

Recognize and synthesize speech

Azure AI Speech is a service that provides speech-related functionality, including:

- A *speech-to-text* API that enables you to implement speech recognition (converting audible spoken words into text).
- A *text-to-speech* API that enables you to implement speech synthesis (converting text into audible speech).

In this exercise, you'll use both of these APIs to implement a speaking clock application.

NOTE This exercise requires that you are using a computer with speakers/headphones. For the best experience, a microphone is also required. Some hosted virtual environments may be able to capture audio from your local microphone, but if this doesn't work (or you don't have a microphone at all), you can use a provided audio file for speech input. Follow the instructions carefully, as you'll need to choose different options depending on whether you are using a microphone or the audio file.

Provision an *Azure AI Speech* resource

If you don't already have one in your subscription, you'll need to provision an **Azure AI Speech** resource.

1. Open the Azure portal at <https://portal.azure.com>, and sign in using the Microsoft account associated with your Azure subscription.
2. In the search field at the top, search for **Azure AI services** and press **Enter**, then select **Create** under **Speech service** in the results.
3. Create a resource with the following settings:
 - **Subscription:** *Your Azure subscription*
 - **Resource group:** *Choose or create a resource group*
 - **Region:** *Choose any available region*
 - **Name:** *Enter a unique name*
 - **Pricing tier:** Select **F0** (*free*), or **S** (*standard*) if F is not available.
 - **Responsible AI Notice:** Agree.
4. Select **Review + create**.
5. Wait for deployment to complete, and then go to the deployed resource.
6. View the **Keys and Endpoint** page. You will need the information on this page later in the exercise.

Prepare to develop an app in Visual Studio Code

You'll develop your speech app using Visual Studio Code. The code files for your app have been provided in a GitHub repo.

Tip: If you have already cloned the **mslearn-ai-language** repo, open it in Visual Studio code. Otherwise, follow these steps to clone it to your development environment.

1. Start Visual Studio Code.
2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the <https://github.com/MicrosoftLearning/mslearn-ai-language> repository to a local folder (it doesn't matter which folder).
3. When the repository has been cloned, open the folder in Visual Studio Code.
4. Wait while additional files are installed to support the C# code projects in the repo.

Note: If you are prompted to add required assets to build and debug, select **Not Now**.

Configure your application

Applications for both C# and Python have been provided. Both apps feature the same functionality. First, you'll complete some key parts of the application to enable it to use your Azure AI Speech resource.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/07-speech** folder and expand the **CSharp** or **Python** folder depending on your language preference and the **speaking-clock** folder it contains. Each folder contains the language-specific code files for an app into which you're going to integrate Azure AI Speech functionality.
2. Right-click the **speaking-clock** folder containing your code files and open an integrated terminal. Then install the Azure AI Speech SDK package by running the appropriate command for your language preference:

C#

CodeCopy

```
dotnet add package Microsoft.CognitiveServices.Speech --version 1.30.0
```

Python

CodeCopy

```
pip install azure-cognitiveservices-speech==1.30.0
```

3. In the **Explorer** pane, in the **speaking-clock** folder, open the configuration file for your preferred language
 - **C#:** appsettings.json

- **Python:** .env
- 4. Update the configuration values to include the **region** and a **key** from the Azure AI Speech resource you created (available on the **Keys and Endpoint** page for your Azure AI Speech resource in the Azure portal).

NOTE: Be sure to add the *region* for your resource, not the endpoint!

- 5. Save the configuration file.

Add code to use the Azure AI Speech SDK

- 1. Note that the **speaking-clock** folder contains a code file for the client application:
 - **C#:** Program.cs
 - **Python:** speaking-clock.py

Open the code file and at the top, under the existing namespace references, find the comment **Import namespaces**. Then, under this comment, add the following language-specific code to import the namespaces you will need to use the Azure AI Speech SDK:

C#: Program.cs

C#Copy

```
// Import namespaces
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
```

Python: speaking-clock.py

CodeCopy

```
# Import namespaces
import azure.cognitiveservices.speech as speech_sdk
```

- 2. In the **Main** function, note that code to load the service key and region from the configuration file has already been provided. You must use these variables to create a **SpeechConfig** for your Azure AI Speech resource. Add the following code under the comment **Configure speech service**:

C#: Program.cs

C#Copy

```
// Configure speech service
speechConfig = SpeechConfig.FromSubscription(aiSvcKey, aiSvcRegion);
Console.WriteLine("Ready to use speech service in " +
speechConfig.Region);

// Configure voice
speechConfig.SpeechSynthesisVoiceName = "en-US-AriaNeural";
```

Python: speaking-clock.py

CodeCopy

```
# Configure speech service
speech_config = speech_sdk.SpeechConfig(ai_key, ai_region)
print('Ready to use speech service in:', speech_config.region)
```

3. Save your changes and return to the integrated terminal for the **speaking-clock** folder, and enter the following command to run the program:

C#

CodeCopy

```
dotnet run
```

Python

CodeCopy

```
python speaking-clock.py
```

4. If you are using C#, you can ignore any warnings about using the **await** operator in asynchronous methods - we'll fix that later. The code should display the region of the speech service resource the application will use.

