3.7 - Create speech-enabled apps with Azure AI services

Overview

The Azure AI Speech service enables you to build speech-enabled applications. This module focuses on using the speech-to-text and text to speech APIs, which enable you to create apps that are capable of speech recognition and speech synthesis.

Learning objectives

In this module, you'll learn how to:

- Provision an Azure resource for the Azure AI Speech service
- Use the Azure AI Speech to text API to implement speech recognition
- Use the Text to speech API to implement speech synthesis
- Configure audio format and voices

- Use Speech Synthesis Markup Language (SSML)

Introduction

Azure AI Speech provides APIs that you can use to build speech-enabled applications. This includes:

- **Speech to text**: An API that enables *speech recognition* in which your application can accept spoken input.
- **Text to speech**: An API that enables *speech synthesis* in which your application can provide spoken output.
- **Speech Translation**: An API that you can use to translate spoken input into multiple languages.
- **Speaker Recognition**: An API that enables your application to recognize individual speakers based on their voice.
- **Intent Recognition**: An API that uses conversational language understanding to determine the semantic meaning of spoken input.

This module focuses on speech recognition and speech synthesis, which are core capabilities of any speech-enabled application.

In this module, you'll learn how to:

- Provision an Azure resource for the Azure AI Speech service
- Use the Speech to text API to implement speech recognition
- Use the Text to speech API to implement speech synthesis
- Configure audio format and voices
- Use Speech Synthesis Markup Language (SSML)

The units in the module include important conceptual information about Azure AI Speech and how to use its API through one of the supported software development kits (SDKs), after which you'll be able to try Azure AI Speech for yourself in a hands-on exercise. To complete the hands-on exercise, you will need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.com/free.

Provision an Azure resource for speech

Before you can use Azure AI Speech, you need to create an Azure AI Speech resource in your Azure subscription. You can use either a dedicated Azure AI Speech resource or a multi-service Azure AI Services resource.

After you create your resource, you'll need the following information to use it from a client application through one of the supported SDKs:

- The *location* in which the resource is deployed (for example, *eastus*)
- One of the *keys* assigned to your resource.

You can view of these values on the **Keys and Endpoint** page for your resource in the Azure portal.

Use the Azure AI Speech to Text API

The Azure AI Speech service supports speech recognition through two REST APIs:

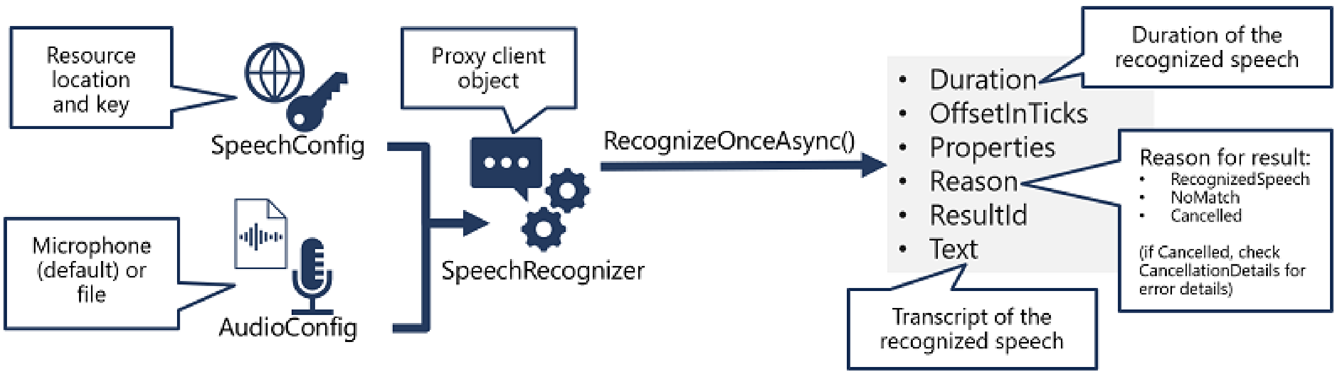
- The **Speech to text** API, which is the primary way to perform speech recognition.
- The **Speech to text Short Audio** API, which is **optimized for short streams of audio (up to 60 seconds)**.

