**3.1 - Analyze text with Azure AI Language**

Azure AI Language is designed to help you extract information from text. It provides functionality that you can use for tasks like:

* **Language detection** - determining the language in which text is written.
* **Key phrase extraction** - identifying important words and phrases in the text that indicate the main points.
* **Sentiment analysis** - quantifying how positive or negative the text is.
* **Named entity recognition** - detecting references to entities, including people, locations, time periods, organizations, and more.
* **Entity linking** - identifying specific entities by providing reference links to Wikipedia articles.

**Detect language**

The Azure AI Language detection API evaluates text input and, for each document submitted, returns language identifiers with a score indicating the strength of the analysis.

This capability is useful for content stores that collect arbitrary text, where language is unknown.

Language detection can work with documents or single phrases. It's important to note that the document size must be under 5,120 characters. The size limit is per document and each collection is restricted to 1,000 items (IDs).

**Extract key phrases**

Key phrase extraction is the process of evaluating the text of a document, or documents, and then identifying the main points around the context of the document(s).

Key phrase extraction works best for larger documents (the maximum size that can be analyzed is 5,120 characters).

As with language detection, the REST interface enables you to submit one or more documents for analysis.

**Analyze sentiment**

Sentiment analysis is used to evaluate how positive or negative a text document is, which can be useful in various workloads, such as:

Evaluating a movie, book, or product by quantifying sentiment based on reviews.

Prioritizing customer service responses to correspondence received through email or social media messaging.

When using Azure AI Language to evaluate sentiment, the response includes overall document sentiment and individual sentence sentiment for each document submitted to the service.

Overall document sentiment is based on sentences:

* + If all sentences are neutral, the overall sentiment is neutral.
  + If sentence classifications include only positive and neutral, the overall sentiment is positive.
  + If the sentence classifications include only negative and neutral, the overall sentiment is negative.
  + If the sentence classifications include positive and negative, the overall sentiment is mixed.

**Extract entities**

Named Entity Recognition identifies entities that are mentioned in the text. Entities are grouped into categories and subcategories, for example:

Person, Location, DateTime, Organization, Address, Email, URL

**Extract linked entities**

In some cases, the same name might be applicable to more than one entity. For example, does an instance of the word "Venus" refer to the planet or the goddess from mythology?

Entity linking can be used to disambiguate entities of the same name by referencing an article in a knowledge base. Wikipedia provides the knowledge base for the Text Analytics service. Specific article links are determined based on entity context within the text.

For example, "I saw Venus shining in the sky" is associated with the link https://en.wikipedia.org/wiki/Venus; while "Venus, the goddess of beauty" is associated with <https://en.wikipedia.org/wiki/Venus_(mythology)>.

**3.2 -** **Create question answering solutions with Azure AI Language**

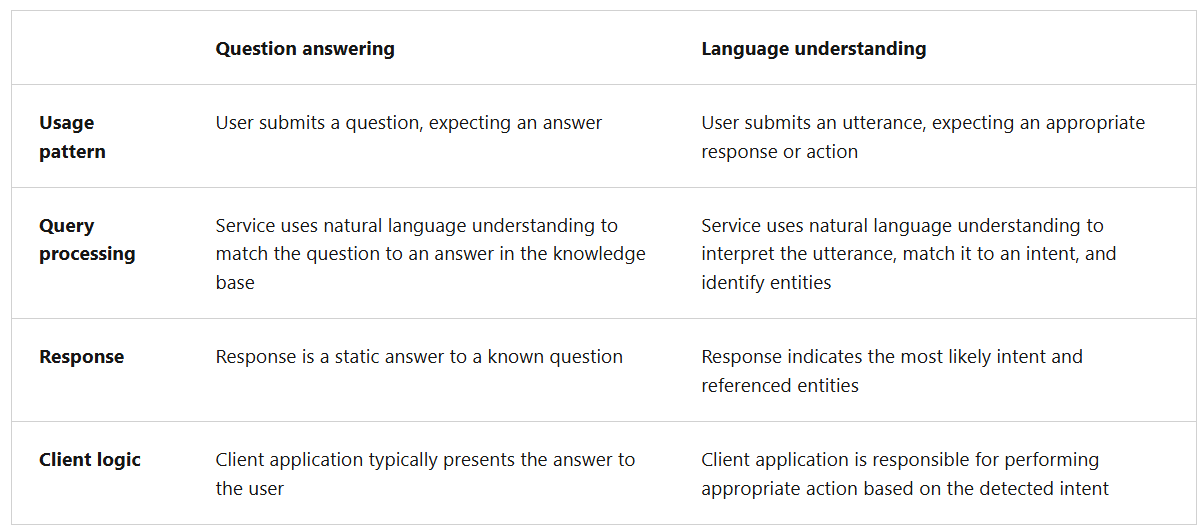
**Azure AI Language** includes a question answering capability, which enables you to define a knowledge base of question and answer pairs that can be queried using natural language input. The knowledge base can be published to a REST endpoint and consumed by client applications, commonly bots.

