Use the Speech and Language Understanding Services

You can integrate the Speech service with the Language Understanding service to create applications that can intelligently determine user intents from spoken input.

> **Note**: This exercise works best if you have a microphone. 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.

Clone the repository for this course

If you have already cloned **AI-102-AIEngineer** code repository to the environment where you're working on this lab, open it in Visual Studio Code; otherwise, follow these steps to clone it now.

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/AI-102-AIEngineer` 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**.

Create Language Understanding resources

If you already have Language Understanding authoring and prediction resources in your Azure subscription, you can use them in this exercise. Otherwise, follow these instructions to create them.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.
2. Select the **+ Create a resource** button, search for *language understanding*, and create a **Language Understanding** resource with the following settings:
   - **Create option**: Both
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *Choose or create a resource group (if you are using a restricted subscription, you may not have permission to create a new resource group - use the one provided)*
   - **Name**: *Enter a unique name*
   - **Authoring location**: *Select your preferred location*
   - **Authoring pricing tier**: F0
   - **Prediction location**: *Choose the same location as your authoring location*
   - **Prediction pricing tier**: F0 (*If F0 is not available, choose S0*)
3. Wait for the resources to be created, and note that two Language Understanding resources are provisioned; one for authoring, and another for prediction. You can view both of these by navigating to the resource group where you created them.

Prepare a Language Understanding app

If you already have a **Clock** app from a previous exercise, open it in the Language Understanding portal at `https://www.luis.ai`. Otherwise, follow these instructions to create it.

1. In a new browser tab, open the Language Understanding portal at `https://www.luis.ai`.

2. Sign in using the Microsoft account associated with your Azure subscription. If this is the first time you have signed into the Language Understanding portal, you may need to grant the app some permissions to access your account details. Then complete the *Welcome* steps by selecting your Azure subscription and the authoring resource you just created.
3. Open the **Conversation Apps** page, next to **+ New app**, view the drop-down list and select **Import As LU**. Browse to the **11-luis-speech** subfolder in the project folder containing the lab files for this exercise, and select **Clock.lu**. Then specify a unique name for the clock app.
4. If a panel with tips for creating an effective Language Understanding app is displayed, close it.

Train and publish the app with *Speech Priming*

1. At the top of the Language Understanding portal, select **Train** to train the app if it has not already been trained.
2. At the top right of the Language Understanding portal, select **Publish**. Then select the **Production slot** and change the settings to enable **Speech Priming** (this will result in better performance for speech recognition).
3. After publishing is complete, at the top of the Language Understanding portal, select **Manage**.
4. On the **Settings** page, note the **App ID**. Client applications need this to use your app.
5. On the **Azure Resources** page, under **Prediction resources**, if no prediction resource is listed, add the prediction resource in your Azure subscription.
6. Note the **Primary Key**, **Secondary Key**, and **Location** (not endpoint!) for the prediction resource. Speech SDK client applications need the location and one of the keys to connect to the prediction resource and be authenticated.

Configure a client application for Language Understanding

In this exercise, you'll create a client application that accepts spoken input and uses your Language Understanding app to predict the user's intent.

> **Note**: You can choose to use the SDK for either **C#** or **Python** in this exercise. In the steps that follow, perform the actions appropriate for your preferred language.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **11-luis-speech** folder and expand the **C-Sharp** or **Python** folder depending on your language preference.
2. View the contents of the **speaking-clock-client** folder, and note that it contains a file for configuration settings:

    - **C#**: appsettings.json
    - **Python**: .env

    Open the configuration file and update the configuration values it contains to include the **App ID** for your Language Understanding app, and the **Location** (not the full endpoint - for example, *eastus*) and one of the **Keys** for its prediction resource (from the **Manage** page for your app in the Language Understanding portal).

Install SDK Packages

To use the Speech SDK with the Language Understanding service, you need to install the Speech SDK package for your programming language.

1. In Visual Studio, right-click the **speaking-clock-client** folder and open an integrated terminal. Then install the Language Understanding SDK package by running the appropriate command for your language preference:

    C#

    Code Copy

```
dotnet add package Microsoft.CognitiveServices.Speech --version 1.14.0
```

Python

Code Copy

```
pip install azure-cognitiveservices-speech==1.14.0
```

2. Additionally, if your system does not have a working microphone, you will need to use an audio file to provide spoken input for your application. In this case, use the following commands to install an additional package so your program can play the audio file (you can skip this if you intend to use a microphone):

C#

Code Copy

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

Python

Code Copy

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

3. Note that the **speaking-clock-client** folder contains a code file for the client application:

   - **C#**: Program.cs
   - **Python**: speaking-clock-client.py

4. 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 Speech SDK:

C#

Code Copy

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

Python

Code Copy

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

5. Additionally, if your system does not have a working microphone, under the existing namespace imports, add the following code to import the library you will use to play an audio file:

C#

Code Copy

```
using System.Media;
```

Python

Code Copy

```
from playsound import playsound
```

Create an *IntentRecognizer*

The **IntentRecognizer** class provides a client object that you can use to get Language Understanding predictions from spoken input.