Add code to recognize speech

Now that you have a **SpeechConfig** for the speech service in your Azure AI Speech resource, you can use the **Speech-to-text** API to recognize speech and transcribe it to text.

IMPORTANT: This section includes instructions for two alternative procedures. Follow the first procedure if you have a working microphone. Follow the second procedure if you want to simulate spoken input by using an audio file.

If you have a working microphone

1. In the **Main** function for your program, note that the code uses the **TranscribeCommand** function to accept spoken input.
2. In the **TranscribeCommand** function, under the comment **Configure speech recognition**, add the appropriate code below to create a **SpeechRecognizer** client that can be used to recognize and transcribe speech using the default system microphone:

C#

C#Copy

```
// Configure speech recognition
using AudioConfig audioConfig = AudioConfig.FromDefaultMicrophoneInput();
using SpeechRecognizer speechRecognizer = new
SpeechRecognizer(speechConfig, audioConfig);
Console.WriteLine("Speak now...");
```

Python

CodeCopy

```
# Configure speech recognition
audio_config = speech_sdk.AudioConfig(use_default_microphone=True)
speech_recognizer = speech_sdk.SpeechRecognizer(speech_config,
audio_config)
print('Speak now...')
```

3. Now skip ahead to the **Add code to process the transcribed command** section below.

Alternatively, use audio input from a file

1. In the terminal window, enter the following command to install a library that you can use to play the audio file:

C#

CodeCopy

```
dotnet add package System.Windows.Extensions --version 4.6.0
```

Python

CodeCopy

```
pip install playsound==1.3.0
```

2. In the code file for your program, under the existing namespace imports, add the following code to import the library you just installed:

C#: Program.cs

C#Copy

```
using System.Media;
```

Python: speaking-clock.py

CodeCopy

```
from playsound import playsound
```

3. In the **Main** function, note that the code uses the **TranscribeCommand** function to accept spoken input. Then in the **TranscribeCommand** function, under the comment **Configure speech recognition**, add the appropriate code below to create a **SpeechRecognizer** client that can be used to recognize and transcribe speech from an audio file:

C#: Program.cs

C#Copy

```
// Configure speech recognition
string audioFile = "time.wav";
SoundPlayer wavPlayer = new SoundPlayer(audioFile);
wavPlayer.Play();
using AudioConfig audioConfig = AudioConfig.FromWavFileInput(audioFile);
using SpeechRecognizer speechRecognizer = new
SpeechRecognizer(speechConfig, audioConfig);
```

Python: speaking-clock.py

CodeCopy

```
# Configure speech recognition
current_dir = os.getcwd()
audioFile = current_dir + '\\time.wav'
playsound(audioFile)
audio_config = speech_sdk.AudioConfig(filename=audioFile)
speech_recognizer = speech_sdk.SpeechRecognizer(speech_config,
audio_config)
```

Add code to process the transcribed command

1. In the **TranscribeCommand** function, under the comment **Process speech input**, add the following code to listen for spoken input, being careful not to replace the code at the end of the function that returns the command:

C#: Program.cs

C#Copy

```
// Process speech input
SpeechRecognitionResult speech = await
speechRecognizer.RecognizeOnceAsync();
if (speech.Reason == ResultReason.RecognizedSpeech)
{
    command = speech.Text;
    Console.WriteLine(command);
}
else
{
    Console.WriteLine(speech.Reason);
    if (speech.Reason == ResultReason.Canceled)
    {
        var cancellation = CancellationDetails.FromResult(speech);
        Console.WriteLine(cancellation.Reason);
    }
}
```

```
        Console.WriteLine(cancellation.ErrorDetails);
    }
}
```

Python: speaking-clock.py

CodeCopy

```
# Process speech input
speech = speech_recognizer.recognize_once_async().get()
if speech.reason == speech_sdk.ResultReason.RecognizedSpeech:
    command = speech.text
    print(command)
else:
    print(speech.reason)
    if speech.reason == speech_sdk.ResultReason.Canceled:
        cancellation = speech.cancellation_details
        print(cancellation.reason)
        print(cancellation.error_details)
```

2. Save your changes and return to the integrated terminal for the **speaking-clock** folder, and enter the following command to run the program:

C#

CodeCopy

```
dotnet run
```

Python

CodeCopy

```
python speaking-clock.py
```

3. If using a microphone, speak clearly and say "what time is it?". The program should transcribe your spoken input and display the time (based on the local time of the computer where the code is running, which may not be the correct time where you are).

The SpeechRecognizer gives you around 5 seconds to speak. If it detects no spoken input, it produces a "No match" result.

If the `SpeechRecognizer` encounters an error, it produces a result of "Cancelled". The code in the application will then display the error message. The most likely cause is an incorrect key or region in the configuration file.

Synthesize speech

Your speaking clock application accepts spoken input, but it doesn't actually speak! Let's fix that by adding code to synthesize speech.