The knowledge base can be created from existing sources, including:

* Web sites containing frequently asked question (FAQ) documentation.
* Files containing structured text, such as brochures or user guides.
* Built-in chit chat question and answer pairs that encapsulate common conversational exchanges.

**Compare question answering to Azure AI Language understanding**

The two features are similar in that they both enable you to define a language model that can be queried using natural language expressions. However, there are some differences in the use cases that they are designed to address, as shown in the following table:



**Create a knowledge base**

To create a question answering solution, you can use the REST API or SDK to write code that defines, trains, and publishes the knowledge base. However, it's more common to use the Language Studio (<https://language.azure.com/>) web interface to define and manage a knowledge base.

**Implement multi-turn conversation**

Although you can often create an effective knowledge base that consists of individual question and answer pairs, sometimes you might need to ask follow-up questions to elicit more information from a user before presenting a definitive answer. This kind of interaction is referred to as a multi-turn conversation.

You can enable multi-turn responses when importing questions and answers from an existing web page or document based on its structure, or you can explicitly define follow-up prompts and responses for existing question and answer pairs.

When you define a follow-up prompt for multi-turn conversation, you can link to an existing answer in the knowledge base or define a new answer specifically for the follow-up. You can also restrict the linked answer so that it is only ever displayed in the context of the multi-turn conversation initiated by the original question.

**Test and publish a knowledge base**

After you have defined a knowledge base, you can train its natural language model, and test it before publishing it for use in an application or bot.

**Testing a knowledge base**

You can test your knowledge base interactively in Language Studio, submitting questions and reviewing the answers that are returned. You can inspect the results to view their confidence scores as well as other potential answers.

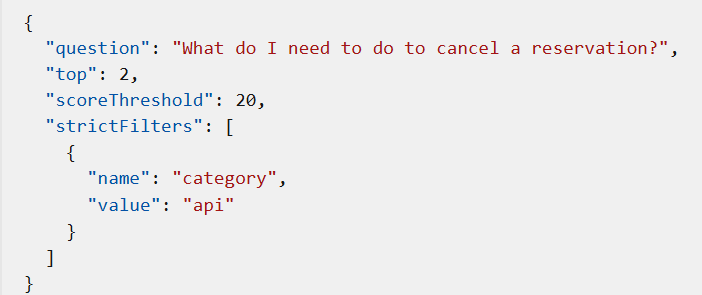
**Deploying a knowledge base**

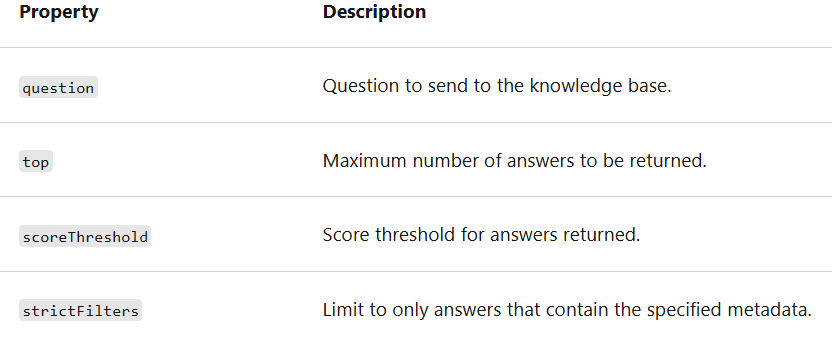
When you're happy with the performance of your knowledge base, you can deploy it to a REST endpoint that client applications can use to submit questions and receive answers. You can deploy it directly from Language Studio.

**Use a knowledge base**

To consume the published knowledge base, you can use the REST interface.

The minimal request body for the function contains a question, like this:





The response includes the closest question match that was found in the knowledge base, along with the associated answer, the confidence score, and other metadata about the question and answer pair.

**Improve question answering performance**

After creating and testing a knowledge base, you can improve its performance with active learning and by defining synonyms.

**Use active learning**

Active learning can help you make continuous improvements to get better at answering user questions correctly over time. People often ask questions that are phrased differently, but ultimately have the same meaning. Active learning can help in situations like this because it enables you to consider alternate questions to each question and answer pair. Active learning is enabled by default. To use active learning, you can do the following:

**Create your question and answer pairs**

You create pairs of questions and answers in Language Studio for your project. You can also import a file that contains question and answer pairs to upload in bulk.

**Review suggestions**

* Active learning then begins to offer alternate questions for each question in your question and answer pairs. You access this from the Review suggestions pane.
* You review, and then accept or reject these alternate phrases suggested for each question by selecting the checkmark or delete symbol next to the alternate phrase. You can bulk accept or reject suggestions using the Accept all suggestions or Reject all suggestions option at the top.
* You can also manually add alternate questions when you select Add alternate question for a pair in the Edit knowledge base pane.