You can use either API for interactive speech recognition, depending on the expected length of the spoken input. You can also use the **Speech to text** API for *batch transcription*, transcribing multiple audio files to text as a batch operation.

You can learn more about the REST APIs in the Speech to text REST API documentation. In practice, most interactive speech-enabled applications use the Speech service through a (programming) language-specific SDK.

Using the Azure AI Speech SDK

While the specific details vary, depending on the SDK being used (Python, C#, and so on); there's a consistent pattern for using the **Speech to text** API:


Resource location and key
SpeechConfig
Microphone (default) or file
AudioConfig
Proxy client object
SpeechRecognizer
RecognizeOnceAsync()
Duration
OffsetInTicks
Properties
Reason
ResultId
Text
Duration of the recognized speech
Reason for result:
RecognizedSpeech
NoMatch
Cancelled
(if Cancelled, check CancellationDetails for error details)
Transcript of the recognized speech

1. Use a **SpeechConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location** and **key**.
2. Optionally, use an **AudioConfig** to define the input source for the audio to be transcribed. By default, this is the default system microphone, but you can also specify an audio file.
3. Use the **SpeechConfig** and **AudioConfig** to create a **SpeechRecognizer** object. This object is a **proxy client** for the **Speech to text** API.
4. Use the methods of the **SpeechRecognizer** object to call the underlying API functions. For example, the **RecognizeOnceAsync()** method uses the Azure AI Speech service to asynchronously transcribe a single spoken utterance.
5. Process the response from the Azure AI Speech service. In the case of the **RecognizeOnceAsync()** method, the result is a **SpeechRecognitionResult** object that includes the following properties:
    - Duration
    - OffsetInTicks
    - Properties
    - Reason
    - ResultId
    - Text

If the operation was successful, the `Reason` property has the enumerated value **RecognizedSpeech**, and the `Text` **property contains the transcription**.

> The `Reason` property provides information about why the recognition result is what it is (i.e., information about the outcome of the recognition operation.

Other possible values for **Result** include **NoMatch** (indicating that the audio was successfully parsed but no speech was recognized) or **Canceled**, indicating that an error occurred (in which case, you can check the **Properties** collection for the **CancellationReason** property to determine what went wrong).

Use the text to speech API

Similarly to its **Speech to text** APIs, the Azure AI Speech service offers other REST APIs for speech synthesis:

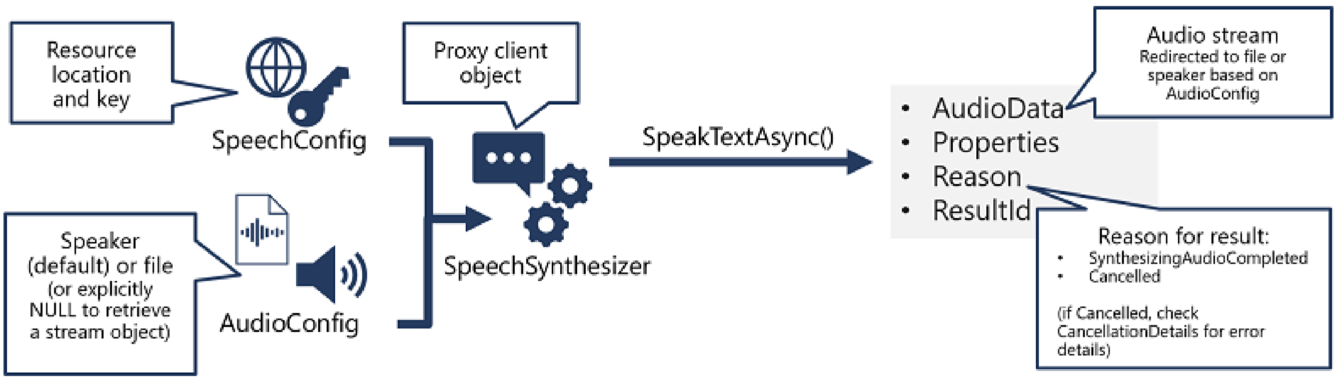
- The **Text to speech** API, which is the primary way to perform speech synthesis.
- The **Batch synthesis** API, which is designed to support batch operations that convert large volumes of text to audio - for example to generate an audio-book from the source text.

