-CZ, Vit)"
da-DK	Danish	Female	"Microsoft Server Speech Text to Speech Voice (da-DK, HelleRUS)"
de-AT	German (Austria)	Male	"Microsoft Server Speech Text to Speech Voice (de-AT, Michael)"
de-CH	German (Switzerland)	Male	"Microsoft Server Speech Text to Speech Voice (de-CH, Karsten)"
de-DE	German (Germany)	Female	"Microsoft Server Speech Text to Speech Voice (de-DE, Hedda)"
		Female	"Microsoft Server Speech Text to Speech Voice (de-DE, HeddaRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (de-DE, Stefan, Apollo)"
el-GR	Greek	Male	"Microsoft Server Speech Text to Speech Voice (el-GR, Stefanos)"
en-AU	English (Australia)	Female	"Microsoft Server Speech Text to Speech Voice (en-AU, Catherine)"
		Female	"Microsoft Server Speech Text to Speech Voice (en-AU, HayleyRUS)"
en-CA	English (Canada)	Female	"Microsoft Server Speech Text to Speech Voice (en-CA, Linda)"
		Female	"Microsoft Server Speech Text to Speech Voice (en-CA, HeatherRUS)"

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
en-GB	English (UK)	Female	"Microsoft Server Speech Text to Speech Voice (en-GB, Susan, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (en-GB, HazelRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (en-GB, George, Apollo)"
en-IE	English (Ireland)	Male	"Microsoft Server Speech Text to Speech Voice (en-IE, Sean)"
en-IE	English (Ireland)	Male	"Microsoft Server Speech Text to Speech Voice (en-IE, Shaun)"
en-IN	English (India)	Female	"Microsoft Server Speech Text to Speech Voice (en-IN, Heera, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (en-IN, PriyaRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (en-IN, Ravi, Apollo)"
en-US	English (US)	Female	"Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)"
		Female	"Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (en-US, BenjaminRUS)"
		Female	"Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (en-US, Guy24kRUS)"
es-ES	Spanish (Spain)	Female	"Microsoft Server Speech Text to Speech Voice (es-ES, Laura, Apollo)"

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
		Female	"Microsoft Server Speech Text to Speech Voice (es-ES, HelenaRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (es-ES, Pablo, Apollo)"
es-MX	Spanish (Mexico)	Female	"Microsoft Server Speech Text to Speech Voice (es-MX, HildaRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (es-MX, Raul, Apollo)"
fi-FI	Finnish	Female	"Microsoft Server Speech Text to Speech Voice (fi-FI, HeidiRUS)"
fr-CA	French (Canada)	Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, Caroline)"
		Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, HarmonieRUS)"
fr-CH	French (Switzerland)	Male	"Microsoft Server Speech Text to Speech Voice (fr-CH, Guillaume)"
fr-FR	French (France)	Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, Julie, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, HortenseRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (fr-FR, Paul, Apollo)"
he-IL	Hebrew (Israel)	Male	"Microsoft Server Speech Text to Speech Voice (he-IL, Asaf)"
hi-IN	Hindi (India)	Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana)"

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
		Male	"Microsoft Server Speech Text to Speech Voice (hi-IN, Hemant)"
hr-HR	Croatian	Male	"Microsoft Server Speech Text to Speech Voice (hr-HR, Matej)"
hu-HU	Hungarian	Male	"Microsoft Server Speech Text to Speech Voice (hu-HU, Szabolcs)"
id-ID	Indonesian	Male	"Microsoft Server Speech Text to Speech Voice (id-ID, Andika)"
it-IT	Italian	Male	"Microsoft Server Speech Text to Speech Voice (it-IT, Cosimo, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (it-IT, LuciaRUS)"
ja-JP	Japanese	Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ayumi, Apollo)"
		Male	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ichiro, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, HarukaRUS)"
ko-KR	Korean	Female	"Microsoft Server Speech Text to Speech Voice (ko-KR, HeamiRUS)"
ms-MY	Malay	Male	"Microsoft Server Speech Text to Speech Voice (ms-MY, Rizwan)"
nb-NO	Norwegian	Female	"Microsoft Server Speech Text to Speech Voice (nb-NO, HuldaRUS)"
nl-NL	Dutch	Female	"Microsoft Server Speech Text to Speech Voice (nl-NL, HannaRUS)"
pl-PL	Polish	Female	"Microsoft Server Speech Text to Speech Voice (pl-PL, PaulinaRUS)"

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
pt-BR	Portuguese (Brazil)	Female	"Microsoft Server Speech Text to Speech Voice (pt-BR, HeloisaRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (pt-BR, Daniel, Apollo)"
pt-PT	Portuguese (Portugal)	Female	"Microsoft Server Speech Text to Speech Voice (pt-PT, HeliaRUS)"
ro-RO	Romanian	Male	"Microsoft Server Speech Text to Speech Voice (ro-RO, Andrei)"
ru-RU	Russian	Female	"Microsoft Server Speech Text to Speech Voice (ru-RU, Irina, Apollo)"
		Male	"Microsoft Server Speech Text to Speech Voice (ru-RU, Pavel, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (ru-RU, EkaterinaRUS)"
sk-SK	Slovak	Male	"Microsoft Server Speech Text to Speech Voice (sk-SK, Filip)"
sl-SI	Slovenian	Male	"Microsoft Server Speech Text to Speech Voice (sl-SI, Lado)"
sv-SE	Swedish	Female	"Microsoft Server Speech Text to Speech Voice (sv-SE, HedvigRUS)"
ta-IN	Tamil (India)	Male	"Microsoft Server Speech Text to Speech Voice (ta-IN, Valluvar)"
te-IN	Telugu (India)	Female	"Microsoft Server Speech Text to Speech Voice (te-IN, Chitra)"
th-TH	Thai	Male	"Microsoft Server Speech Text to Speech Voice (th-TH, Pattara)"
tr-TR	Turkish	Female	"Microsoft Server Speech Text to Speech Voice (tr-TR, SedaRUS)"

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
vi-VN	Vietnamese	Male	"Microsoft Server Speech Text to Speech Voice (vi-VN, An)"
zh-CN	Chinese (Mainland)	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, HuihuiRUS)"
		Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, Yaoyao, Apollo)"
		Male	"Microsoft Server Speech Text to Speech Voice (zh-CN, Kangkang, Apollo)"
zh-HK	Chinese (Hong Kong)	Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, Tracy, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, TracyRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (zh-HK, Danny, Apollo)"
zh-TW	Chinese (Taiwan)	Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, Yating, Apollo)"
		Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, HanHanRUS)"
		Male	"Microsoft Server Speech Text to Speech Voice (zh-TW, Zhiwei, Apollo)"

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

Customization

Voice customization is available for US English (en-US), mainland Chinese (zh-CN), and Italian (it-IT).

> **NOTE**
> Italian voice training starts with a data set of 2,000+ utterances. Chinese-English bilingual models also are supported with an initial data set of 2,000+ utterances.

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 following Speech language table. The available target languages

depend on whether the translation target is speech or text.

Speech languages

SPEECH LANGUAGE	LANGUAGE CODE
Arabic (Modern Standard)	`ar`
Chinese (Mandarin)	`zh`
English	`en`
French	`fr`
German	`de`
Italian	`it`
Japanese	`jp`
Portuguese (Brazilian)	`pt`
Russian	`ru`
Spanish	`es`

Text languages

TEXT LANGUAGE	LANGUAGE CODE
Afrikaans	`af`
Arabic	`ar`
Bangla	`bn`
Bosnian (Latin)	`bs`
Bulgarian	`bg`
Cantonese (Traditional)	`yue`
Catalan	`ca`
Chinese Simplified	`zh-Hans`
Chinese Traditional	`zh-Hant`
Croatian	`hr`
Czech	`cs`

TEXT LANGUAGE	LANGUAGE CODE
Danish	`da`
Dutch	`nl`
English	`en`
Estonian	`et`
Fijian	`fj`
Filipino	`fil`
Finnish	`fi`
French	`fr`
German	`de`
Greek	`el`
Haitian Creole	`ht`
Hebrew	`he`
Hindi	`hi`
Hmong Daw	`mww`
Hungarian	`hu`
Indonesian	`id`
Italian	`it`
Japanese	`ja`
Kiswahili	`sw`
Klingon	`tlh`
Klingon (plqaD)	`tlh-Qaak`
Korean	`ko`
Latvian	`lv`
Lithuanian	`lt`

TEXT LANGUAGE	LANGUAGE CODE
Malagasy	`mg`
Malay	`ms`
Maltese	`mt`
Norwegian	`nb`
Persian	`fa`
Polish	`pl`
Portuguese	`pt`
Queretaro Otomi	`otq`
Romanian	`ro`
Russian	`ru`
Samoan	`sm`
Serbian (Cyrillic)	`sr-Cyrl`
Serbian (Latin)	`sr-Latn`
Slovak	`sk`
Slovenian	`sl`
Spanish	`es`
Swedish	`sv`
Tahitian	`ty`
Tamil	`ta`
Thai	`th`
Tongan	`to`
Turkish	`tr`
Ukrainian	`uk`
Urdu	`ur`

TEXT LANGUAGE	LANGUAGE CODE
Vietnamese	`vi`
Welsh	`cy`
Yucatec Maya	`yua`

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

Regions of the Speech Service

10/30/2018 • 2 minutes to read • Edit Online

The Speech Service is available in different regions. When you create a subscription, you can select an available region based on your needs.

When you use your subscription, you have to account for the region you selected.

REST API

Use the REST API to select the correct region-specific endpoints. See REST APIs for details.

Speech SDK

In the Speech Service SDK, regions are specified as a string (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

Regions for speech recognition and translation

The following table lists the available regions for **speech recognition** and **translation**.

REGION	SPEECH SDK PARAMETER	SPEECH CUSTOMIZATION PORTAL
West US	`westus`	https://westus.cris.ai
West US2	`westus2`	https://westus2.cris.ai
East US	`eastus`	https://eastus.cris.ai
East US2	`eastus2`	https://eastus2.cris.ai
East Asia	`eastasia`	https://eastasia.cris.ai
South East Asia	`southeastasia`	https://southeastasia.cris.ai
North Europe	`northeurope`	https://northeurope.cris.ai
West Europe	`westeurope`	https://westeurope.cris.ai

Regions for intent recognition

Available regions for **intent recognition** via the Speech SDK are listed on the Language Understanding service region page. For each publishing region listed, the corresponding Speech SDK region parameter is determined as the first part of the domain name of the endpoint. For example, use `westus` to specify the West US publishing region.

Speech Service REST APIs

10/24/2018 • 13 minutes to read • Edit Online

The REST APIs of the Azure Cognitive Services Speech service are similar to the APIs provided by the Bing Speech API. The endpoints differ from the endpoints used by the Bing Speech service. Regional endpoints are available, and you must use a subscription key that corresponds to the endpoint you're using.

Speech to Text

The endpoints for the Speech to Text REST API are shown in the following table. Use the one that matches your subscription region.