**Define synonyms**

Synonyms are useful when questions submitted by users might include multiple different words to mean the same thing. For example, a travel agency customer might refer to a "reservation" or a "booking". By defining these as synonyms, the question answering service can find an appropriate answer regardless of which term an individual customer uses.

**3.3 - Build a conversational language understanding model**

**Understand prebuilt capabilities of the Azure AI Language service**

* The Azure AI Language service provides various features for understanding human language. You can use each feature to better communicate with users, better understand incoming communication, or use them together to provide more insight into what the user is saying, intending, and asking about.
* Azure AI Language service features fall into two categories: Pre-configured features, and Learned features. Learned features require building and training a model to correctly predict appropriate labels.

**Pre-configured features**

The Azure AI Language service provides certain features without any model labeling or training. Once you create your resource, you can send your data and use the returned results within your app.

The following features are all pre-configured.

* **Summarization** is available for both documents and conversations, and will summarize the text into key sentences that are predicted to encapsulate the input's meaning.
* **Named entity recognition** can extract and identify entities, such as people, places, or companies, allowing your app to recognize different types of entities for improved natural language responses.
* **PII detection** allows you to identify, categorize, and redact information that could be considered sensitive, such as email addresses, home addresses, IP addresses, names, and protected health information.
* **Key phrase extraction** is a feature that quickly pulls the main concepts out of the provided text.
* **Sentiment analysis** identifies how positive or negative a string or document is.
* **Language detection** takes one or more documents, and identifies the language for each.

**Learned features**

Learned features require you to label data, train, and deploy your model to make it available to use in your application. These features allow you to customize what information is predicted or extracted.

The following features are all learned features.

* **Conversational language understanding (CLU)** is one of the core custom features offered by Azure AI Language. CLU helps users to build custom natural language understanding models to predict overall intent and extract important information from incoming utterances. CLU does require data to be tagged by the user to teach it how to predict intents and entities accurately.
* **Custom entity recognition** takes custom labeled data and extracts specified entities from unstructured text.
* **Custom text classification** enables users to classify text or documents as custom defined groups. For example, you can train a model to look at news articles and identify the category they should fall into, such as News or Entertainment.
* **Question answering** is a mostly pre-configured feature that provides answers to questions provided as input. The data to answer these questions comes from documents like FAQs or manuals.

**Understand resources for building a conversational language understanding model**

To use the Language Understanding service to develop a NLP solution, you'll need to create a Language resource in Azure. That resource will be used for both authoring your model and processing prediction requests from client applications.

**Build your model**

For features that require a model for prediction, you'll need to build, train and deploy that model before using it to make a prediction. This building and training will teach the Azure AI Language service what to look for.

**Use Language Studio**

For a more visual method of building, training, and deploying your model, you can use Language Studio to achieve each of the steps. On the main page, you can choose to create a Conversational language understanding project. Once the project is created, then go through the process to build, train, and deploy your model.

**Use the REST API**

One way to build your model is through the REST API. The pattern would be to create your project, import data, train, deploy, then use your model.

These tasks are done asynchronously; you'll need to submit a request to the appropriate URI for each step, and then send another request to get the status of that job.

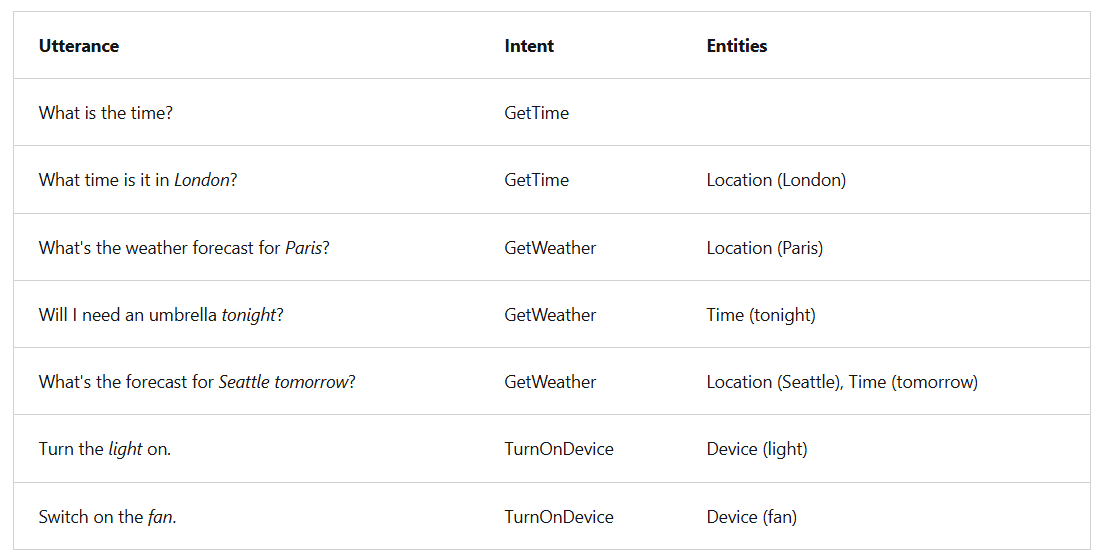
**Define intents, utterances, and entities**

**Utterances** are the phrases that a user might enter when interacting with an application that uses your language model.