You can learn more about the REST APIs in the Text to speech REST API documentation. In practice, most interactive speech-enabled applications use the Azure AI Speech service through a (programming) language-specific SDK.

Using the Azure AI Speech SDK

As with speech recognition, in practice most interactive speech-enabled applications are built using the Azure AI Speech SDK.

The pattern for implementing speech synthesis is similar to that of speech recognition:


Resource location and key
SpeechConfig
Proxy client object
SpeakTextAsync()
Audio stream
Redirected to file or speaker based on AudioConfig
AudioData
Properties
Reason
ResultId
Speaker (default) or file (or explicitly NULL to retrieve a stream object)
AudioConfig
SpeechSynthesizer
Reason for result:
SynthesizingAudioCompleted
Cancelled
(if Cancelled, check CancellationDetails for error details)

1. Use a **SpeechConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location** and **key**.
2. Optionally, use an **AudioConfig** to define the output device for the speech to be synthesized. By default, this is the default system speaker, but you can also specify an audio file, or by explicitly setting this value to a **null value, you can process the audio stream object that is returned directly**.
3. Use the **SpeechConfig** and **AudioConfig** to create a **SpeechSynthesizer** object. This object is a proxy client for the **Text to speech** API.
4. Use the methods of the **SpeechSynthesizer** object to call the underlying API functions. For example, the **SpeakTextAsync()** method uses the Azure AI Speech service to convert text to spoken audio.

> `# Sample code`
> `synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config)`
> `result = synthesizer.speak_text_async(text_to_speak).get()`

5. Process the response from the Azure AI Speech service. In the case of the **SpeakTextAsync** method, the result is a **SpeechSynthesisResult** object that contains the following properties:
   - AudioData
   - Properties
   - Reason
   - ResultId

When speech has been successfully synthesized, the **Reason** property is set to the **SynthesizingAudioCompleted** enumeration and the **AudioData** property contains the audio stream (which, depending on the **AudioConfig** may have been automatically sent to a speaker or file).

Configure audio format and voices

When synthesizing speech, you can use a **SpeechConfig** object to customize the audio that is returned by the Azure AI Speech service.

Audio format

The Azure AI Speech service **supports multiple output formats for the audio stream** that is generated by speech synthesis. Depending on your specific needs, you can choose a format based on the required:

- Audio file type
- Sample-rate
- Bit-depth

The supported formats are indicated in the SDK using the **SpeechSynthesisOutputFormat** enumeration. For example, `SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm`.

To specify the required output format, use the **SetSpeechSynthesisOutputFormat** method of the **SpeechConfig** object:

```
speechConfig.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm);
```

For a full list of supported formats and their enumeration values, see the [Azure AI Speech SDK documentation](#).

Voices

The Azure AI Speech service provides multiple voices that you can use to personalize your speech-enabled applications. There are two kinds of voice that you can use:

- *Standard voices* - synthetic voices created from audio samples.
- *Neural voices* - more natural sounding voices created using deep neural networks.

Voices are identified by names that indicate a locale and a person's name - for example `en-GB-George`.

To specify a voice for speech synthesis in the **SpeechConfig**, set its **SpeechSynthesisVoiceName** property to the voice you want to use:

C#

```
speechConfig.SpeechSynthesisVoiceName = "en-GB-George";
```

For information about voices, see the Azure AI Speech SDK documentation.

Use Speech Synthesis Markup Language

While the Azure AI Speech SDK enables you to submit plain text to be synthesized into speech (for example, by using the **SpeakTextAsync()** method), the service also supports an **XML-based syntax for describing characteristics of the speech you want to generate**. This **Speech Synthesis Markup Language** (SSML) syntax offers greater control over how the spoken output sounds, enabling you to:

- Specify a speaking style, such as "excited" or "cheerful" when using a neural voice.
- Insert pauses or silence.
- Specify *phonemes* (phonetic pronunciations), for example to pronounce the text "SQL" as "sequel".
- Adjust the *prosody* of the voice (affecting the pitch, timbre, and speaking rate).
- Use common "say-as" rules, for example to specify that a given string should be expressed as a date, time, telephone number, or other form.
- Insert recorded speech or audio, for example to include a standard recorded message or simulate background noise.