1. In the **Main** function, note that code to load the App ID, prediction region, and key from the configuration file has already been provided. Then find the comment **Configure speech service and get intent recognizer**, and add the following code depending on whether you will use a microphone or an audio file for speech input:

   ### **If you have a working microphone:**

   C#

   Code Copy

   ```
   // Configure speech service and get intent recognizer
   SpeechConfig speechConfig = SpeechConfig.FromSubscription(predictionKey, predictionRegion);
   AudioConfig audioConfig = AudioConfig.FromDefaultMicrophoneInput();
   IntentRecognizer recognizer = new IntentRecognizer(speechConfig, audioConfig);
   ```

   Python

   Code Copy

   ```
   # Configure speech service and get intent recognizer
   speech_config = speech_sdk.SpeechConfig(subscription=lu_prediction_key,
   region=lu_prediction_region)
   audio_config = speech_sdk.AudioConfig(use_default_microphone=True)
   recognizer = speech_sdk.intent.IntentRecognizer(speech_config, audio_config)
   ```

   ### **If you need to use an audio file:**

   C#

   Code Copy

   ```
   // Configure speech service and get intent recognizer
   string audioFile = "time-in-london.wav";
   SoundPlayer wavPlayer = new SoundPlayer(audioFile);
   wavPlayer.Play();
   System.Threading.Thread.Sleep(2000);
   SpeechConfig speechConfig = SpeechConfig.FromSubscription(predictionKey, predictionRegion);
   AudioConfig audioConfig = AudioConfig.FromWavFileInput(audioFile);
   IntentRecognizer recognizer = new IntentRecognizer(speechConfig, audioConfig);
   ```

   Python

   Code Copy

```
# Configure speech service and get intent recognizer
audioFile = 'time-in-london.wav'
playsound(audioFile)
speech_config = speech_sdk.SpeechConfig(subscription=lu_prediction_key,
region=lu_prediction_region)
audio_config = speech_sdk.AudioConfig(filename=audioFile)
recognizer = speech_sdk.intent.IntentRecognizer(speech_config, audio_config)
```

Get a predicted intent from spoken input

Now you're ready to implement code that uses the Speech SDK to get a predicted intent from spoken input.

1. In the **Main** function, immediately beneath the code you just added, find the comment **Get the model from the AppID and add the intents we want to use** and add the following code to get your Language Understanding model (based on its App ID) and specify the intents that we want the recognizer to identify.

   C#

   Code Copy

   ```
   // Get the model from the AppID and add the intents we want to use
   var model = LanguageUnderstandingModel.FromAppId(luAppId);
   recognizer.AddIntent(model, "GetTime", "time");
   recognizer.AddIntent(model, "GetDate", "date");
   recognizer.AddIntent(model, "GetDay", "day");
   recognizer.AddIntent(model, "None", "none");
   ```

   Note that you can specify a string-based ID for each intent

   Python

   Code Copy

   ```
   # Get the model from the AppID and add the intents we want to use
   model = speech_sdk.intent.LanguageUnderstandingModel(app_id=lu_app_id)
   intents = [
       (model, "GetTime"),
       (model, "GetDate"),
       (model, "GetDay"),
       (model, "None")
   ]
   recognizer.add_intents(intents)
   ```

2. Under the comment **Process speech input**, add the following code, which uses the recognizer to asynchronously call the Language Understanding service with spoken input, and retrieve response. If the response includes a predicted intent, the spoken query, predicted intent, and full JSON response are displayed. Otherwise the code handles the response based on the reason returned.

C#

Code Copy

```
// Process speech input
string intent = "";
var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);
if (result.Reason == ResultReason.RecognizedIntent)
{
    // Intent was identified
    intent = result.IntentId;
    Console.WriteLine($"Query: {result.Text}");
    Console.WriteLine($"Intent Id: {intent}.");
    string jsonResponse =
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult);
    Console.WriteLine($"JSON Response:\n{jsonResponse}\n");

    // Get the first entity (if any)

    // Apply the appropriate action

}
else if (result.Reason == ResultReason.RecognizedSpeech)
{
    // Speech was recognized, but no intent was identified.
    intent = result.Text;
    Console.Write($"I don't know what {intent} means.");
}
else if (result.Reason == ResultReason.NoMatch)
{
    // Speech wasn't recognized
    Console.WriteLine($"Sorry. I didn't understand that.");
}
else if (result.Reason == ResultReason.Canceled)
{
    // Something went wrong
    var cancellation = CancellationDetails.FromResult(result);
    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");
    if (cancellation.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
    }
}
```