An **intent** represents a task or action the user wants to perform, or more simply the meaning of an utterance. You create a model by defining intents and associating them with one or more utterances.

**Entities** are used to add specific context to intents. For example, you might define a TurnOnDevice intent that can be applied to multiple devices, and use entities to define the different devices.

Consider the following utterances, intents, and entities:



**You can split entities into a few different component types:**

**Learned entities** are the most flexible kind of entity, and should be used in most cases. You define a learned component with a suitable name, and then associate words or phrases with it in training utterances. When you train your model, it learns to match the appropriate elements in the utterances with the entity.

**List entities** are useful when you need an entity with a specific set of possible values - for example, days of the week. You can include synonyms in a list entity definition, so you could define a DayOfWeek entity that includes the values "Sunday", "Monday", "Tuesday", and so on; each with synonyms like "Sun", "Mon", "Tue", and so on.

**Prebuilt entities** are useful for common types such as numbers, datetimes, and names.

**3.4 - Create a custom text classification solution**

Create a custom text classification solutionCustom text classification assigns labels, which in the Azure AI Language service is a class that the developer defines, to text files. For example, a video game summary might be classified as "Adventure", "Strategy", "Action" or "Sports".

Custom text classification falls into two types of projects:

* **Single label classification** - you can assign only one class to each file. Following the above example, a video game summary could only be classified as "Adventure" or "Strategy".
* **Multiple label classification** - you can assign multiple classes to each file. This type of project would allow you to classify a video game summary as "Adventure" or "Adventure and Strategy".

When creating your custom text classification project, you can specify which project you want to build.

**Single vs. multiple label projects**

Beyond the ability to put files into multiple classifications, the key differences with multiple label classification projects are labeling, considerations for improving your model, and the API payload for classification tasks.

**Labeling data**

In **single label** projects, **each file is assigned one class** during the labeling process; class assignment in Azure AI Language only allows you to select one class.

When labeling **multiple label** projects, you can **assign as many classes that you want per file**. The impact of the added complexity means your data has to remain clear and provide a good distribution of possible inputs for your model to learn from.

Labeling data correctly, especially for multiple label projects, is directly correlated with how well your model performs. The higher the quality, clarity, and variation of your data set is, the more accurate your model will be.

**Evaluating and improving your model**

Measuring predictive performance of your model goes beyond how many predictions were correct. Correct classifications are when the actual label is x and the model predicts a label x. In the real world, documents result in different kinds of errors when a classification isn't correct:

**False positive** - model predicts x, but the file isn't labeled x.

**False negative** - model doesn't predict label x, but the file in fact is labeled x.

These metrics are translated into three measures provided by Azure AI Language:

* **Recall** - Of all the actual labels, how many were identified; the ratio of true positives to all that was labeled.
* **Precision** - How many of the predicted labels are correct; the ratio of true positives to all identified positives.
* **F1 Score** - A function of recall and precision, intended to provide a single score to maximize for a balance of each component

With a single label project, you can identify which classes aren't classified as well as others and find more quality data to use in training your model. For multiple label projects, figuring out quality data becomes more complex due to the matrix of possible permutations of combined labels.

**API payload**

Azure AI Language provides a REST API to build and interact with your model, using a JSON body to specify the request.

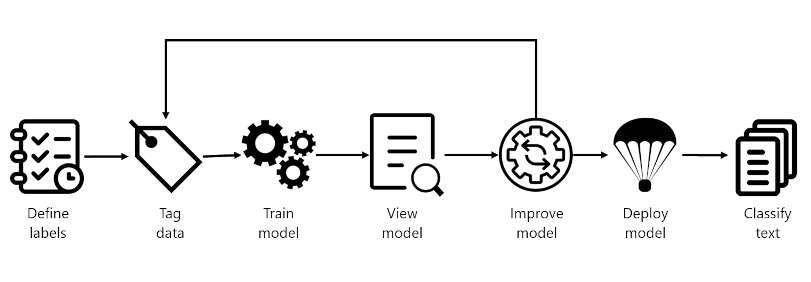
To submit a classification task, the API requires the JSON body to specify which task to execute.

* Single label classification models specify a project type of **customSingleLabelClassification.**
* Multiple label classification models specify a project type of **CustomMultiLabelClassification**.

**Understand how to build text classification projects**

Custom text classification projects are your workspace to build, train, improve, and deploy your classification model. You can work with your project in two ways: through Language Studio and via the REST API. Language Studio is the GUI that will be used in the lab, but the REST API has the same functionality. Regardless of which method you prefer, the steps for developing your model are the same.