For example, consider the following SSML:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
                     xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
    <voice name="en-US-AriaNeural">
        <mstts:express-as style="cheerful">
          I say tomato
        </mstts:express-as>
    </voice>
    <voice name="en-US-GuyNeural">
        I say <phoneme alphabet="sapi" ph="t ao m ae t ow"> tomato </phoneme>.
        <break strength="weak"/>Lets call the whole thing off!
    </voice>
</speak>
```

This SSML specifies a spoken dialog between two different neural voices, like this:

- **Ariana** (*cheerfully*): "I say tomato:
- **Guy**: "I say tomato (pronounced *tom-ah-toe*) ... Let's call the whole thing off!"

To submit an SSML description to the Speech service, you can use the **SpeakSsmlAsync()** method, like this:

C#

```
speechSynthesizer.SpeakSsmlAsync(ssml_string);
```

For more information about SSML, see the Azure AI Speech SDK documentation.

Exercise - Create a speech-enabled app

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.

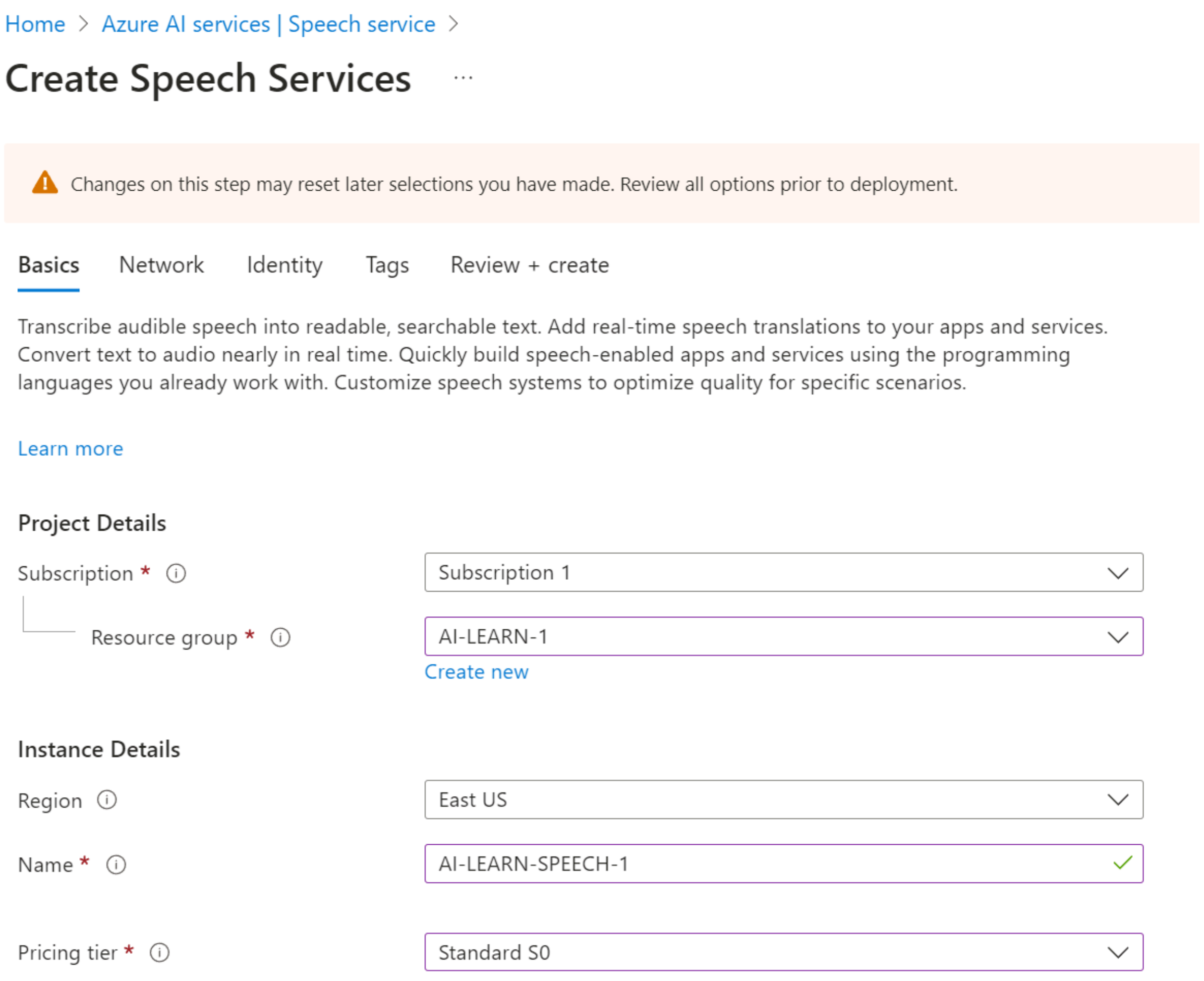
> **NOTE** This exercise requires that you are using a computer with speakers/headphones. For the best experience, a microphone is also required. Some hosted virtual environments may be able to capture audio from your local microphone, but if this doesn't work (or you don't have a microphone at all), 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.


Home > Azure AI services | Speech service >

Create Speech Services ...

Changes on this step may reset later selections you have made. Review all options prior to deployment.

Basics | Network | Identity | Tags | Review + create

Transcribe audible speech into readable, searchable text. Add real-time speech translations to your apps and services. Convert text to audio nearly in real time. Quickly build speech-enabled apps and services using the programming languages you already work with. Customize speech systems to optimize quality for specific scenarios.

Learn more

Project Details

Subscription *	Subscription 1
Resource group *	AI-LEARN-1
	Create new

Instance Details

Region	East US
Name *	AI-LEARN-SPEECH-1
Pricing tier *	Standard S0

4. Select **Review + create**, then select **Create** to provision the resource.
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.

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 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#

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

   Python