Python

Code Copy

```
# Process speech input
intent = ''
result = recognizer.recognize_once_async().get()
if result.reason == speech_sdk.ResultReason.RecognizedIntent:
    intent = result.intent_id
    print("Query: {}".format(result.text))
    print("Intent: {}".format(intent))
    json_response = json.loads(result.intent_json)
    print("JSON Response:\n{}\n".format(json.dumps(json_response, indent=2)))

    # Get the first entity (if any)

    # Apply the appropriate action

elif result.reason == speech_sdk.ResultReason.RecognizedSpeech:
    # Speech was recognized, but no intent was identified.
    intent = result.text
    print("I don't know what {} means.".format(intent))
elif result.reason == speech_sdk.ResultReason.NoMatch:
    # Speech wasn't recognized
    print("Sorry. I didn't understand that.")
elif result.reason == speech_sdk.ResultReason.Canceled:
    # Something went wrong
    print("Intent recognition canceled: {}".format(result.cancellation_details.reason))
    if result.cancellation_details.reason == speech_sdk.CancellationReason.Error:
        print("Error details: {}".format(result.cancellation_details.error_details))
```

The code you've added so far identifies the *intent*, but some intents can reference *entities*, so you must add code to extract the entity information from the JSON returned by the service.

1. In the code you just added, find the comment **Get the first entity (if any)** and add the following code beneath it:

C#

Code Copy

```
// Get the first entity (if any)
JObject jsonResults = JObject.Parse(jsonResponse);
string entityType = "";
string entityValue = "";
if (jsonResults["entities"].HasValues)
{
    JArray entities = new JArray(jsonResults["entities"][0]);
    entityType = entities[0]["type"].ToString();
    entityValue = entities[0]["entity"].ToString();
    Console.WriteLine(entityType + ": " + entityValue);
}
```

Python

Code Copy

```
# Get the first entity (if any)
entity_type = ''
entity_value = ''
if len(json_response["entities"]) > 0:
    entity_type = json_response["entities"][0]["type"]
    entity_value = json_response["entities"][0]["entity"]
    print(entity_type + ': ' + entity_value)
```

Your code now uses the Language Understanding app to predict an intent as well as any entities that were detected in the input utterance. Your client application must now use that prediction to determine and perform the appropriate action.

1. Beneath the code you just added, find the comment **Apply the appropriate action**, and add the following code, which checks for intents supported by the application (**GetTime**, **GetDate**, and **GetDay**) and determines if any relevant entities have been detected, before calling an existing function to produce an appropriate response.

C#

Code Copy

```
// Apply the appropriate action
switch (intent)
{
    case "time":
        var location = "local";
        // Check for entities
        if (entityType == "Location")
        {
            location = entityValue;
        }
        // Get the time for the specified location
        var getTimeTask = Task.Run(() => GetTime(location));
        string timeResponse = await getTimeTask;
        Console.WriteLine(timeResponse);
        break;
    case "day":
        var date = DateTime.Today.ToShortDateString();
        // Check for entities
        if (entityType == "Date")
        {
            date = entityValue;
        }
        // Get the day for the specified date
        var getDayTask = Task.Run(() => GetDay(date));
        string dayResponse = await getDayTask;
        Console.WriteLine(dayResponse);
        break;
    case "date":
        var day = DateTime.Today.DayOfWeek.ToString();
        // Check for entities
        if (entityType == "Weekday")
        {
            day = entityValue;
        }

        var getDateTask = Task.Run(() => GetDate(day));
        string dateResponse = await getDateTask;
        Console.WriteLine(dateResponse);
        break;
    default:
        // Some other intent (for example, "None") was predicted
        Console.WriteLine("You said " + result.Text.ToLower());
        if (result.Text.ToLower().Replace(".", "") == "stop")
        {
            intent = result.Text;
        }
        else
        {
            Console.WriteLine("Try asking me for the time, the day, or the date.");
        }
        break;
}
```

Python

Code Copy

```
# Apply the appropriate action
if intent == 'GetTime':
    location = 'local'
    # Check for entities
    if entity_type == 'Location':
        location = entity_value
    # Get the time for the specified location
    print(GetTime(location))

elif intent == 'GetDay':
    date_string = date.today().strftime("%m/%d/%Y")
    # Check for entities
    if entity_type == 'Date':
        date_string = entity_value
    # Get the day for the specified date
    print(GetDay(date_string))

elif intent == 'GetDate':
    day = 'today'
    # Check for entities
    if entity_type == 'Weekday':
        # List entities are lists
        day = entity_value
    # Get the date for the specified day
    print(GetDate(day))

else:
    # Some other intent (for example, "None") was predicted
    print('You said {}'.format(result.text))
    if result.text.lower().replace('.', '') == 'stop':
        intent = result.text
    else:
        print('Try asking me for the time, the day, or the date.')
```

Run the client application

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

   C#

   Code Copy

   ```
   dotnet run
   ```

   Python

   Code Copy

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

2. If using a microphone, speak utterances aloud to test the application. For example, try the following (re-running the program each time):

   What's the time?

   What time is it?

What day is it?

What is the time in London?

What's the date?

What date is Sunday?

> **Note**: The logic in the application is deliberately simple, and has a number of limitations, but should serve the purpose of testing the ability for the Language Understanding model to predict intents from spoken input using the Speech SDK. You may have trouble recognizing the **GetDay** intent with a specific date entity due to the difficulty in verbalizing a date in *MM/DD/YYYY* format!

More information

To learn more about Speech and Language Understanding integration, see the Speech documentation.