**Azure AI Language project life cycle**



* **Define labels**: Understanding the data you want to classify, identify the possible labels you want to categorize into.
* **Tag data**: Tag, or label, your existing data, specifying the label or labels each file falls under. Labeling data is important since it's how your model will learn how to classify future files. Best practice is to have clear differences between labels to avoid ambiguity, and provide good examples of each label for the model to learn from.
* **Train model**: Train your model with the labeled data. Training will teach our model what types of video game summaries should be labeled which genre.
* **View model**: After your model is trained, view the results of the model. Your model is scored between 0 and 1, based on the precision and recall of the data tested. Take note of which genre didn't perform well.
* **Improve model**: Improve your model by seeing which classifications failed to evaluate to the right label, see your label distribution, and find out what data to add to improve performance. Try to find more examples of each label to add to your dataset for retraining your model.
* **Deploy model**: Once your model performs as desired, deploy your model to make it available via the API. Your model might be named "GameGenres", and once deployed can be used to classify game summaries.
* **Classify text**: Use your model for classifying text.

**How to split datasets for training**

When labeling your data, you can specify which dataset you want each file to be:

* **Training** - The training dataset is used to actually train the model; the data and labels provided are fed into the machine learning algorithm to teach your model what data should be classified to which label. The training dataset will be the larger of the two datasets, recommended to be about 80% of your labeled data.
* **Testing** - The testing dataset is labeled data used to verify you model after it's trained. Azure will take the data in the testing dataset, submit it to the model, and compare the output to how you labeled your data to determine how well the model performed. The result of that comparison is how your model gets scored and helps you know how to improve your predictive performance.

During the Train model step, there are two options for how to train your model.

* **Automatic split** - Azure takes all of your data, splits it into the specified percentages randomly, and applies them in training the model. This option is best when you have a larger dataset, data is naturally more consistent, or the distribution of your data extensively covers your classes.
* **Manual split** - Manually specify which files should be in each dataset. When you submit the training job, the Azure AI Language service will tell you the split of the dataset and the distribution. This split is best used with smaller datasets to ensure the correct distribution of classes and variation in data are present to correctly train your model.

To use the automatic split, put all files into the training dataset when labeling your data (this option is the default). To use the manual split, specify which files should be in testing versus training during the labeling of your data.

**Deployment options**

Azure AI Language allows each project to create both multiple models and multiple deployments, each with their own unique name. Benefits include ability to:

* Test two models side by side
* Compare how the split of datasets impact performance
* Deploy multiple versions of your model

**Each project has a limit of ten deployment names**

During deployment you can choose the name for the deployed model, which can then be selected when submitting a classification task

**3.5 - Custom named entity recognition(NER)**

Custom NER is an Azure API service that looks at documents, identifies, and extracts user defined entities. These entities could be anything from names and addresses from bank statements to knowledge mining to improve search results.

Custom NER is part of Azure AI Language in Azure AI services.

**Custom vs built-in NER**

Azure AI Language provides certain **built-in entity recognition**, to recognize things such as a person, location, organization, or URL. Built-in NER allows you to set up the service with minimal configuration, and extract entities. To call a built-in NER, create your service and call the endpoint for that NER service like this:

<YOUR-ENDPOINT>/language/analyze-text/jobs?api-version=<API-VERSION>

**Custom NER** is available when the entities you want to extract aren't part of the built-in service or you only want to extract specific entities. You can make your custom NER model as simple or complex as is required for your app.

**Considerations for data selection and refining entities**

For the best performance, you'll need to use both high quality data to train the model and clearly defined entity types.

High quality data will let you spend less time refining and yield better results from your model.

* **Diversity** - use as diverse of a dataset as possible without losing the real-life distribution expected in the real data. You'll want to use sample data from as many sources as possible, each with their own formats and number of entities. It's best to have your dataset represent as many different sources as possible.
* **Distribution** - use the appropriate distribution of document types. A more diverse dataset to train your model will help your model avoid learning incorrect relationships in the data.
* **Accuracy** - use data that is as close to real world data as possible. Fake data works to start the training process, but it likely will differ from real data in ways that can cause your model to not extract correctly.

**How to extract entities**

To submit an extraction task, the API requires the JSON body to specify which task to execute. For custom NER, the task for the JSON payload is CustomEntityRecognition.

Your payload will look similar to the following JSON:



**Project limits**

The Azure AI Language service enforces the following restrictions:

* **Training** - at least 10 files, and not more than 100,000
* **Deployments** - 10 deployment names per project
* **APIs**
  + **Authoring** - this API creates a project, trains, and deploys your model. Limited to 10 POST and 100 GET per minute
  + **Analyze** - this API does the work of actually extracting the entities; it requests a task and retrieves the results. Limited to 20 GET or POST
* **Projects** - only 1 storage account per project, 500 projects per resource, and 50 trained models per project
* **Entities** - each entity can be up to 500 characters. You can have up to 200 entity types.