```
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 (View here: 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
   `// Import namespaces using Microsoft.CognitiveServices.Speech; using Microsoft.CognitiveServices.Speech.Audio;`

   Python: speaking-clock.py (View here: speaking-clock.py)
   `# 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
   `// 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 (View here: speaking-clock.py)
   `# 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#

   ```
   dotnet run
   ```

Python

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

   Python
   ```
   # 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#

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

   Python

```
pip install playsound==1.2.2
```

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
   `using System.Media;`

   Python: speaking-clock.py (View here: speaking-clock.py)
   `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
   `// 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 (View here: speaking-clock.py)
   `# 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
   `// 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 (View here: speaking-clock.py)
   `# 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#

   ```
   dotnet run
   ```

   Python

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

   Python: speaking-clock.py (View here: speaking-clock.py)
   `# 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
   `// Synthesize spoken output SpeechSynthesisResult speak = await`

`speechSynthesizer.SpeakTextAsync(responseText); if (speak.Reason != ResultReason.SynthesizingAudioCompleted) { Console.WriteLine(speak.Reason); }`

Python: speaking-clock.py (View here: [speaking-clock.py]())
`# 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#

   ```
   dotnet run
   ```

   Python

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

   Python: speaking-clock.py (View here: [speaking-clock.py]())
   `# 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#

   ```
   dotnet run
   ```

   Python

```
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
   ```
   // 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 (View here: speaking-clock.py)
   ```
   # 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#