REGION	SPEECH TO TEXT ENDPOINT
West US	`https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiv`
West US2	`https://westus2.stt.speech.microsoft.com/speech/recognition/conversation/cogniti`
East US	`https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiv`
East US2	`https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiv`
East Asia	`https://eastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognit`
South East Asia	`https://southeastasia.stt.speech.microsoft.com/speech/recognition/conversation/c`
North Europe	`https://northeurope.stt.speech.microsoft.com/speech/recognition/conversation/cog`
West Europe	`https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cogn`

NOTE

You must append the required language in the URI to avoid an HTTP 401 error. So for en-US, the correct URI is https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US.

NOTE

If you customized the acoustic model or language model, or pronunciation, use your custom endpoint instead.

This API supports only short utterances. Requests may contain up to 10 seconds of audio and last a maximum of 14 seconds overall. The REST API returns only final results, not partial or interim results. The Speech service also has a batch transcription API that can transcribe longer audio.

Query parameters

The following parameters may be included in the query string of the REST request.

PARAMETER NAME	REQUIRED/OPTIONAL	MEANING
`language`	Required	The identifier of the language to be recognized. See Supported languages.
`format`	Optional default: `simple`	Result format, `simple` or `detailed`. Simple results include `RecognitionStatus`, `DisplayText`, `Offset`, and duration. Detailed results include multiple candidates with confidence values and four different representations.
`profanity`	Optional default: `masked`	How to handle profanity in recognition results. May be `masked` (replaces profanity with asterisks), `removed` (removes all profanity), or `raw` (includes profanity).

Request headers

The following fields are sent in the HTTP request header.

HEADER	MEANING
`Ocp-Apim-Subscription-Key`	Your Speech service subscription key. Either this header or `Authorization` must be provided.
`Authorization`	An authorization token preceded by the word `Bearer`. Either this header or `Ocp-Apim-Subscription-Key` must be provided. See Authentication.
`Content-type`	Describes the format and codec of the audio data. Currently, this value must be `audio/wav; codec=audio/pcm; samplerate=16000`.
`Transfer-Encoding`	Optional. If given, must be `chunked` to allow audio data to be sent in multiple small chunks instead of a single file.
`Expect`	If using chunked transfer, send `Expect: 100-continue`. The Speech service acknowledges the initial request and awaits additional data.
`Accept`	Optional. If provided, must include `application/json`, as the Speech service provides results in JSON format. (Some Web request frameworks provide an incompatible default value if you do not specify one, so it is good practice to always include `Accept`.)

Audio format

The audio is sent in the body of the HTTP `POST` request. It should be in 16-bit WAV format with PCM single channel (mono) at 16 KHz of the following formats/encoding.

- WAV format with PCM codec
- Ogg format with OPUS codec

> **NOTE**
> The above formats are supported through REST API and WebSocket in the Speech Service. The Speech SDK currently only supports the WAV format with PCM codec.

Chunked transfer

Chunked transfer (`Transfer-Encoding: chunked`) can help reduce recognition latency because it allows the Speech service to begin processing the audio file while it's being transmitted. The REST API does not provide partial or interim results. This option is intended solely to improve responsiveness.

The following code illustrates 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.

```
using (fs = new FileStream(audioFile, FileMode.Open, FileAccess.Read))
{

    /*
    * Open a request stream and write 1024 byte chunks in the stream one at a time.
    */
    byte[] buffer = null;
    int bytesRead = 0;
    using (Stream requestStream = request.GetRequestStream())
    {
        /*
        * Read 1024 raw bytes from the input audio file.
        */
        buffer = new Byte[checked((uint)Math.Min(1024, (int)fs.Length))];
        while ((bytesRead = fs.Read(buffer, 0, buffer.Length)) != 0)
        {
            requestStream.Write(buffer, 0, bytesRead);
        }

        // Flush
        requestStream.Flush();
    }
}
```

Example request

The following is a typical request.

```
POST speech/recognition/conversation/cognitiveservices/v1?language=en-US&format=detailed HTTP/1.1
Accept: application/json;text/xml
Content-Type: audio/wav; codec=audio/pcm; samplerate=16000
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.stt.speech.microsoft.com
Transfer-Encoding: chunked
Expect: 100-continue
```

HTTP status

The HTTP status of the response indicates success or common error conditions.

HTTP CODE	MEANING	POSSIBLE REASON
100	Continue	The initial request has been accepted. Proceed with sending the rest of the data. (Used with chunked transfer.)
200	OK	The request was successful; the response body is a JSON object.
400	Bad request	Language code not provided or is not a supported language; invalid audio file.
401	Unauthorized	Subscription key or authorization token is invalid in the specified region, or invalid endpoint.
403	Forbidden	Missing subscription key or authorization token.

JSON response

Results are returned in JSON format. The `simple` format includes only the following top-level fields.

FIELD NAME	CONTENT
`RecognitionStatus`	Status, such as `Success` for successful recognition. See next table.
`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 might contain the following values.

STATUS VALUE	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 the same fields as the `simple` format, along with an `NBest` field. The `NBest` field is a list of alternative interpretations of the same speech, ranked from most likely to least likely. The first entry is the same as the main recognition result. Each entry contains the following fields:

FIELD NAME	CONTENT
`Confidence`	The confidence score of the entry from 0.0 (no confidence) to 1.0 (full confidence)
`Lexical`	The lexical form of the recognized text: the actual words recognized.
`ITN`	The inverse-text-normalized ("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. Same as `DisplayText` in the top-level result.

Sample responses

The following is a typical response for `simple` recognition.

```
{
  "RecognitionStatus": "Success",
  "DisplayText": "Remind me to buy 5 pencils.",
  "Offset": "1236645672289",
  "Duration": "1236645672289"
}
```

The following is a typical response for `detailed` recognition.

```
{
  "RecognitionStatus": "Success",
  "Offset": "1236645672289",
  "Duration": "1236645672289",
  "NBest": [
      {
        "Confidence" : "0.87",
        "Lexical" : "remind me to buy five pencils",
        "ITN" : "remind me to buy 5 pencils",
        "MaskedITN" : "remind me to buy 5 pencils",
        "Display" : "Remind me to buy 5 pencils.",
      },
      {
        "Confidence" : "0.54",
        "Lexical" : "rewind me to buy five pencils",
        "ITN" : "rewind me to buy 5 pencils",
        "MaskedITN" : "rewind me to buy 5 pencils",
        "Display" : "Rewind me to buy 5 pencils.",
      }
  ]
}
```

Text to Speech

The following are the REST endpoints for the Speech service's Text to Speech API. Use the endpoint that matches your subscription region.

REGION	TEXT TO SPEECH ENDPOINT
West US	`https://westus.tts.speech.microsoft.com/cognitiveservices/v1`
West US2	`https://westus2.tts.speech.microsoft.com/cognitiveservices/v1`
East US	`https://eastus.tts.speech.microsoft.com/cognitiveservices/v1`
East US2	`https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1`
East Asia	`https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1`
South East Asia	`https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1`
North Europe	`https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1`
West Europe	`https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1`

> **NOTE**
> If you created a custom voice font, use the endpoint you created for it instead of the endpoints here.

The Speech service supports 24-KHz audio output in addition to the 16-Khz output supported by Bing Speech. Four 24-KHz output formats are available for use in the `X-Microsoft-OutputFormat` HTTP header, as are two 24-KHz voices, `Jessa24kRUS` and `Guy24kRUS`.

LOCALE	LANGUAGE	GENDER	SERVICE NAME MAPPING
en-US	US English	Female	"Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)"
en-US	US English	Male	"Microsoft Server Speech Text to Speech Voice (en-US, Guy24kRUS)"

A full list of available voices is available in Supported languages.

Request headers

The following fields are sent in the HTTP request header.

HEADER	MEANING
`Authorization`	An authorization token preceded by the word `Bearer`. Required. See Authentication.
`Content-Type`	The input content type: `application/ssml+xml`.
`X-Microsoft-OutputFormat`	The output audio format. See next table.
`User-Agent`	Application name. Required; must contain fewer than 255 characters.

The available audio output formats (`X-Microsoft-OutputFormat`) incorporate both a bit rate and an encoding.

`raw-16khz-16bit-mono-pcm`	`raw-8khz-8bit-mono-mulaw`
`riff-8khz-8bit-mono-mulaw`	`riff-16khz-16bit-mono-pcm`
`audio-16khz-128kbitrate-mono-mp3`	`audio-16khz-64kbitrate-mono-mp3`
`audio-16khz-32kbitrate-mono-mp3`	`raw-24khz-16bit-mono-pcm`
`riff-24khz-16bit-mono-pcm`	`audio-24khz-160kbitrate-mono-mp3`
`audio-24khz-96kbitrate-mono-mp3`	`audio-24khz-48kbitrate-mono-mp3`

> **NOTE**
> If your selected voice and output format have different bit rates, the audio is resampled as necessary. However, 24khz voices do not support `audio-16khz-16kbps-mono-siren` and `riff-16khz-16kbps-mono-siren` output formats.

Request body

The text to be converted to speech is sent as the body of an HTTP `POST` request in either plain text (ASCII or UTF-8) or Speech Synthesis Markup Language (SSML) format (UTF-8). Plain text requests use the service's default voice and language. Send SSML to use a different voice.

Sample request

The following HTTP request uses an SSML body to choose the voice. The body must be no longer than 1,000 characters.

```
POST /cognitiveservices/v1 HTTP/1.1

X-Microsoft-OutputFormat: raw-16khz-16bit-mono-pcm
Content-Type: application/ssml+xml
Host: westus.tts.speech.microsoft.com
Content-Length: 225
Authorization: Bearer [Base64 access_token]

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Female'
    name='Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)'>
        Microsoft Speech Service Text-to-Speech API
</voice></speak>
```

HTTP response

The HTTP status of the response indicates success or common error conditions.

HTTP CODE	MEANING	POSSIBLE REASON
200	OK	The request was successful; the response body is an audio file.
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
413	Request Entity Too Large	The SSML input is longer than 1024 characters.
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 or saved to a buffer or file for later playback or other use.

Authentication

Sending a request to the Speech service's REST API requires either a subscription key or an access token. In general, it's easiest to send the subscription key directly. The Speech service then obtains the access token for you. To minimize response time, you might want to use an access token instead.

To obtain a token, present your subscription key to a regional Speech service `issueToken` endpoint, as shown in the following table. Use the endpoint that matches your subscription region.

REGION	TOKEN SERVICE ENDPOINT
West US	`https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken`
West US2	`https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken`
East US	`https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken`
East US2	`https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken`
East Asia	`https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken`
South East Asia	`https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken`
North Europe	`https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken`
West Europe	`https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken`

Each access token is valid for 10 minutes. You can obtain a new token at any time. If you like, you can obtain a token just before every Speech REST API request. To minimize network traffic and latency, we recommend using the same token for nine minutes.

The following sections show how to get a token and how to use it in a request.

Get a token: HTTP

The following example is a sample HTTP request for obtaining 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 to this request is the access token in Java Web Token (JWT) format.

Get a token: PowerShell

The following Windows PowerShell script illustrates how to obtain an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech service subscription key. If your subscription isn't in the West US region, change the host name of the given URI accordingly.

```
$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
```

Get a token: cURL

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). The following cURL command illustrates how to obtain an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech service subscription key. If your subscription isn't in the West US region, change the host name of the given URI accordingly.