**Label your data**

Labeling, or tagging, your data correctly is an important part of the process to create a custom entity extraction model. Labels identify examples of specific entities in text used to train the model. Three things to focus on are:

* **Consistency** - Label your data the same way across all files for training. Consistency allows your model to learn without any conflicting inputs.
* **Precision** - Label your entities consistently, without unnecessary extra words. Precision ensures only the correct data is included in your extracted entity.
* **Completeness** - Label your data completely, and don't miss any entities. Completeness helps your model always recognize the entities present.

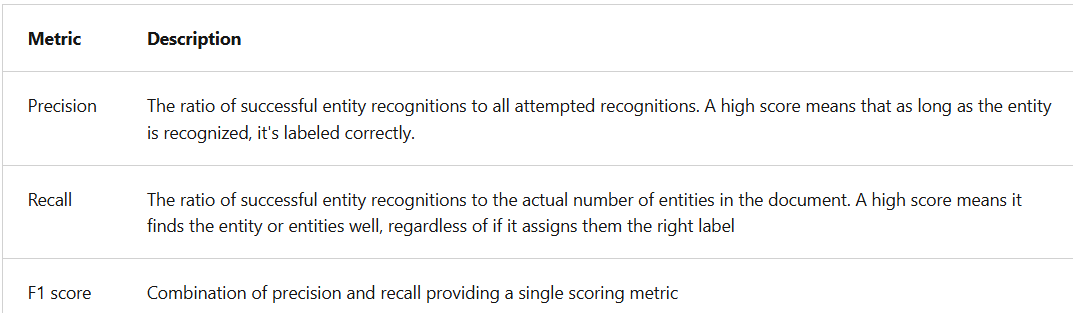
**How to label your data**

Language Studio is the most straight forward method for labeling your data. Language Studio allows you to see the file, select the beginning and end of your entity, and specify which entity it is.

Each label that you identify gets saved into a file that lives in your storage account with your dataset, in an auto-generated JSON file. This file then gets used by the model to learn how to extract custom entities. It's possible to provide this file when creating your project (if you're importing the same labels from a different project, for example) however it must be in the Accepted custom NER data formats.

**Train and evaluate your model**

* Training and evaluating your model is an iterative process of adding data and labels to your training dataset to teach the model more accurately. To know what types of data and labels need to be improved, Language Studio provides scoring in the View model details page on the left hand pane.
* Individual entities and your overall model score are broken down into three metrics to explain how they're performing and where they need to improve.
* Scores are available both per entity and for the model as a whole. You may find an entity scores well, but the whole model doesn't.



**How to interpret metrics**

Ideally we want our model to score well in both precision and recall, which means the entity recognition works well. If both metrics have a low score, it means the model is both struggling to recognize entities in the document, and when it does extract that entity, it doesn't assign it the correct label with high confidence.

If precision is low but recall is high, it means that the model recognizes the entity well but doesn't label it as the correct entity type.

If precision is high but recall is low, it means that the model doesn't always recognize the entity, but when the model extracts the entity, the correct label is applied.

**Confusion matrix**

On the same View model details page, there's another tab on the top for the Confusion matrix. This view provides a visual table of all the entities and how each performed, giving a complete view of the model and where it's falling short.

**3.6 - Translate text with Azure AI Translator service**

The Azure AI Translator provides an API for **translating text** **between 90 supported languages**.

**Provision an Azure AI Translator resource**

Azure AI Translator provides a multilingual text translation API that you can use for:

* Language detection.
* One-to-many translation.
* Script transliteration (converting text from its native script to an alternative script).

**Language detection**

You can use the Detect function of the REST API to detect the language in which text is written. For example, you could submit the following text to the **https://api.cognitive.microsofttranslator.com/detect?api-version=3.0** endpoint using curl.

**Translation**

To translate text from one language to another, use the Translate function; specifying a single from parameter to indicate the source language, and one or more to parameters to specify the languages into which you want the text translated.

For example, you could submit specifying a **from** parameter of ja (Japanese) and two **to** parameters with the values en (English) and fr (French). To do this, you'd call:

**h[ttps://api.cognitive.microsofttranslator.com/translate?apiv](ttps://api.cognitive.microsofttranslator.com/translate?api-v)ersion=3.0&from=ja&to=fr&to=en**

**Transliteration**

Our Japanese text is written using Hiragana script, so rather than translate it to a different language, you may want to transliterate it to a different script - for example to render the text in Latin script (as used by English language text).

To accomplish this, we can submit the Japanese text to the Transliterate function with a fromScript parameter of Jpan and a toScript parameter of Latn:

h<ttps://api.cognitive.microsofttranslator.com/transliterate?api-v>ersion=3.0&fromScript=Jpan&toScript=Latn

**3.7 - Create speech-enabled apps with Azure AI services**

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.
* **Keyword Recognition**: An API that enables your application to recognize keywords or short phrases.
* **Intent Recognition**: An API that uses conversational language understanding to determine the semantic meaning of spoken input.