1. In the **Main** function for your program, note that the code uses the **TellTime** function to tell the user the current time.
2. In the **TellTime** function, under the comment **Configure speech synthesis**, add the following code to create a **SpeechSynthesizer** client that can be used to generate spoken output:

C#: Program.cs

C#Copy

```
// Configure speech synthesis
speechConfig.SpeechSynthesisVoiceName = "en-GB-RyanNeural";
using SpeechSynthesizer speechSynthesizer = new
SpeechSynthesizer(speechConfig);
```

Python: speaking-clock.py

CodeCopy

```
# Configure speech synthesis
speech_config.speech_synthesis_voice_name = "en-GB-RyanNeural"
speech_synthesizer = speech_sdk.SpeechSynthesizer(speech_config)
```

NOTE The default audio configuration uses the default system audio device for output, so you don't need to explicitly provide an **AudioConfig**. If you need to redirect audio output to a file, you can use an **AudioConfig** with a filepath to do so.

3. In the **TellTime** function, under the comment **Synthesize spoken output**, add the following code to generate spoken output, being careful not to replace the code at the end of the function that prints the response:

C#: Program.cs

C#Copy

```
// Synthesize spoken output
SpeechSynthesisResult speak = await
speechSynthesizer.SpeakTextAsync(responseText);
if (speak.Reason != ResultReason.SynthesizingAudioCompleted)
{
    Console.WriteLine(speak.Reason);
}
```

Python: speaking-clock.py

CodeCopy

```
# Synthesize spoken output
speak = speech_synthesizer.speak_text_async(response_text).get()
if speak.reason != speech_sdk.ResultReason.SynthesizingAudioCompleted:
    print(speak.reason)
```

4. Save your changes and return to the integrated terminal for the **speaking-clock** folder, and enter the following command to run the program:

C#

CodeCopy

```
dotnet run
```

Python

CodeCopy

```
python speaking-clock.py
```

5. When prompted, speak clearly into the microphone and say “what time is it?”. The program should speak, telling you the time.

Use a different voice

Your speaking clock application uses a default voice, which you can change. The Speech service supports a range of *standard* voices as well as more human-like *neural* voices. You can also create *custom* voices.

Note: For a list of neural and standard voices, see [Voice Gallery](#) in the Speech Studio.

1. In the **TellTime** function, under the comment **Configure speech synthesis**, modify the code as follows to specify an alternative voice before creating the **SpeechSynthesizer** client:

C#: Program.cs

C#Copy

```
// Configure speech synthesis
speechConfig.SpeechSynthesisVoiceName = "en-GB-LibbyNeural"; // change
this
using SpeechSynthesizer speechSynthesizer = new
SpeechSynthesizer(speechConfig);
```

Python: speaking-clock.py

CodeCopy

```
# Configure speech synthesis
speech_config.speech_synthesis_voice_name = 'en-GB-LibbyNeural' # change
this
speech_synthesizer = speech_sdk.SpeechSynthesizer(speech_config)
```

2. Save your changes and return to the integrated terminal for the **speaking-clock** folder, and enter the following command to run the program:

C#

CodeCopy

```
dotnet run
```

Python

CodeCopy

```
python speaking-clock.py
```

3. When prompted, speak clearly into the microphone and say "what time is it?". The program should speak in the specified voice, telling you the time.

Use Speech Synthesis Markup Language

Speech Synthesis Markup Language (SSML) enables you to customize the way your speech is synthesized using an XML-based format.

1. In the **TellTime** function, replace all of the current code under the comment **Synthesize spoken output** with the following code (leave the code under the comment **Print the response**):

C#: Program.cs

C#Copy

```
// Synthesize spoken output
string responseSsml = $"
    <speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis'
xml:lang='en-US'>
        <voice name='en-GB-LibbyNeural'>
            {responseText}
            <break strength='weak' />
            Time to end this lab!
        </voice>
    </speak>";
SpeechSynthesisResult speak = await
speechSynthesizer.SpeakSsmlAsync(responseSsml);
if (speak.Reason != ResultReason.SynthesizingAudioCompleted)
{
    Console.WriteLine(speak.Reason);
}
```

Python: speaking-clock.py

CodeCopy

```
# Synthesize spoken output
responseSsml = " \
    <speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis'
xml:lang='en-US'> \
        <voice name='en-GB-LibbyNeural'> \
            {} \
            <break strength='weak' /> \
            Time to end this lab! \
        </voice> \
    </speak>".format(response_text)
speak = speech_synthesizer.speak_ssml_async(responseSsml).get()
if speak.reason != speech_sdk.ResultReason.SynthesizingAudioCompleted:
    print(speak.reason)
```

2. Save your changes and return to the integrated terminal for the **speaking-clock** folder, and enter the following command to run the program:

C#

CodeCopy

```
dotnet run
```

Python

CodeCopy

```
python speaking-clock.py
```

3. When prompted, speak clearly into the microphone and say “what time is it?”. The program should speak in the voice that is specified in the SSML (overriding the voice specified in the SpeechConfig), telling you the time, and then after a pause, telling you it’s time to end this lab - which it is!