> **NOTE**
> The command is shown on multiple lines for readability, but enter it on a single line at a shell prompt.

```
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"
```

Get a token: C#

The following C# class illustrates how to obtain 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 host name of `FetchTokenUri` appropriately.

```
/*
    * This class demonstrates how to get a valid access token.
    */
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();
        }
    }
}
```

Use a token

To use a token in a REST API request, provide it in the `Authorization` header, following the word `Bearer`. Here is a sample Text to Speech REST request that contains a token. Substitute your actual token for `YOUR_ACCESS_TOKEN`. Use the correct host name in the `Host` header.

```
POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.tts.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice name='Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)'>
    Hello, world!
</voice></speak>
```

Renew authorization

The authorization token expires after 10 minutes. Renew your authorization by obtaining a new token before it expires. As an example, you can obtain a new token after nine minutes.

The following C# code is a drop-in replacement for the class presented earlier. The `Authentication` class automatically obtains a new access token every nine minutes by using a timer. This approach ensures that a valid token is always available while your program is running.

> **NOTE**
> Instead of using a timer, you can store a timestamp of when the last token was obtained. Then you can request a new one only if it's close to expiring. This approach avoids requesting new tokens unnecessarily and might be more suitable for programs that make infrequent Speech requests.

As before, make sure the `FetchTokenUri` value matches your subscription region. Pass your subscription key when you instantiate the class.

```
/*
    * This class demonstrates how to maintain a valid access token.
    */
public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
    private string subscriptionKey;
    private string token;
    private Timer accessTokenRenewer;

    //Access token expires every 10 minutes. Renew it every 9 minutes.
    private const int RefreshTokenDuration = 9;

    public Authentication(string subscriptionKey)
    {
        this.subscriptionKey = subscriptionKey;
        this.token = FetchToken(FetchTokenUri, subscriptionKey).Result;

        // renew the token on set duration.
        accessTokenRenewer = new Timer(new TimerCallback(OnTokenExpiredCallback),
                                       this,
                                       TimeSpan.FromMinutes(RefreshTokenDuration),
                                       TimeSpan.FromMilliseconds(-1));
    }

    public string GetAccessToken()
    {
        return this.token;
    }

    private void RenewAccessToken()
    {
        this.token = FetchToken(FetchTokenUri, this.subscriptionKey).Result;
        Console.WriteLine("Renewed token.");
    }

    private void OnTokenExpiredCallback(object stateInfo)
    {
        try
        {
            RenewAccessToken();
        }
        catch (Exception ex)
        {
            Console.WriteLine(string.Format("Failed renewing access token. Details: {0}", ex.Message));
        }
        finally
        {
            try
            {
                accessTokenRenewer.Change(TimeSpan.FromMinutes(RefreshTokenDuration), TimeSpan.FromMilliseconds(-1));
            }
            catch (Exception ex)
            {
                Console.WriteLine(string.Format("Failed to reschedule the timer to renew access token. Details: {0}", ex.Message));
            }
        }
    }

    private async Task<string> FetchToken(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();
        }
    }
}
```

Next steps

- Get your Speech trial subscription
- Customize acoustic models
- Customize language models

Speech SDK API reference

10/19/2018 • 2 minutes to read • Edit Online

The Speech Software Development Kit (SDK) gives your applications access to the functions of the Speech service, making it easier to develop speech-enabled software. Currently, the SDKs provide access to **Speech to Text**, **Speech Translation**, and **Intent Recognition**. A general overview about the capabilites and supported platforms can be found on the documentation entry page.

PROGRAMMING LANGUAGE	PLATFORM	API REFERENCE
C#, .NET Standard	Windows, UWP, .NET Standard (Windows)	Browse
C/C++	Windows, Linux	Browse
Java	Android	Browse
Java*	Devices	Browse
Objective C	iOS	Browse
JavaScript	Browser	Browse

** The Java SDK is also available as part of the Speech Devices SDK.*

Next steps

- Try the Speech service for free
- Get your Speech trial subscription
- See how to recognize speech in C#

Speech Synthesis Markup Language

10/19/2018 • 2 minutes to read • Edit Online

The Speech Synthesis Markup Language (SSML) is an XML-based markup language that provides a way to control the pronunciation and *prosody* of text-to-speech. Prosody refers to the rhythm and pitch of speech—its music, if you will. You can specify words phonetically, provide hints for interpreting numbers, insert pauses, control pitch, volume, and rate, and more.

For more information, see Speech Synthesis Markup Language (SSML) Version 1.0 at the W3C.

The following examples show how to use SSML for common speech synthesis needs:

Add a break

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)'>
    Welcome to Microsoft Cognitive Services <break time="100ms" /> Text-to-Speech API.
</voice> </speak>
```

Change speaking rate

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, Guy24kRUS)'>
<prosody rate="+30.00%">
    Welcome to Microsoft Cognitive Services Text-to-Speech API.
</prosody></voice> </speak>
```

Pronunciation

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)'>
    <phoneme alphabet="ipa" ph="t&#x259;mei&#x325;&#x27E;ou&#x325;"> tomato </phoneme>
</voice> </speak>
```

Change volume

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>
<prosody volume="+20.00%">
    Welcome to Microsoft Cognitive Services Text-to-Speech API.
</prosody></voice> </speak>
```

Change pitch

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
    <voice  name='Microsoft Server Speech Text to Speech Voice (en-US, Guy24kRUS)'>
    Welcome to <prosody pitch="high">Microsoft Cognitive Services Text-to-Speech API.</prosody>
</voice> </speak>
```

Change pitch contour

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>
<prosody contour="(80%,+20%) (90%,+30%)" >
    Good morning.
</prosody></voice> </speak>
```

Use multiple voices

```
<speak version='1.0' xmlns="http://www.w3.org/2001/10/synthesis" xml:lang='en-US'>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>
    Good morning!
</voice>
<voice  name='Microsoft Server Speech Text to Speech Voice (en-US, Guy24kRUS)'>
    Good morning to you too Jessa!
</voice> </speak>
```

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

Speech to Text frequently asked questions

10/19/2018 • 7 minutes to read • Edit Online

If you can't find answers to your questions in this FAQ, check out other support options.

General

Q: What is the difference between a baseline model and a custom Speech to Text model?

A: A baseline model has been trained by using Microsoft-owned data and is already deployed in the cloud. You can use a custom model to adapt a model to better fit a specific environment that has specific ambient noise or language. Factory floors, cars, or noisy streets would require an adapted acoustic model. Topics like biology, physics, radiology, product names, and custom acronyms would require an adapted language model.

Q: Where do I start if I want to use a baseline model?

A: First, get a subscription key. If you want to make REST calls to the predeployed baseline models, see the REST APIs. If you want to use WebSockets, download the SDK.

Q: Do I always need to build a custom speech model?

A: No. If your application uses generic, day-to-day language, you don't need to customize a model. If your application is used in an environment where there's little or no background noise, you don't need to customize a model.

You can deploy baseline and customized models in the portal and then run accuracy tests against them. You can use this feature to measure the accuracy of a baseline model versus a custom model.

Q: How will I know when processing for my dataset or model is complete?

A: Currently, the status of the model or dataset in the table is the only way to know. When the processing is complete, the status is **Succeeded**.

Q: Can I create more than one model?

A: There's no limit on the number of models you can have in your collection.

Q: I realized I made a mistake. How do I cancel my data import or model creation that's in progress?

A: Currently, you can't roll back an acoustic or language adaptation process. You can delete imported data and models when they're in a terminal state.

Q: What's the difference between the Search and Dictation model and the Conversational model?

A: You can choose from more than one baseline model in the Speech service. The Conversational model is useful for recognizing speech that is spoken in a conversational style. This model is ideal for transcribing phone calls. The Search and Dictation model is ideal for voice-triggered apps. The Universal model is a new model that aims to address both scenarios.

Q: Can I update my existing model (model stacking)?

A: You can't update an existing model. As a solution, combine the old dataset with the new dataset and readapt.

The old dataset and the new dataset must be combined in a single .zip file (for acoustic data) or in a .txt file (for language data). When adaptation is finished, the new, updated model needs to be redeployed to obtain a new endpoint

Q: When a new version of a baseline is available is my deployment automatically updated?

A: Deployments will NOT be automatically updated.

If you have adapted and deployed a model with baseline V1.0, that deployment will remain as is. Customers can decommision the deployed model, re-adapt using the newer version of the baseline and re-deploy.

Q: What if I need higher concurrency for my deployed model than what is offered in the portal?

A: You can scale up your model in increments of 20 concurrent requests.

Contact us if you require a higher scale.

Q: Can I download my model and run it locally?

A: Models can't be downloaded and executed locally.

Q: Are my requests logged?

A: You have a choice when you create a deployment to switch off tracing. At that point, no audio or transcriptions will be logged. Otherwise, requests are typically logged in Azure in secure storage.

Q: Are my requests throttled?

A: The REST API limits requests to 25 per 5 seconds. Details can be found in our pages for Speech to text.

If you have further privacy concerns that prohibit you from using the custom Speech service, contact one of the support channels.

Importing data

Q: What is the limit on the size of a dataset, and why is it the limit?

A: The current limit for a dataset is 2 GB. The limit is due to the restriction on the size of a file for HTTP upload.

Q: Can I zip my text files so I can upload a larger text file?

A: No. Currently, only uncompressed text files are allowed.

Q: The data report says there were failed utterances. What is the issue?

A: Failing to upload 100 percent of the utterances in a file is not a problem. If the vast majority of the utterances in an acoustic or language dataset (for example, more than 95 percent) are successfully imported, the dataset can be usable. However, we recommend that you try to understand why the utterances failed and fix the problems. Most common problems, such as formatting errors, are easy to fix.

Creating an acoustic model

Q: How much acoustic data do I need?

A: We recommend starting with between 30 minutes and one hour of acoustic data.

Q: What data should I collect?

A: Collect data that's as close to the application scenario and use case as possible. The data collection should match the target application and users in terms of device or devices, environments, and types of speakers. In general, you should collect data from as broad a range of speakers as possible.

Q: How should I collect acoustic data?

A: You can create a standalone data collection application or use off-the-shelf audio recording software. You can also create a version of your application that logs the audio data and then uses the data.

Q: Do I need to transcribe adaptation data myself?

A: Yes! You can transcribe it yourself or use a professional transcription service. Some users prefer professional transcribers and others use crowdsourcing or do the transcriptions themselves.

Accuracy testing

Q: Can I perform offline testing of my custom acoustic model by using a custom language model?

A: Yes, just select the custom language model in the drop-down menu when you set up the offline test.

Q: Can I perform offline testing of my custom language model by using a custom acoustic model?

A: Yes, just select the custom acoustic model in the drop-down menu when you set up the offline test.

Q: What is word error rate (WER) and how is it computed?

A: WER is the evaluation metric for speech recognition. WER is counted as the total number of errors, which includes insertions, deletions, and substitutions, divided by the total number of words in the reference transcription. For more information, see word error rate.