Speech SDK package for python is **azure-cognitiveservices-speech**

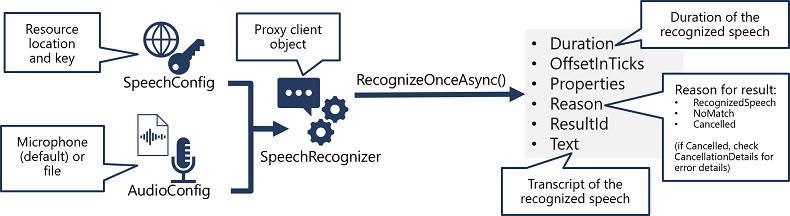
**Use the Azure AI Speech to Text API**

The Azure AI Speech service supports speech recognition through the following features:

* **Real-time transcription**: Instant transcription with intermediate results for live audio inputs.
* **Fast transcription**: Fastest synchronous output for situations with predictable latency.
* **Batch transcription**: Efficient processing for large volumes of prerecorded audio.
* **Custom speech**: Models with enhanced accuracy for specific domains and conditions.

**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:



* Use a **SpeechConfig** object to encapsulate the information required to connect to your **Azure AI Speech resource**. Specifically, its location and key.
* 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.
* Use the **SpeechConfig and AudioConfig** to create a **SpeechRecognizer** object. This object is a **proxy client for the Speech to text API**.
* 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.
* 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**. 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).

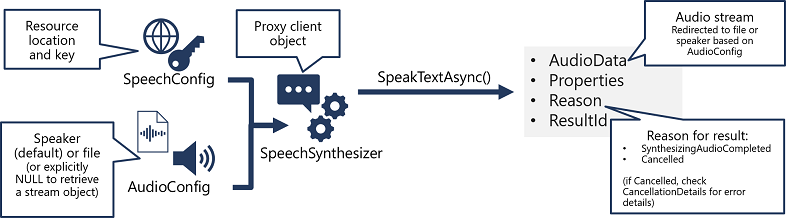
**Use the text to speech API**

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.

**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:



* Use the **SpeechConfig and AudioConfig** to create a **SpeechSynthesizer** object. This object is a **proxy client for the Text to speech API**.
* 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.
* 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

**speech\_config.set\_speech\_synthesis\_output\_format(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm)**

**Voices**

The Azure AI Speech service provides multiple voices that you can use to personalize your speech-enabled applications. Voices are identified by names that indicate a locale and a person's name

**speech\_config.speech\_synthesis\_voice\_name = "en-GB-George"**

**Use Speech Synthesis Markup Language**

While the Azure AI Speech SDK enables you to submit plain text to be synthesized into speech, 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.

**To submit an SSML description** to the **Speech service**, you can use an appropriate method of a **SpeechSynthesizer** object, like this:

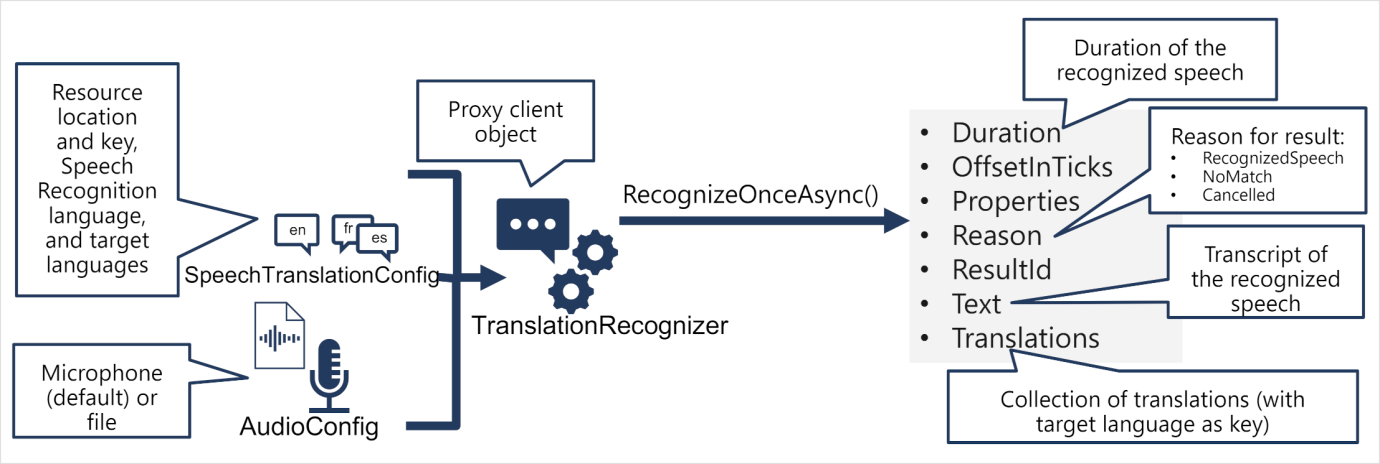
**speech\_synthesizer.speak\_ssml('<speak>...')**;

**3.8 - Translate speech with the Azure AI Speech service**

Translation of speech builds on speech recognition by recognizing and transcribing spoken input in a specified language, and returning translations of the transcription in one or more other languages.