   ```
   dotnet run
   ```

   Python

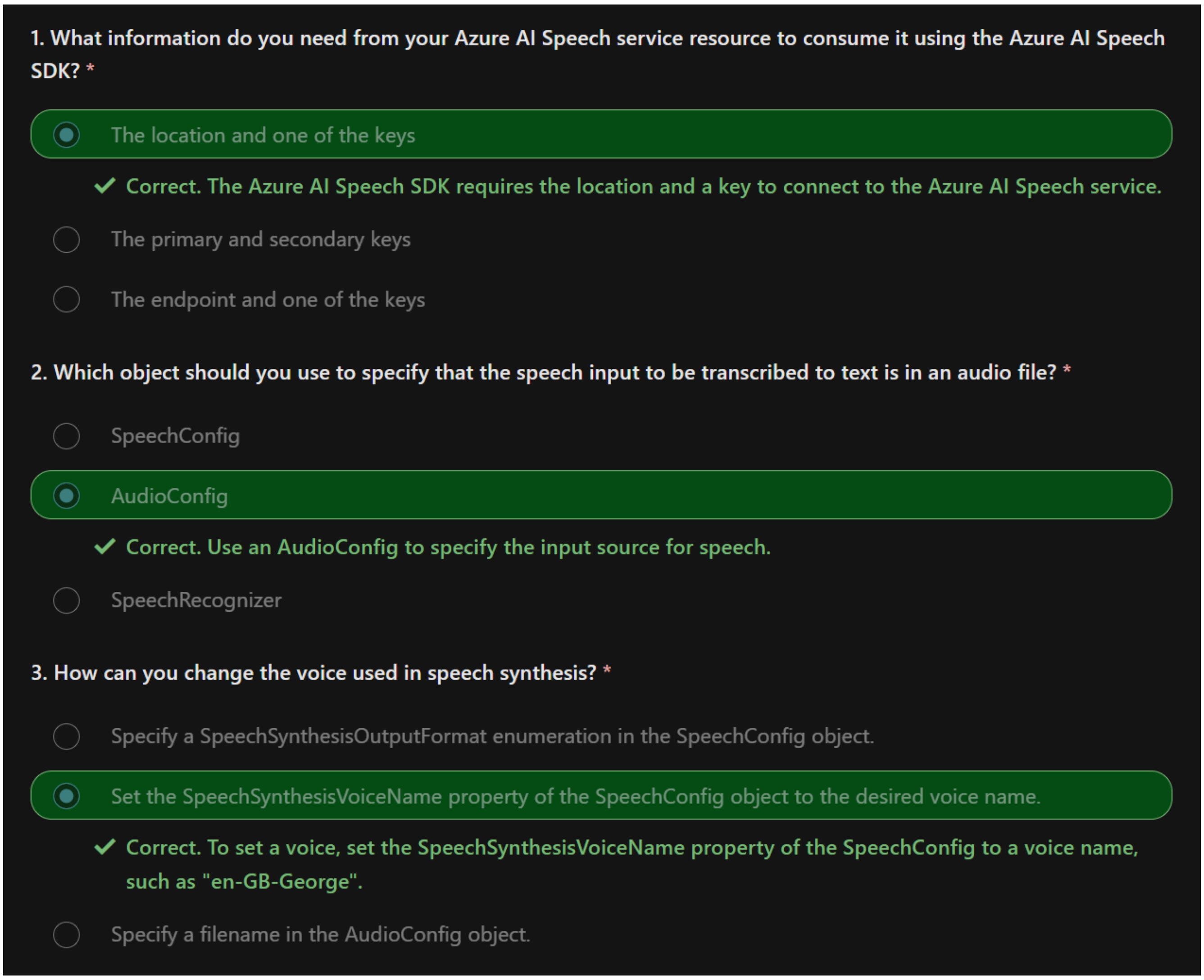
   ```
   python speaking-clock.py
   ```

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!

More information

For more information about using the **Speech-to-text** and **Text-to-speech** APIs, see the Speech-to-text documentation and Text-to-speech documentation.

Knowledge Check


1. What information do you need from your Azure AI Speech service resource to consume it using the Azure AI Speech SDK? *

[✓ The location and one of the keys]

✓ Correct. The Azure AI Speech SDK requires the location and a key to connect to the Azure AI Speech service.

The primary and secondary keys

The endpoint and one of the keys

2. Which object should you use to specify that the speech input to be transcribed to text is in an audio file? *

SpeechConfig

[✓ AudioConfig]

✓ Correct. Use an AudioConfig to specify the input source for speech.

SpeechRecognizer

3. How can you change the voice used in speech synthesis? *

Specify a SpeechSynthesisOutputFormat enumeration in the SpeechConfig object.

[✓ Set the SpeechSynthesisVoiceName property of the SpeechConfig object to the desired voice name.]

✓ Correct. To set a voice, set the SpeechSynthesisVoiceName property of the SpeechConfig to a voice name, such as "en-GB-George".

Specify a filename in the AudioConfig object.

Summary

In this module, you learned how to:

- Provision an Azure resource for the Azure AI Speech service
- Use the Speech to text API to implement speech recognition
- Use the Text to speech API to implement speech synthesis
- Configure audio format and voices
- Use Speech Synthesis Markup Language (SSML)

To learn more about the Azure AI Speech, refer to the Azure AI Speech service documentation.

✍️ Compiled by Kenneth Leung (2025)