Q: How do I determine whether the results of an accuracy test are good?

A: The results show a comparison between the baseline model and the model you customized. You should aim to beat the baseline model to make customization worthwhile.

Q: How do I determine the WER of a base model so I can see if there was an improvement?

A: The offline test results show the baseline accuracy of the custom model and the improvement over baseline.

Creating a language model

Q: How much text data do I need to upload?

A: It depends on how different the vocabulary and phrases used in your application are from the starting language models. For all new words, it's useful to provide as many examples as possible of the usage of those words. For common phrases that are used in your application, including phrases in the language data is also useful because it tells the system to also listen for these terms. It's common to have at least 100, and typically several hundred or more utterances in the language dataset. Also, if some types of queries are expected to be more common than others, you can insert multiple copies of the common queries in the dataset.

Q: Can I just upload a list of words?

A: Uploading a list of words will add the words to the vocabulary, but it won't teach the system how the words are typically used. By providing full or partial utterances (sentences or phrases of things that users are likely to say), the language model can learn the new words and how they are used. The custom language model is good not only for adding new words to the system, but also for adjusting the likelihood of known words for your application. Providing full utterances helps the system learn better.

Next steps

- Troubleshooting
- Release notes

Text to Speech frequently asked questions

10/19/2018 • 3 minutes to read • Edit Online

If you can't find answers to your questions in this FAQ, check out other support options.

General

Q: What is the difference between a standard voice model and a custom voice model?

A: The standard voice model (also called a *voice font*) has been trained by using Microsoft-owned data and is already deployed in the cloud. You can use a custom voice model either to adapt an average model and transfer the timbre and expression of the speaker's voice style or train a full, new model based on the training data prepared by the user. Today, more and more customers want a one-of-a-kind, branded voice for their bots. The custom voice-building platform is the right choice for that option.

Q: Where do I start if I want to use a standard voice model?

A: More than 80 standard voice models in over 45 languages are available through HTTP requests. First, get a subscription key. To make REST calls to the predeployed voice models, see the REST API.

Q: If I want to use a customized voice model, is the API the same as the one that's used for standard voices?

A: When a custom voice model is created and deployed, you get a unique endpoint for your model. To use the voice to speak in your apps, you must specify the endpoint in your HTTP requests. The same functionality that's available in the REST API for the Text to Speech service is available for your custom endpoint. Learn how to create and use your custom endpoint.

Q: Do I need to prepare the training data to create custom voice models on my own?

A: Yes, you must prepare the training data yourself for a custom voice model.

A collection of speech data is required to create a customized voice model. This collection consists of a set of audio files of speech recordings and a text file of the transcription of each audio file. The result of your digital voice relies heavily on the quality of your training data. To produce a good text-to-speech voice, it's important that the recordings are made in a quiet room with a high-quality, standing microphone. A consistent volume, speaking rate, and speaking pitch, and even consistency in expressive mannerisms of speech are essential for building a great digital voice. We highly recommend recording the voices in a recording studio.

Currently, we don't provide online recording support or have any recording studio recommendations. For the format requirement, see how to prepare recordings and transcripts.

Q: What scripts should I use to record the speech data for custom voice training?

A: We don't limit the scripts for voice recording. You can use your own scripts to record the speech. Just ensure that you have sufficient phonetic coverage in your speech data. To train a custom voice, you can start with a small volume of speech data, which might be 50 different sentences (about 3-5 minutes of speech). The more data you provide, the more natural your voice will be. You can start to train a full voice font when you provide recordings of more than 2,000 sentences (about 3-4 hours of speech). To get a high-quality, full voice, you need to prepare recordings of more than 6,000 sentences (about 8-10 hours of speech).

We provide additional services to help you prepare scripts for recording. Contact Custom Voice customer support for inquiries.

Q: What if I need higher concurrency than the default value or what is offered in the portal?

A: You can scale up your model in increments of 20 concurrent requests. Contact Custom Voice customer support for inquiries about higher scaling.

Q: Can I download my model and run it locally?

A: Models can't be downloaded and executed locally.

Q: Are my requests throttled?

A: The REST API limits requests to 25 per 5 seconds. Details can be found in our pages for Text to Speech.

Next steps

- Troubleshooting
- Release notes

About the Speech Service SDK

10/19/2018 • 2 minutes to read • Edit Online

The Speech Service Software Development Kit (SDK) gives your applications native access to the functions of the Speech service, making it easier to develop software. Currently, the SDK provides access to **Speech to Text**, **Speech Translation**, and **Intent Recognition**.

PROGRAMMING LANGUAGE	PLATFORM	API REFERENCE
C#, .NET Standard	Windows, UWP, .NET Standard (Windows)	Browse
C/C++	Windows, Linux	Browse
Java	Android	Browse
Java*	Devices	Browse
Objective C	iOS	Browse
JavaScript	Browser	Browse

** The Java SDK is also available as part of the Speech Devices SDK.*

IMPORTANT

By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

Get the SDK

Windows

For Windows, we support the following languages:

- C# (UWP and .NET), C++: You can reference and use the latest version of our Speech SDK NuGet package. The package includes 32-bit and 64-bit client libraries and managed (.NET) libraries. The SDK can be installed in Visual Studio by using NuGet. Search for **Microsoft.CognitiveServices.Speech**.
- Java: You can reference and use the latest version of our Speech SDK Maven package, which supports only Windows x64. In your Maven project, add `https://csspeechstorage.blob.core.windows.net/maven/` as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1` as a dependency.

Linux

NOTE

Currently, we support only Ubuntu 16.04 on a PC (x86 or x64 for C++ development and x64 for .NET Core and Java).

Make sure you have the required compiler and libraries installed by running the following shell commands:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libcurl3 libasound2
```

- C#: You can reference and use the latest version of our Speech SDK NuGet package. To reference the SDK, add the following package reference to your project:

  ```
  <PackageReference Include="Microsoft.CognitiveServices.Speech" Version="1.0.1" />
  ```

- Java: You can reference and use the latest version of our Speech SDK Maven package. In your Maven project, add `https://csspeechstorage.blob.core.windows.net/maven/` as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1` as a dependency.

- C++: Download the SDK as a .tar package and unpack the files in a directory of your choice. The following table shows the SDK folder structure:

PATH	DESCRIPTION
`license.md`	License
`ThirdPartyNotices.md`	Third-party notices
`include`	Header files for C and C++
`lib/x64`	Native x64 library for linking with your application
`lib/x86`	Native x86 library for linking with your application

  To create an application, copy or move the required binaries (and libraries) into your development environment. Include them as required in your build process.

Android

The Java SDK for Android is packaged as an AAR (Android Library), which includes the necessary libraries as well as required Android permissions for using it. It's hosted in a Maven repository at `https://csspeechstorage.blob.core.windows.net/maven/` as package `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1`.

To consume the package from your Android Studio project, make the following changes:

- In the project-level build.gradle file, add the following to the `repository` section:

  ```
  maven { url 'https://csspeechstorage.blob.core.windows.net/maven/' }
  ```

- In the module-level build.gradle file, add the following to the `dependencies` section:

  ```
  implementation 'com.microsoft.cognitiveservices.speech:client-sdk:1.0.1'
  ```

The Java SDK is also part of the Speech Devices SDK.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

About the Speech Service SDK

10/19/2018 • 2 minutes to read • Edit Online

The Speech Service Software Development Kit (SDK) gives your applications native access to the functions of the Speech service, making it easier to develop software. Currently, the SDK provides access to **Speech to Text**, **Speech Translation**, and **Intent Recognition**.

PROGRAMMING LANGUAGE	PLATFORM	API REFERENCE
C#, .NET Standard	Windows, UWP, .NET Standard (Windows)	Browse
C/C++	Windows, Linux	Browse
Java	Android	Browse
Java*	Devices	Browse
Objective C	iOS	Browse
JavaScript	Browser	Browse

** The Java SDK is also available as part of the Speech Devices SDK.*

IMPORTANT

By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

Get the SDK

Windows

For Windows, we support the following languages:

- C# (UWP and .NET), C++: You can reference and use the latest version of our Speech SDK NuGet package. The package includes 32-bit and 64-bit client libraries and managed (.NET) libraries. The SDK can be installed in Visual Studio by using NuGet. Search for **Microsoft.CognitiveServices.Speech**.
- Java: You can reference and use the latest version of our Speech SDK Maven package, which supports only Windows x64. In your Maven project, add `https://csspeechstorage.blob.core.windows.net/maven/` as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1` as a dependency.

Linux

NOTE

Currently, we support only Ubuntu 16.04 on a PC (x86 or x64 for C++ development and x64 for .NET Core and Java).

Make sure you have the required compiler and libraries installed by running the following shell commands:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libcurl3 libasound2
```

- C#: You can reference and use the latest version of our Speech SDK NuGet package. To reference the SDK, add the following package reference to your project:

  ```
  <PackageReference Include="Microsoft.CognitiveServices.Speech" Version="1.0.1" />
  ```

- Java: You can reference and use the latest version of our Speech SDK Maven package. In your Maven project, add `https://csspeechstorage.blob.core.windows.net/maven/` as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1` as a dependency.

- C++: Download the SDK as a .tar package and unpack the files in a directory of your choice. The following table shows the SDK folder structure:

PATH	DESCRIPTION
`license.md`	License
`ThirdPartyNotices.md`	Third-party notices
`include`	Header files for C and C++
`lib/x64`	Native x64 library for linking with your application
`lib/x86`	Native x86 library for linking with your application

  To create an application, copy or move the required binaries (and libraries) into your development environment. Include them as required in your build process.

Android

The Java SDK for Android is packaged as an AAR (Android Library), which includes the necessary libraries as well as required Android permissions for using it. It's hosted in a Maven repository at `https://csspeechstorage.blob.core.windows.net/maven/` as package `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1`.

To consume the package from your Android Studio project, make the following changes:

- In the project-level build.gradle file, add the following to the `repository` section:

  ```
  maven { url 'https://csspeechstorage.blob.core.windows.net/maven/' }
  ```

- In the module-level build.gradle file, add the following to the `dependencies` section:

  ```
  implementation 'com.microsoft.cognitiveservices.speech:client-sdk:1.0.1'
  ```

The Java SDK is also part of the Speech Devices SDK.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

About the Speech Service SDK

10/19/2018 • 2 minutes to read • Edit Online

The Speech Service Software Development Kit (SDK) gives your applications native access to the functions of the Speech service, making it easier to develop software. Currently, the SDK provides access to **Speech to Text**, **Speech Translation**, and **Intent Recognition**.

PROGRAMMING LANGUAGE	PLATFORM	API REFERENCE
C#, .NET Standard	Windows, UWP, .NET Standard (Windows)	Browse
C/C++	Windows, Linux	Browse
Java	Android	Browse
Java*	Devices	Browse
Objective C	iOS	Browse
JavaScript	Browser	Browse

** The Java SDK is also available as part of the Speech Devices SDK.*

IMPORTANT

By downloading any of the Cognitive Services Speech SDK components on this page, you acknowledge its license. See Speech SDK license agreement.

Get the SDK

Windows

For Windows, we support the following languages:

- C# (UWP and .NET), C++: You can reference and use the latest version of our Speech SDK NuGet package. The package includes 32-bit and 64-bit client libraries and managed (.NET) libraries. The SDK can be installed in Visual Studio by using NuGet. Search for **Microsoft.CognitiveServices.Speech**.
- Java: You can reference and use the latest version of our Speech SDK Maven package, which supports only Windows x64. In your Maven project, add `https://csspeechstorage.blob.core.windows.net/maven/` as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1` as a dependency.

Linux

NOTE

Currently, we support only Ubuntu 16.04 on a PC (x86 or x64 for C++ development and x64 for .NET Core and Java).

Make sure you have the required compiler and libraries installed by running the following shell commands:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libcurl3 libasound2
```

- C#: You can reference and use the latest version of our Speech SDK NuGet package. To reference the SDK, add the following package reference to your project:

  ```
  <PackageReference Include="Microsoft.CognitiveServices.Speech" Version="1.0.1" />
  ```

- Java: You can reference and use the latest version of our Speech SDK Maven package. In your Maven project, add `https://csspeechstorage.blob.core.windows.net/maven/` as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1` as a dependency.

- C++: Download the SDK as a .tar package and unpack the files in a directory of your choice. The following table shows the SDK folder structure:

PATH	DESCRIPTION
`license.md`	License
`ThirdPartyNotices.md`	Third-party notices
`include`	Header files for C and C++
`lib/x64`	Native x64 library for linking with your application
`lib/x86`	Native x86 library for linking with your application

  To create an application, copy or move the required binaries (and libraries) into your development environment. Include them as required in your build process.

Android

The Java SDK for Android is packaged as an AAR (Android Library), which includes the necessary libraries as well as required Android permissions for using it. It's hosted in a Maven repository at `https://csspeechstorage.blob.core.windows.net/maven/` as package `com.microsoft.cognitiveservices.speech:client-sdk:1.0.1`.

To consume the package from your Android Studio project, make the following changes:

- In the project-level build.gradle file, add the following to the `repository` section:

  ```
  maven { url 'https://csspeechstorage.blob.core.windows.net/maven/' }
  ```

- In the module-level build.gradle file, add the following to the `dependencies` section:

  ```
  implementation 'com.microsoft.cognitiveservices.speech:client-sdk:1.0.1'
  ```

The Java SDK is also part of the Speech Devices SDK.

Get the samples

For the latest samples, see the Cognitive Services Speech SDK sample code repository on GitHub.

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

Ship an application

10/19/2018 • 2 minutes to read • Edit Online

Observe the Speech SDK license, as well as the third-party software notices when you distribute the Azure Cognitive Services Speech SDK. Also, review the Microsoft Privacy Statement.

Depending on the platform, different dependencies exist to execute your application.

Windows

The Cognitive Services Speech SDK is tested on Windows 10 and on Windows Server 2016.

The Cognitive Services Speech SDK requires the Microsoft Visual C++ Redistributable for Visual Studio 2017 on the system. You can download installers for the latest version of the `Microsoft Visual C++ Redistributable for Visual Studio 2017` here:

- Win32
- x64

If your application uses managed code, the `.NET Framework 4.6.1` or later is required on the target machine.

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 (Win32/x64) that matches your application.

NAME	FUNCTION
`Microsoft.CognitiveServices.Speech.core.dll`	Core SDK, required for native and managed deployment
`Microsoft.CognitiveServices.Speech.csharp.bindings.dll`	Required for managed deployment
`Microsoft.CognitiveServices.Speech.csharp.dll`	Required for managed deployment

Linux

For a native application, you need to ship the Speech SDK library, `libMicrosoft.CognitiveServices.Speech.core.so`. Make sure you select the version (x86, x64) that matches your application. Depending on the Linux version, you also might need to include the following dependencies:

- The shared libraries of the GNU C library (including the POSIX Threads Programming library, `libpthreads`)
- The OpenSSL library (`libssl.so.1.0.0`)
- The cURL library (`libcurl.so.4`)
- The shared library for ALSA applications (`libasound.so.2`)

On Ubuntu 16.04, for example, the GNU C libraries should already be installed by default. The last three can be installed by using these commands:

```
sudo apt-get update
sudo apt-get install libssl1.0.0 libcurl3 libasound2 wget
```

Next steps

- Get your Speech trial subscription
- See how to recognize speech in C#

Release notes

10/19/2018 • 3 minutes to read • Edit Online

Speech Service SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our sample repository, a new sample for JavaScript was added.

Cognitive Services Speech SDK 1.0.0: 2018-September release

New features

- Support for Objective-C on iOS. Check out our Objective-C quickstart for iOS.
- Support for JavaScript in browser. Check out our JavaScript quickstart.

Breaking changes

- With this release a number of breaking changes are introduced. Please check this page for details.

Cognitive Services Speech SDK 0.6.0: 2018-August release

New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the UWP quickstart.
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the Java Runtime Environment quickstart.

Functional change

- Expose additional error detail information on connection errors.

Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.

- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

Cognitive Services Speech SDK 0.5.0: 2018-July release

New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the Android quickstart.
- Support .NET Standard 2.0 on Windows. Check out the .NET Core quickstart.
- Experimental: Support UWP on Windows (version 1709 or later).
  - Check out the UWP quickstart.
  - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support AuthorizationToken for creating factory instances.

Breaking changes

- Recognition events: NoMatch event type was merged into the Error event.
- SpeechOutputFormat in C# was renamed to OutputFormat to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a regions page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- AudioInputStream

  A recognizer now can consume a stream as the audio source. For more information, see the related how-to guide.

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- 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.

About the Speech Devices SDK (Preview)

10/19/2018 • 2 minutes to read • Edit Online

The Speech Service works with a wide variety of devices and audio sources. Now, you can take your speech applications to the next level with matched hardware and software. The Speech Devices SDK is a pretuned library that's paired with purpose-built, microphone array development kits.

The Speech Devices SDK can help you:

- Rapidly test new voice scenarios.
- More easily integrate the cloud-based Speech service into your device.
- Create an exceptional user experience for your customers.

The Speech Devices SDK consumes the Speech SDK. It uses the Speech SDK to send the audio that's processed by our advanced audio processing algorithm from the device's microphone array to the Speech service. It uses multichannel audio to provide more accurate far-field speech recognition via noise suppression, echo cancellation, beamforming, and dereverberation.

You can also use the Speech Devices SDK to build ambient devices that have your own customized wake word—so the cue that initiates a user interaction is unique to your brand.

The Speech Devices SDK facilitates a variety of voice-enabled scenarios, such as drive-thru ordering systems, in-store or in-home assistants, 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!

Development kit providers

Currently, these complete, end-to-end system reference designs are available:

roobo	ROOBO provides complete artificial intelligence (AI) system solutions for household electric appliances, automobiles, robots, toys, and other industries. ROOBO's reference designs greatly reduce development time-to-market via integration with the Microsoft Speech service. Visit ROOBO.

Next steps

To get started, get a free Azure account and sign up for the Speech Devices SDK.

Sign up for the Speech Devices SDK

Get the Cognitive Services Speech Devices SDK

10/19/2018 • 2 minutes to read • Edit Online

The Speech Devices SDK is in restricted preview and requires you to be enrolled in the program. Currently, Microsoft prefers large companies as candidates for access to this product.

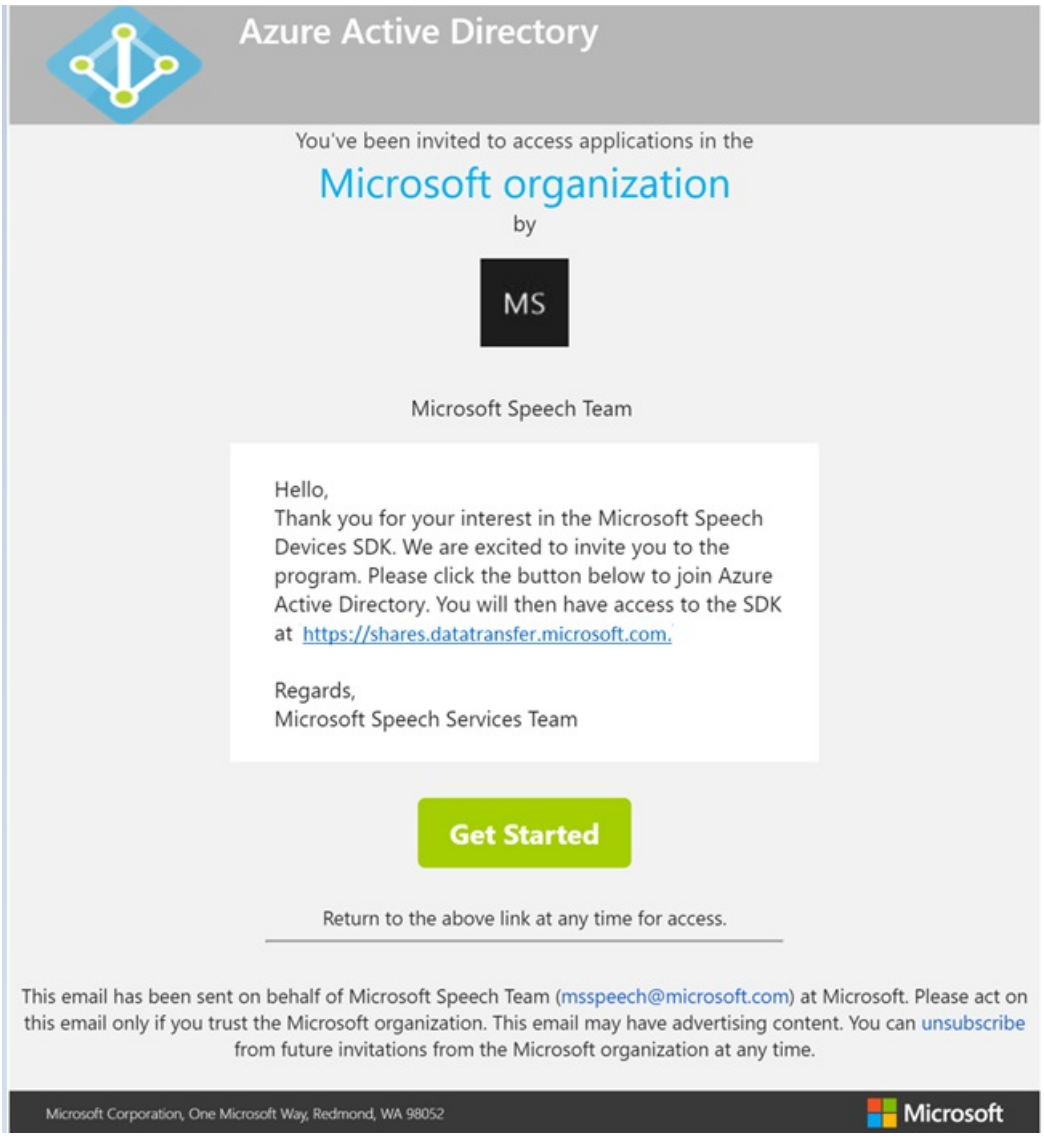
Request access

To get access to the Speech Devices SDK:

1. Go to the Microsoft Speech Devices SDK sign-up form.
2. Read the license agreement.
3. If you agree to the terms of the license agreement, select **I agree**.
4. Answer the questions in the form.
5. Submit the form.
6. If your email address is not already part of Azure Active Directory (Azure AD), you receive an invitation email like the following example when you're approved for access. If your email address is already in Azure AD, you receive an email message from the Microsoft Speech Team when you're approved for access, and you can skip ahead to Download the Speech Devices SDK.

Approval e-mail

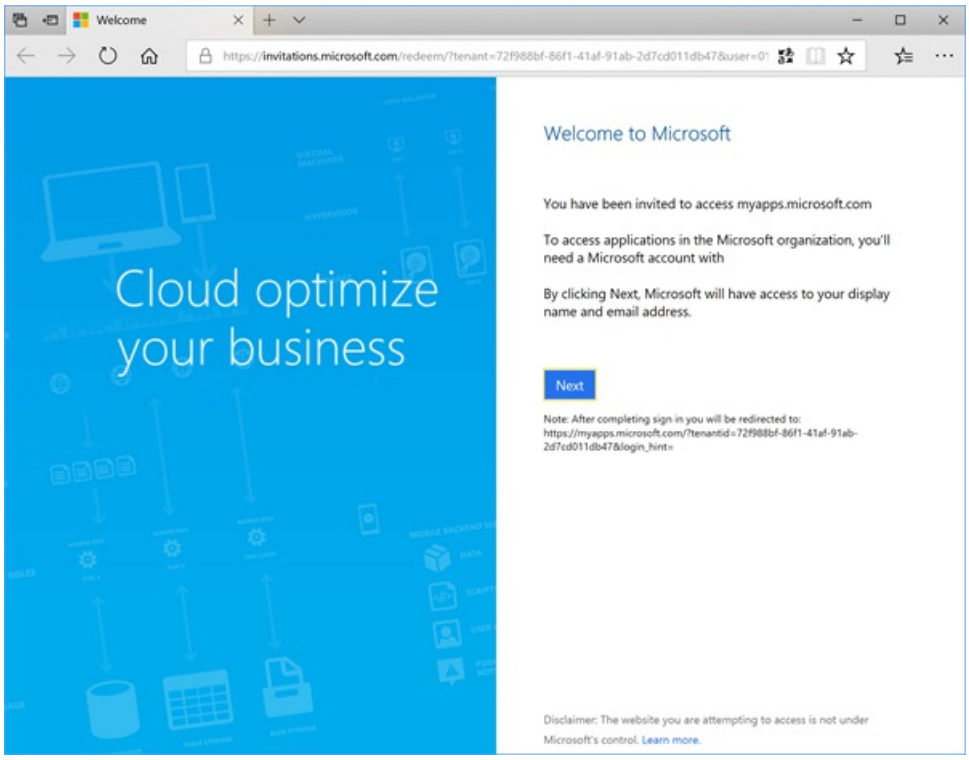
```
From: Microsoft Speech Team from Microsoft (via Microsoft) <invites@microsoft.com>
Subject: You're invited to the Microsoft organization
```



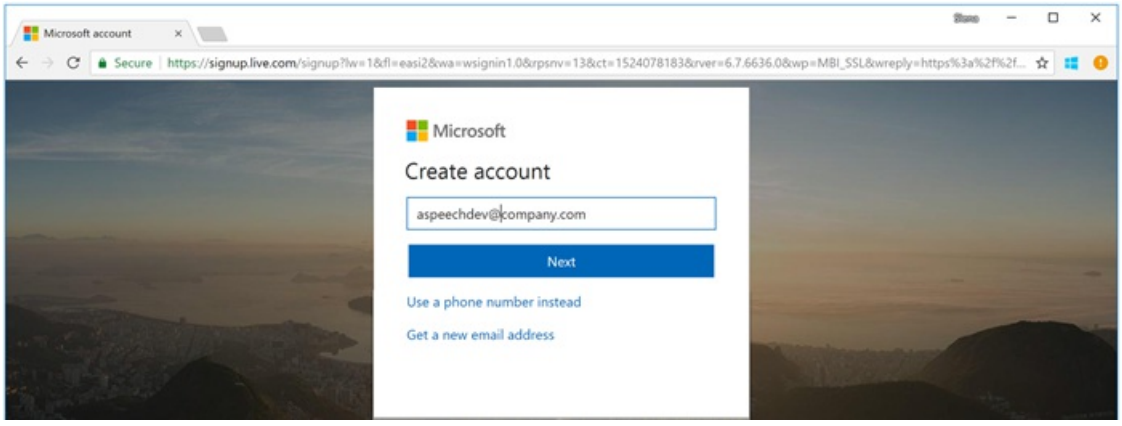
Accept access

Complete the following steps to join Azure AD with the email address you provided during registration. This process grants you access to the Speech Devices SDK download site.

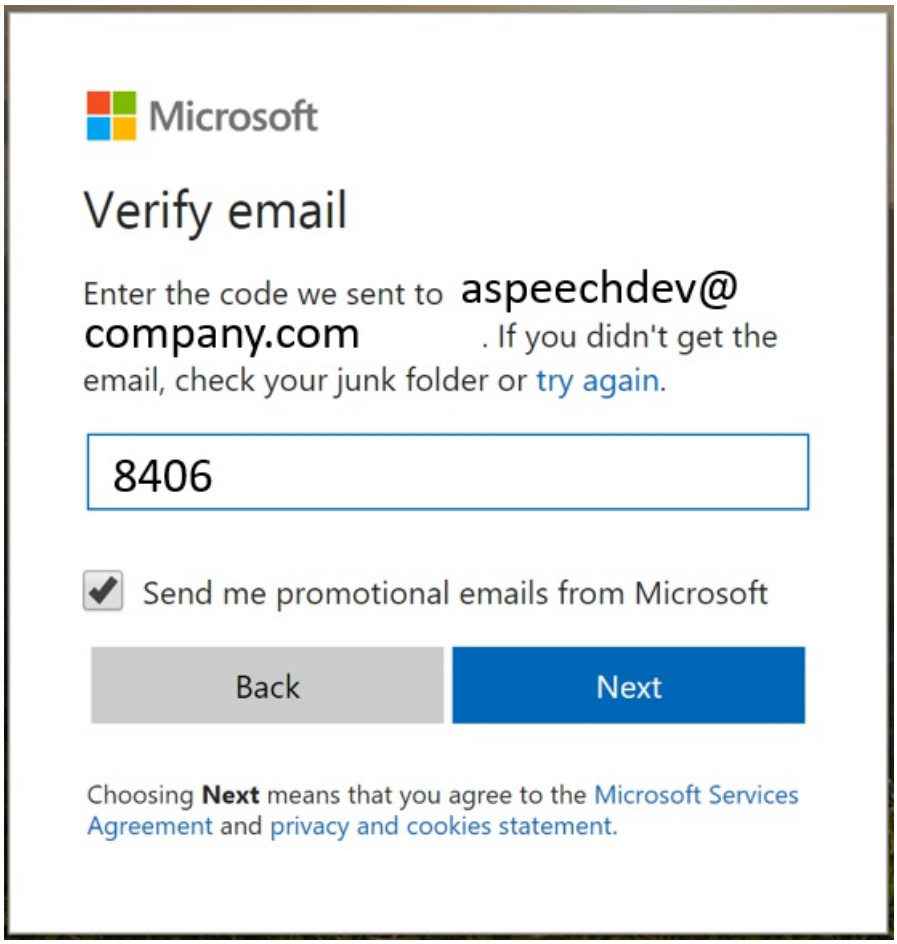
1. In the email message you received, select **Get Started**. If your organization is already an Office 365 customer, you are prompted to sign in and you can skip ahead to step 8.
2. In the browser window that opens, select **Next**.



3. Create a Microsoft account if you don't already have one. Enter the same email address at which you received the invitation email.



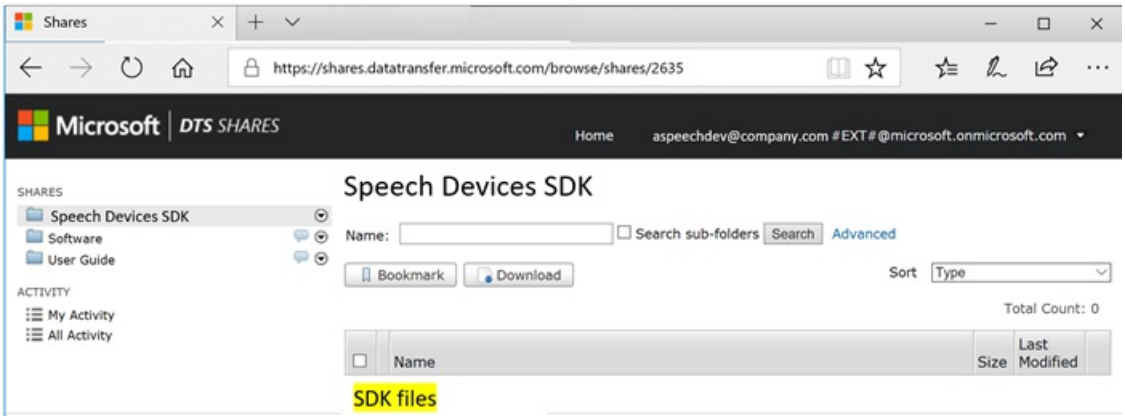
4. Select **Next** to create a password.
5. When prompted to verify your e-mail, get the verification code from the invitation email you received.
6. Paste or type the security code from the email message in the dialog box. In this example, the security code is **8406**. Select **Next**.



7. When you see the Access Panel Application in the browser, you have confirmed that your email address is part of Azure AD. You now have access to the Speech Devices SDK download site.

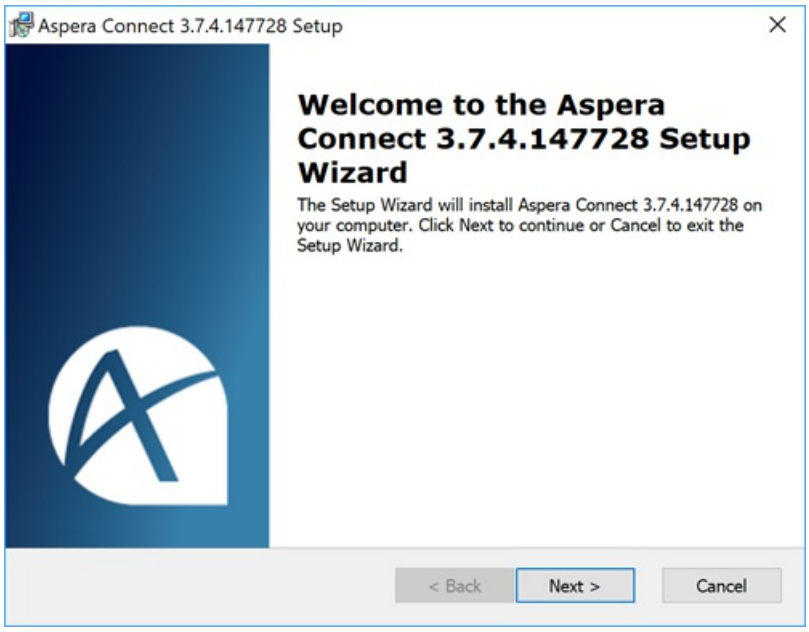
Download the Speech Devices SDK

Go to the Speech Devices SDK download site. Sign in with the Microsoft account you created earlier.

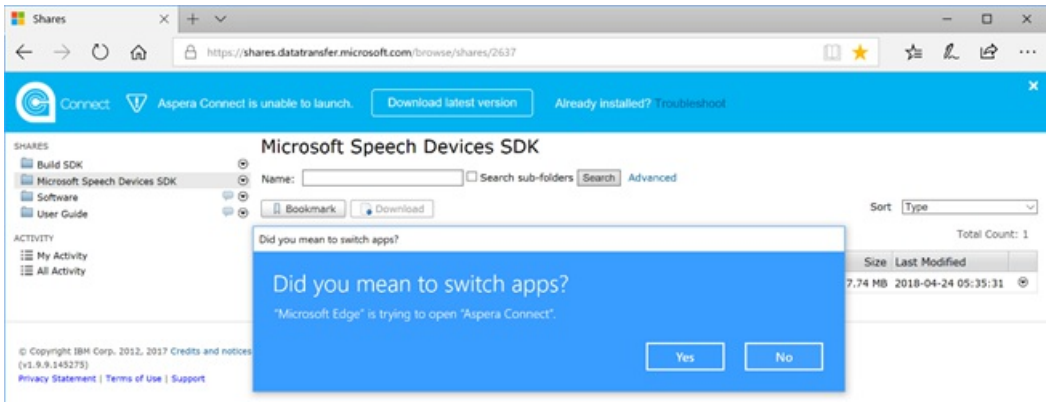


To download the Speech Devices SDK, associated sample code, and reference material:

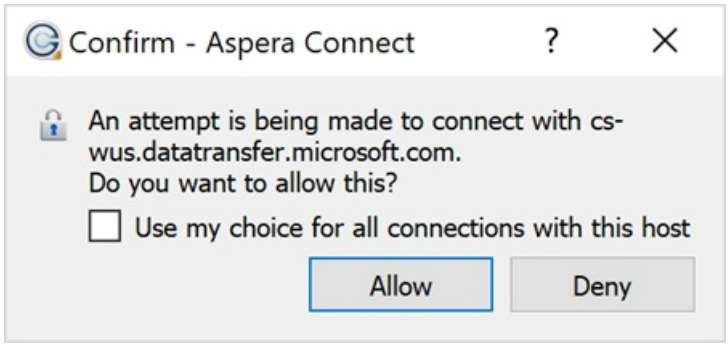
1. Download and install the Aspera Connect tool when prompted in the browser.



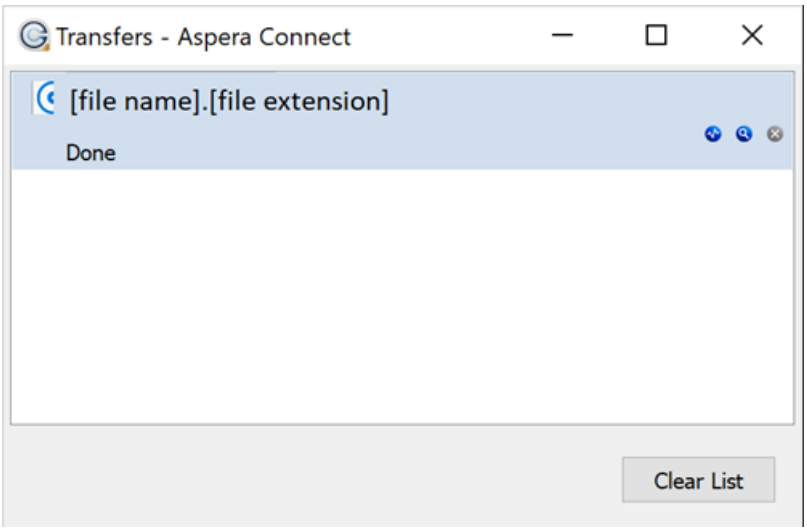
2. Select **Yes** to switch apps to Aspera Connect.



3. Select **Allow** to confirm downloading the files by using Aspera Connect.



4. Close the Aspera Connect Transfers window after the files are downloaded.



By default, the files are downloaded to your **Downloads** folder. You can sign out of this site now.

Next steps

Get started with the Speech Devices SDK

Release notes of Cognitive Services Speech Devices SDK

10/23/2018 • 2 minutes to read • Edit Online

The following sections list changes in the most recent releases.

Cognitive Services Speech Devices SDK 1.0.1: 2018-Oct release

- Updated the Speech SDK component to version 1.0.1. For more information, see its release notes.
- Speech recognition accuracy will be improved with our improved audio processing algorithm
- One continuous recognition audio session bug is fixed.

Breaking changes

- With this release a number of breaking changes are introduced. Please check this page for details relating to the APIs.
- The KWS model files are not compatible with Speech Devices SDK 1.0.1. The existing Wake Word files will be deleted after the new Wake Word files are written to the device.

Cognitive Services 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.

Cognitive Services Speech Devices SDK 0.2.12733: 2018-May release

The first public preview release of the Cognitive Services Speech Devices SDK.

Speech Devices SDK license agreement

10/19/2018 • 8 minutes to read • Edit Online

MICROSOFT SOFTWARE LICENSE TERMS
SPEECH DEVICES SOFTWARE DEVELOPMENT KIT (SDK)

> IF YOU LIVE IN (OR ARE A BUSINESS WITH YOUR PRINCIPAL PLACE OF BUSINESS IN) THE UNITED STATES, PLEASE READ THE "BINDING ARBITRATION AND CLASS ACTION WAIVER" SECTION BELOW. IT AFFECTS HOW DISPUTES ARE RESOLVED.

These license terms are an agreement between you and Microsoft Corporation (or one of its affiliates). They apply to the software named above and any Microsoft services or software updates (except to the extent such services or updates are accompanied by new or additional terms, in which case those different terms apply prospectively and do not alter your or Microsoft's rights relating to pre-updated software or services). **IF YOU COMPLY WITH THESE LICENSE TERMS, YOU HAVE THE RIGHTS BELOW.**

1. INSTALLATION AND USE RIGHTS.

a) General. You may install and use any number of copies of the software on your devices, solely to evaluate and test it for your internal business purposes. You may not use the software in a live operating environment unless Microsoft permits you to do so under another agreement.

b) Included Microsoft Applications. The software may include other Microsoft applications. These license terms apply to those included applications, if any, unless other license terms are provided with the other Microsoft applications.

c) Third Party Components. The software may include third party components with separate legal notices or governed by other agreements, as may be described in the Third Party Notices file(s) accompanying the software.

d) Package Managers. The software may include package managers, like NuGet, that give you the option to download other Microsoft and third-party software packages to use with your application. Those packages are under their own licenses, and not this agreement. Microsoft does not distribute, license or provide any warranties for any of the third-party packages.

e) If you are a direct competitor, and you access or use the software for purposes of competitive benchmarking, analysis, or intelligence gathering, you waive as against Microsoft, its subsidiaries, and its affiliated companies (including prospectively) any competitive use, access, and benchmarking test restrictions in the terms governing your software to the extent your terms of use are, or purport to be, more restrictive than Microsoft's terms. If you do not waive any such purported restrictions in the terms governing your software, you are not allowed to access or use this software, and will not do so.

2. SCOPE OF LICENSE. The software is licensed, not sold. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you will not (and have no right to):

a) work around any technical limitations in the software that only allow you to use it in certain ways;

b) reverse engineer, decompile or disassemble the software;

c) remove, minimize, block, or modify any notices of Microsoft or its suppliers in the software;

d) use the software in any way that is against the law or to create or propagate malware; or

e) share, publish, or lend the software (except for any distributable code, and then subject to the applicable terms

above), provide the software as a stand-alone hosted solution for others to use, or transfer the software or this agreement to any third party.

f) You must use the software only in combination with Microsoft Azure Cognitive Services, Speech service

3. DATA COLLECTION. This software may interact with other Microsoft products that collect data that is transmitted to Microsoft. To learn more about how Microsoft processes personal data we collect, please see the Microsoft Privacy Statement at http://go.microsoft.com/fwlink/?LinkId=248681.

4. PRE-RELEASE SOFTWARE. The software is a pre-release version. It may not operate correctly. It may be different from the commercially released version.

5. FEEDBACK. If you give feedback about the software to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because Microsoft includes your feedback in them. These rights survive this agreement.

6. PERIOD. This agreement is effective on your acceptance and may be terminated by you or Microsoft at any time.

7. TERMINATION. Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with any of its terms or conditions. In such event, you must destroy all copies of the software and all of its component parts.

8.EXPORT RESTRICTIONS. You must comply with all domestic and international export laws and regulations that apply to the software, which include restrictions on destinations, end users, and end use. For further information on export restrictions, visit (aka.ms/exporting).

9. SUPPORT SERVICES. Microsoft is not obligated under this agreement to provide any support services for the software. Any support provided is "as is", "with all faults", and without warranty of any kind.

10. UPDATES. The software may periodically check for updates, and download and install them for you. You may obtain updates only from Microsoft or authorized sources. Microsoft may need to update your system to provide you with updates. You agree to receive these automatic updates without any additional notice. Updates may not include or support all existing software features, services, or peripheral devices.

11. BINDING ARBITRATION AND CLASS ACTION WAIVER. This Section applies if you live in (or, if a business, your principal place of business is in) the United States. If you and Microsoft have a dispute, you and Microsoft agree to try for 60 days to resolve it informally. If you and Microsoft can't, you and Microsoft agree to binding individual arbitration before the American Arbitration Association under the Federal Arbitration Act ("FAA"), and not to sue in court in front of a judge or jury. Instead, a neutral arbitrator will decide. Class action lawsuits, class-wide arbitrations, private attorney-general actions, and any other proceeding where someone acts in a representative capacity are not allowed; nor is combining individual proceedings without the consent of all parties. The complete Arbitration Agreement contains more terms and is at aka.ms/arb-agreement-1. You and Microsoft agree to these terms.

12. ENTIRE AGREEMENT. This agreement, and any other terms Microsoft may provide for supplements, updates, or third-party applications, is the entire agreement for the software.

13. APPLICABLE LAW AND PLACE TO RESOLVE DISPUTES. If you acquired the software in the United States or Canada, the laws of the state or province where you live (or, if a business, where your principal place of business is located) govern the interpretation of this agreement, claims for its breach, and all other claims (including consumer protection, unfair competition, and tort claims), regardless of conflict of laws principles, except that the FAA governs everything related to arbitration. If you acquired the software in any other country, its laws apply, except that the FAA governs everything related to arbitration. If U.S. federal jurisdiction exists, you and Microsoft consent to exclusive jurisdiction and venue in the federal court in King County, Washington for all disputes heard in court (excluding arbitration). If not, you and Microsoft consent to exclusive jurisdiction and

venue in the Superior Court of King County, Washington for all disputes heard in court (excluding arbitration).

14. CONSUMER RIGHTS; REGIONAL VARIATIONS. This agreement describes certain legal rights. You may have other rights, including consumer rights, under the laws of your state, province, or country. Separate and apart from your relationship with Microsoft, you may also have rights with respect to the party from which you acquired the software. This agreement does not change those other rights if the laws of your state, province, or country do not permit it to do so. For example, if you acquired the software in one of the below regions, or mandatory country law applies, then the following provisions apply to you:

a) Australia. You have statutory guarantees under the Australian Consumer Law and nothing in this agreement is intended to affect those rights.

b) Canada. If you acquired this software in Canada, you may stop receiving updates by turning off the automatic update feature, disconnecting your device from the Internet (if and when you re-connect to the Internet, however, the software will resume checking for and installing updates), or uninstalling the software. The product documentation, if any, may also specify how to turn off updates for your specific device or software.

c) Germany and Austria.

> (i) Warranty. The properly licensed software will perform substantially as described in any Microsoft materials that accompany the software. However, Microsoft gives no contractual guarantee in relation to the licensed software.
>
> (ii) Limitation of Liability. In case of intentional conduct, gross negligence, claims based on the Product Liability Act, as well as, in case of death or personal or physical injury, Microsoft is liable according to the statutory law.
>
> Subject to the foregoing clause (ii), Microsoft will only be liable for slight negligence if Microsoft is in breach of such material contractual obligations, the fulfillment of which facilitate the due performance of this agreement, the breach of which would endanger the purpose of this agreement and the compliance with which a party may constantly trust in (so-called "cardinal obligations"). In other cases of slight negligence, Microsoft will not be liable for slight negligence.

15. DISCLAIMER OF WARRANTY. THE SOFTWARE IS LICENSED "AS IS." YOU BEAR THE RISK OF USING IT. MICROSOFT GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. TO THE EXTENT PERMITTED UNDER APPLICABLE LAWS, MICROSOFT EXCLUDES ALL IMPLIED WARRANTIES, INCLUDING MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.

16. LIMITATION ON AND EXCLUSION OF DAMAGES. IF YOU HAVE ANY BASIS FOR RECOVERING DAMAGES DESPITE THE PRECEDING DISCLAIMER OF WARRANTY, YOU CAN RECOVER FROM MICROSOFT AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO U.S. $5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.

This limitation applies to (a) anything related to the software, services, content (including code) on third party Internet sites, or third-party applications; and (b) claims for breach of contract, warranty, guarantee, or condition; strict liability, negligence, or other tort; or any other claim; in each case to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your state, province, or country may not allow the exclusion or limitation of incidental, consequential, or other damages.

Third-party software

10/30/2018 • 18 minutes to read • Edit Online

Do not translate or localize

This file provides information regarding components that are being relicensed to you by Microsoft under Microsoft's software licensing terms. Microsoft reserves all rights not expressly granted herein, whether by implication, estoppel or otherwise.

Microsoft is offering you a license to use the following components with Speech Devices SDK subject to the terms of the Microsoft software license terms for Speech Devices SDK products (the "Microsoft Program").

1. kgabis/parson version b87a27c
2. Azure/azure-c-shared-utility version ed84cdb
3. catchorg/Catch2 version d2d8455
4. curl/curl version 6d7d0eb
5. curl/curl version 7.21.3
6. openssl/openssl version b2758a2
7. nlohmann/json version d2dd27d
8. tinyalsa/tinyalsa version df11091
9. xianyi/OpenBLAS version 5f998ef
10. openfst version 1.6.5
11. kaldi-asr/kaldi version eba50e4

1. kgabis/parson

MIT License

Copyright (c) 2012 - 2017 Krzysztof Gabis

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2. Azure/azure-c-shared-utility

Microsoft Azure IoT SDKs Copyright (c) Microsoft Corporation All rights reserved. MIT License Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ""Software""), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED *AS IS*, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3. catchorg/Catch2

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4. curl/curl

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1996 - 2018, Daniel Stenberg, daniel@haxx.se, and many contributors, see the THANKS file.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

5. curl/curl

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1996 - 2010, Daniel Stenberg, daniel@haxx.se.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

6. openssl/openssl

The OpenSSL toolkit stays under a double license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts.

```
OpenSSL License

/* ====================
* Copyright (c) 1998-2017 The OpenSSL Project.  All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. All advertising materials mentioning features or use of this
*    software must display the following acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
*
* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
*    endorse or promote products derived from this software without
*    prior written permission. For written permission, please contact
*    openssl-core@openssl.org.
*
* 5. Products derived from this software may not be called "OpenSSL"
*    nor may "OpenSSL" appear in their names without prior written
*    permission of the OpenSSL Project.
*
* 6. Redistributions of any form whatsoever must retain the following
*    acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
```

```
* OF THE POSSIBILITY OF SUCH DAMAGE.
* ==========================
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com).  This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/
```

Original SSLeay License

```
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to.  The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code.  The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*    must display the following acknowledgement:
*    "This product includes cryptographic software written by
*     Eric Young (eay@cryptsoft.com)"
*    The word 'cryptographic' can be left out if the rouines from the library
*    being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
*    the apps directory (application code) you must include an acknowledgement:
*    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publicly available version or
* derivative of this code cannot be changed.  i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

7. nlohmann/json

MIT License

Copyright (c) 2013-2018 Niels Lohmann

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

range-v3 Boost Software License - Version 1.0 - August 17th, 2003

Copyright Eric Niebler 2013-2014

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Files with sample code from IETF RFC 7049

Copyright (c) 2013 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Internet Society, IETF or IETF Trust, nor the names of specific contributors, may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY

EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8. tinyalsa/tinyalsa

Copyright 2011, The Android Open Source Project Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Android Open Source Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY The Android Open Source Project ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL The Android Open Source Project BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9. xianyi/OpenBLAS version #1e729d7ba20cee417259934da9424b50e2d41548

OpenBLAS

Copyright (c) 2011-2014, The OpenBLAS Project All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the OpenBLAS project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF

THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

GotoBLAS2

Copyright 2009, 2010 The University of Texas at Austin. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE UNIVERSITY OF TEXAS AT AUSTIN ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE UNIVERSITY OF TEXAS AT AUSTIN OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of The University of Texas at Austin.

File with code "adapted from" Google performance tools

- Copyright (c) 2007, Google Inc.
  - All rights reserved.
  -
  - Redistribution and use in source and binary forms, with or without
  - modification, are permitted provided that the following conditions are
  - met:
  -
  - - Redistributions of source code must retain the above copyright
  - notice, this list of conditions and the following disclaimer.
  - - Redistributions in binary form must reproduce the above
  - copyright notice, this list of conditions and the following disclaimer
  - in the documentation and/or other materials provided with the
  - distribution.
  - - Neither the name of Google Inc. nor the names of its
  - contributors may be used to endorse or promote products derived from
  - this software without specific prior written permission.
  -
  - THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  - "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  - LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
  - A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
  - OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  - SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

- LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
- DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
- THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
- (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
- OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. *
- ---
- Author: Craig Silverstein */

10. openfst

Licensed under the Apache License, Version 2.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at

```
http://www.apache.org/licenses/LICENSE-2.0
```

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Copyright 2005-2015 Google, Inc.

11. kaldi-asr/kaldi

Copyright (c) kaldi-asr contributors

Licensed under the Apache License, Version 2.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at

```
http://www.apache.org/licenses/LICENSE-2.0
```

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Troubleshoot the Speech Service SDK

10/19/2018 • 2 minutes to read • Edit Online

This article provides information to help you solve issues you might encounter when you use the Speech Service 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"
```

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
```

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

Support and help options

10/19/2018 • 2 minutes to read • Edit Online

Are you just starting to explore the functionality of the Speech service? Are you implementing a new feature to your application? Here are suggestions about where you can get help as a developer.

- Stay informed about new developments in *Azure Cognitive Services*, or find the latest news related to *Speech service*.
- Search to see if your issue was discussed by the community, or if existing documentation for the feature you want to implement already exists.
- If you can't find a satisfactory answer, ask a question on *Stack Overflow*.
- If you find an issue with one of the samples on GitHub, raise a *GitHub* issue.
- Search for a solution in the *UserVoice forum*.

Stay informed

News about Cognitive Services is collected in the Cognitive Services blog. For the latest information about Speech service, track the Speech service blog.

Search

You might find the answer you need in the documentation, the samples, or answers to Stack Overflow questions or in the samples.

Scoped Search

For faster results, scope your search to Stack Overflow, the documentation, and code samples by using the following query on your favorite search engine:

```
{Your Search Terms} (site:stackoverflow.com OR site:docs.microsoft.com OR site:github.com/azure-samples)
```

Where *{Your Search Terms}* is your search keywords.

Create an Azure support request

Azure customers can create and manage support requests in the Azure portal.

- Azure Portal
- Azure portal for the United States government

Post a question to Stack Overflow

Stack Overflow is the preferred channel for development-related questions. It's where members of the community and Microsoft team members are directly involved in helping you solve your problems.

If you can't find an answer to your problem via search, submit a new question to Stack Overflow. Use one of the following tags when you formulate the question:

COMPONENT/AREA	TAGS
Speech Recognition	[microsoft-cognitive+speech-to-text]
Speech Synthesis	[microsoft-cognitive+text-to-speech]
Speech Translation	[microsoft-cognitive+translation]
Speech Intent	[microsoft-cognitive+luis]
General Speech SDK	[microsoft-cognitive+microsoft-speech-api]

TIP

The following posts from Stack Overflow contain tips on how to form questions and add source code. Following these guidelines might help increase the chances that community members assess and respond to your question quickly:

- How do I ask a good question?
- How to create a Minimal, Complete, and Verifiable example

Create a GitHub issue

Samples are often posted as open source. For questions and issues, create an *issue* in the respective GitHub repository. You can submit a pull request, too. The following list contains links to the sample repositories:

- Speech SDK
- Devices SDK

You can create a bug report, feature request, or ask an general question and share best practices. For bug reports, please follow the provided template:

Describe the bug

A clear and concise description of what the bug is.

To Reproduce

Steps to reproduce the behavior:

1. ...
2. ...

Expected behavior

A clear and concise description of what you expected to happen.

Version of the Cognitive Services Speech SDK

Which version of the SDK are you using.

Platform, Operating System, and Programming Language

- OS: [e.g. Windows, Linux, Android, iOS, ...] - please be specific
- Hardware - x64, x86, ARM, ...
- Browser [e.g. Chrome, Safari] (if applicable)- please be specific

Additional context

- Error messages, log information, stack trace, ...
- If you report an error for a specific service interaction, please report the SessionId and time (incl. timezone) of the reported incidents. The SessionId is reported in all call-backs/events you receive.
- Any other additional information

UserVoice forum

Share your ideas for making Cognitive Services and the accompanying APIs work better for the applications you develop. Use our growing Knowledge Base to find answers to common questions:

UserVoice