**Translate speech to text**

The pattern for speech translation using the Azure AI Speech SDK is similar to speech recognition, with the addition of information about the source and target languages for translation:



* Use a **SpeechTranslationConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its location and key.
* The **SpeechTranslationConfig** object is also used to specify the speech recognition language (the language in which the input speech is spoken) and the target languages into which it should be translated.
* 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.
* Use the **SpeechTranslationConfig**, and **AudioConfig** to create a **TranslationRecognizer** object. This object is a proxy client for the Azure AI Speech translation API.
* Use the methods of the **TranslationRecognizer** object to call the underlying API functions. For example, the **RecognizeOnceAsync()** method uses the Azure AI Speech service to asynchronously translate a single spoken utterance.
* Process the response from Azure AI Speech. In the case of the **RecognizeOnceAsync()** method, the result is a **SpeechRecognitionResult** object that includes the following properties:
  + Duration
  + OffsetInTicks
  + Properties
  + Reason
  + ResultId
  + Text
  + Translations

If the operation was successful, the Reason property has the enumerated value RecognizedSpeech, the Text property contains the transcription in the original language. You can also access a Translations property which contains a dictionary of the translations (using the two-character ISO language code, such as "en" for English, as a key).

**Synthesize translations**

The **TranslationRecognizer** returns translated transcriptions of spoken input - essentially translating audible speech to text.

You can also synthesize the translation as speech to create speech-to-speech translation solutions. There are two ways you can accomplish this.

**Event-based synthesis**

When you want to perform 1:1 translation (translating from one source language into a single target language), you can use event-based synthesis to capture the translation as an audio stream. To do this, you need to:

* Specify the desired voice for the translated speech in the TranslationConfig. Create an event handler for the TranslationRecognizer object's Synthesizing event. In the event handler, use the GetAudio() method of the Result parameter to retrieve the byte stream of translated audio. The specific code used to implement an event handler varies depending on the programming language you're using.

**Manual synthesis**

Manual synthesis is an alternative approach to event-based synthesis that doesn't require you to implement an event handler. You can use manual synthesis to generate audio translations for one or more target languages.

Manual synthesis of translations is essentially just the combination of two separate operations in which you:

* Use a TranslationRecognizer to translate spoken input into text transcriptions in one or more target languages.
* Iterate through the Translations dictionary in the result of the translation operation, using a SpeechSynthesizer to synthesize an audio stream for each language.

**3.9 - Develop an audio-enabled generative AI application**

**Deploy a multimodal model**

To handle prompts that include audio, you need to deploy a multimodal generative AI model - in other words, a model that supports not only text-based input, but audio-based input as well. Multimodal models available in Azure AI Foundry include (among others):

Microsoft Phi-4-multimodal-instruct

OpenAI gpt-4o

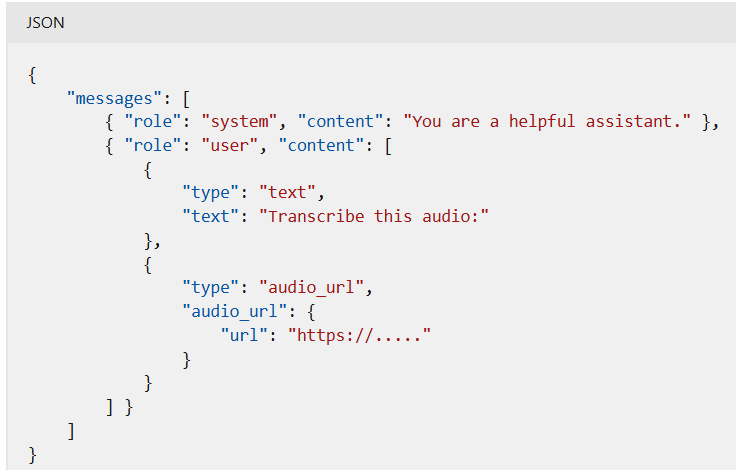
OpenAI gpt-4o-mini

**Testing multimodal models with audio-based prompts**

* After deploying a multimodal model, you can test it in the chat playground in Azure AI Foundry portal. Some models allow you to include audio attachments in the playground, either by uploading a file or recording a message.
* In the chat playground, you can upload a local audio file and add text to the message to elicit a response from a multimodal model.

**Develop an audio-based chat app**

* To develop a client app that engages in audio-based chats with a multimodal model, you can use the same basic techniques used for text-based chats. You require a connection to the endpoint where the model is deployed, and you use that endpoint to submit prompts that consists of messages to the model and process the responses.
* The key difference is that prompts for an audio-based chat include multi-part user messages that contain both a text content item and an audio content item.



The audio content item can be:

* A URL to an audio file in a web site.
* Binary audio data

When using binary data to submit a local audio file, the audio\_url content takes the form of a base64 encoded value in a data URL format:

