The Azure AI Language provides an API for common text analysis techniques that you can easily integrate into your own application code.

**Azure AI Language resource**

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

* *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.

**Azure AI Language Detection API**

**Overview**

* **Purpose**: Detects the language of text input and returns language identifiers with confidence scores.
* **Use Cases**:
  + Content stores with unknown text language.
  + Chat bots for determining the user’s language to configure responses appropriately.

**Capabilities**

* **Input**:
  + Handles both documents and single phrases.
  + Each document must be under 5,120 characters.
  + Can process up to 1,000 items (IDs) in a single collection.
* **Output**:
  + Returns language identifiers.
  + Includes a confidence score (between 0 and 1).

**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.

Sentence sentiment is based on confidence scores for **positive**, **negative**, and **neutral** classification values between 0 and 1.

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?

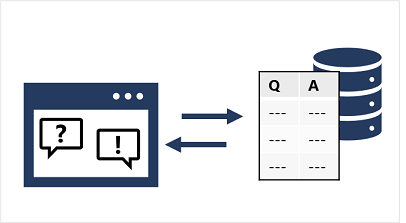
Entity linking can be used to disambiguate entities of the same name by referencing an article in a knowledge base. Wikipedia provides the knowledge base for the Text Analytics service. Specific article links are determined based on entity context within the text.

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)>.

As with all Azure AI Language service functions, you can submit one or more documents for analysis

**Understand question answering**

**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*.



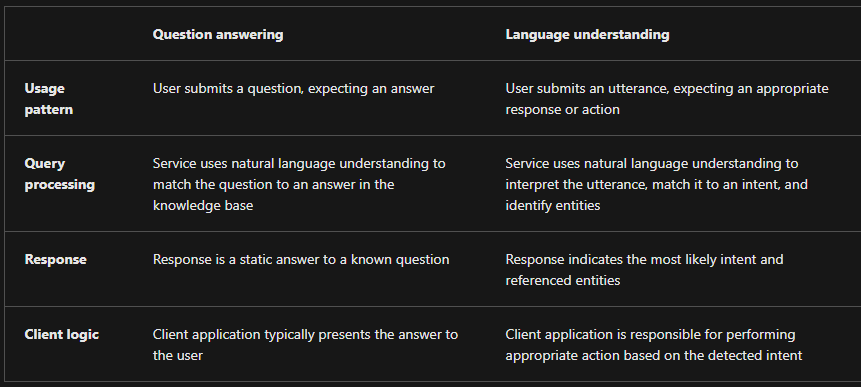
The knowledge base can be created from existing sources, including:

* 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**

A question answering knowledge base is a form of language model, which raises the question of when to use question answering, and when to use the *conversational language understanding* capabilities of Azure AI Language.

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



**Multi-turn Conversations in Knowledge Bases**

* **Multi-turn Conversations**: Interactions that involve follow-up questions to gather more information before providing a definitive answer.
* **Usefulness**: Crucial for handling complex queries where initial questions may be ambiguous or require additional context.
* **Enabling Multi-turn Responses**:
  + **Automatic**: When importing Q&A pairs from structured web pages or documents.
  + **Manual**: By explicitly defining follow-up prompts and responses for existing Q&A pairs in the knowledge base.
* **Example Scenario**:
  + **Initial Question**: "How can I cancel a reservation?"
  + **Ambiguity**: Could refer to either a hotel or a flight.
  + **Follow-up Prompt**: Clarifies the type of reservation.
  + **Example Answer**: "Cancellation policies depend on the type of reservation."
  + **Follow-up Prompts**: Include links to answers about canceling flights and canceling hotels.

**Improving Question Answering Performance**

**Use Active Learning**

* **Continuous Improvement**: Enables the knowledge base to get better at answering user questions over time.
* **Alternate Phrasings**: Helps handle different ways of phrasing the same question.
* **Enabled by Default**: Active learning is on by default in Language Studio.

**Steps to Use Active Learning**

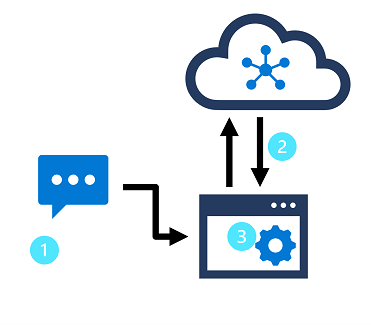
1. **Create Q&A Pairs**:
   * Use Language Studio to create question and answer pairs.
   * Import files containing Q&A pairs for bulk upload.
2. **Review Suggestions**:
   * Access alternate questions from the "Review suggestions" pane.
   * Accept or reject suggestions using the checkmark or delete symbol.
   * Bulk actions: Accept all suggestions or Reject all suggestions.
3. **Manual Addition**:
   * Add alternate questions manually in the "Edit knowledge base" pane.

**Define Synonyms**

* **Handling Variations**: Ensures the system understands different terms with the same meaning.
* **Example**: Synonyms like "reservation" and "booking" for travel agencies.
* **Benefits**: Enhances matching user queries with the correct answers, improving accuracy and user satisfaction.

Natural language processing (NLP) is a common AI problem in which software must be able to work with text or speech in the natural language form that a human user would write or speak.

A common design pattern for a natural language understanding solution looks like this:



In this design pattern:

1. An app accepts natural language input from a user.
2. A language model is used to determine semantic meaning (the user's *intent*).
3. The app performs an appropriate action.

**Azure AI Language** enables developers to build apps based on language models that can be trained with a relatively small number of samples to discern a user's intended meaning.

**Azure AI Language Service Features**

**Pre-configured Features**

These features do not require any model labeling or training:

* **Summarization**:
  + Summarizes documents and conversations into key sentences.
* **Named Entity Recognition (NER)**:
  + Extracts and identifies entities like people, places, or companies.
  + Example: "Seattle" in "The waterfront pier is my favorite Seattle attraction" is recognized as a location.
* **Personally Identifiable Information (PII) Detection**:
  + Identifies, categorizes, and redacts sensitive information such as email addresses, home addresses, IP addresses, names, and health information.
  + Example: "email@contoso.com" can be identified and redacted.
* **Key Phrase Extraction**:
  + Pulls out main concepts from text.
  + Example: "Azure AI Services" and "Text Analytics" from "Text Analytics is one of the features in Azure AI Services."
* **Sentiment Analysis**:
  + Determines the positivity or negativity of text.
  + Example: "Great hotel. Close to plenty of food and attractions we could walk to" is identified as positive.
* **Language Detection**:
  + Identifies the language of documents.
  + Example: "Bonjour" is identified as French.

**Learned Features**

These features require data labeling, training, and deploying a model:

* **Conversational Language Understanding (CLU)**:
  + Builds custom natural language understanding models to predict intents and extract information from utterances.
  + Requires tagged data for accurate prediction.
* **Custom Named Entity Recognition**:
  + Extracts specified entities from unstructured text using custom labeled data.
  + Example: Extracting involved parties from contract documents.
* **Custom Text Classification**:
  + Classifies text or documents into custom defined categories.
  + Example: Classifying news articles as News or Entertainment.
* **Question Answering**:
  + Provides answers to input questions using documents like FAQs or manuals.
  + Example: A virtual chat assistant using a company FAQ to answer common questions.

**Building a Conversational Language Understanding Model with Azure AI**

**Steps to Create and Use a Language Resource**

1. **Create Azure AI Language Resource**:
   * **Search** for Azure AI services in the Azure portal.
   * **Select** Language Service.
   * **Create** a new Language Service resource.
   * **Fill out** necessary details: region (closest geographically), unique name.
   * **Retrieve** the key and endpoint from the resource overview page (under Keys and Endpoint).
2. **Build, Train, and Deploy the Model**:
   * **Authoring**: Use the Azure AI Language resource for authoring your model.
   * **Prediction**: Process prediction requests from client applications using the same resource.

**Using Language Studio**

* **Visual Method**:
  + **Create** a Conversational language understanding project in Language Studio.
  + **Build, Train, Deploy**: Follow the same steps as above to build, train, and deploy your model using a visual interface.

**Using REST API**

* **Asynchronous Tasks**:
  + **Project Creation**: Submit a request to create a project.
  + **Import Data**: Send a request to import data.
  + **Training**: Submit a training job request.
  + **Deployment**: Submit a deployment job request and then check the status of the deployment job.

**Definitions of Intents, Utterances, and Entities**

**Intents:**

* **Definition**: Represent tasks or actions users want to perform, or the meaning behind an utterance.
* **Function**: Associated with one or more utterances to define the purpose of user input.
* **Example**:
  + GetTime
  + GetWeather
  + TurnOnDevice
  + None (for miscellaneous or out-of-scope utterances)

**Utterances:**

* **Definition**: Phrases entered by users when interacting with an application.
* **Function**: Examples of user input associated with specific intents.
* **Guidelines**:
  + Capture multiple variations of the same meaning.
  + Vary length and structure.
  + Use correct and incorrect grammar.
* **Example**:
  + "What time is it?"
  + "Do I need an umbrella?"
  + "Turn on the light."

**Entities:**

* **Definition**: Add specific context to intents, representing objects or concepts in utterances.
* **Types**:
  + **Learned Entities**: Associated with words or phrases in training utterances, allowing the model to learn matching elements during training.
  + **List Entities**: Have a specific set of possible values, often with synonyms.
  + **Prebuilt Entities**: Automatically detect common types such as numbers, dates, and names.
* **Function**: Enhance understanding by identifying and extracting relevant information.
* **Example**:
  + Learned Entity: DeviceName (e.g., "light", "fan")
  + List Entity: DayOfWeek (e.g., "Sunday", "Monday")
  + Prebuilt Entity: Organization (e.g., "Microsoft")

**Using Patterns to Differentiate Similar Utterances**

In scenarios where multiple intents have similar utterances, you can utilize patterns to disambiguate intents while minimizing the number of sample utterances. This approach helps the model correctly categorize intents based on the format and punctuation of the utterances. For example:

**TurnOnDevice:**

* "Turn on the {DeviceName}"
* "Switch on the {DeviceName}"
* "Turn the {DeviceName} on"

**GetDeviceStatus:**

* "Is the {DeviceName} on[?]"

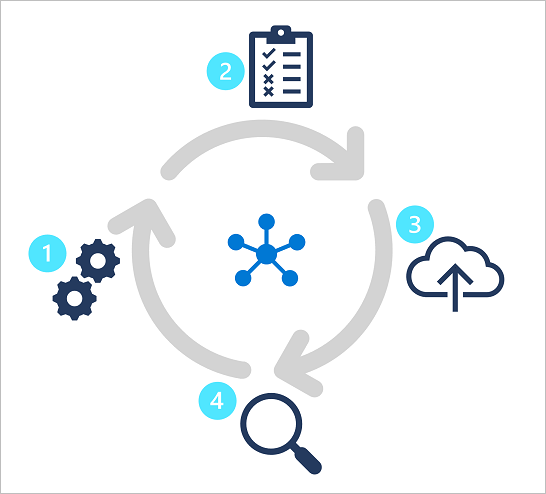
**TurnOffDevice:**

* "Turn the {DeviceName} off"
* "Switch off the {DeviceName}"
* "Turn off the {DeviceName}"

By providing examples of each intent with different utterance formats, the Azure AI Language service can effectively learn to differentiate between intents and categorize them accurately.

**Train, test, publish, and review a conversational language understanding model**

Creating a model is an iterative process with the following activities:



1. Train a model to learn intents and entities from sample utterances.
2. Test the model interactively or using a testing dataset with known labels
3. Deploy a trained model to a public endpoint so client apps can use it
4. Review predictions and iterate on utterances to train your model

By following this iterative approach, you can improve the language model over time based on user input, helping you develop solutions that reflect the way users indicate their intents using natural language.

Part of NLP is the ability to classify text, and Azure provides ways to classify text including sentiment, language, and custom categories defined by the user.

**Types of Custom Text Classification Projects in Azure AI Language Service**

Custom text classification projects in Azure AI Language Service involve assigning labels (classes) to text files. These projects can be categorized into two types:

1. **Single Label Classification:**
   * Each file can be assigned only one class.
   * Example: A video game summary could be classified as "Adventure" or "Strategy," but not both.
2. **Multiple Label Classification:**
   * Each file can be assigned multiple classes.
   * Example: A video game summary could be classified as "Adventure" and "Strategy."

**Key Differences Between Single and Multiple Label Projects**

**Labeling Data:**

* **Single Label Projects:**
  + Each file is assigned one class.
  + Simple labeling process.
* **Multiple Label Projects:**
  + Each file can be assigned multiple classes.
  + More complex labeling process.
  + Requires clear and well-distributed data for accurate model training.

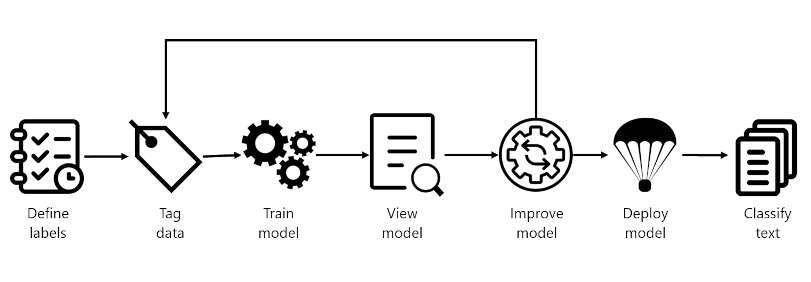
**Evaluating and Improving Your Model:**

* **Performance Metrics:**
  + **Recall:** Ratio of true positives to all actual labels.
  + **Precision:** Ratio of true positives to all predicted positives.
  + **F1 Score:** A balance of recall and precision.
* **Error Types:**
  + **False Positive:** Model predicts a label that isn't correct.
  + **False Negative:** Model fails to predict a correct label.
* **Improvement Strategies:**
  + **Single Label Projects:** Identify poorly classified classes and find more quality data for those classes.
  + **Multiple Label Projects:** More complex due to possible label combinations. Focus on high-quality and varied data for multiple label permutations.

**Summary:**

* **Single Label Classification** is simpler but restrictive in terms of assigning only one label per file.
* **Multiple Label Classification** offers flexibility to assign multiple labels but adds complexity in data labeling and model training.
* **Performance Metrics** such as recall, precision, and F1 score help in evaluating model performance.
* **Improvement** involves addressing specific classification errors by enriching the dataset with high-quality examples.

**Azure AI Language Project Life Cycle**



1. **Define Labels:**
   * Identify possible categories for your data.
   * Example: For video games, labels could be "Action", "Adventure", "Strategy".
2. **Tag Data:**
   * Label your existing data with the defined categories.
   * Ensure clear distinctions between labels to avoid ambiguity.
   * Example: Label "Quest for the Mine Brush" as "Adventure" and "Flight Trainer" as "Action".
3. **Train Model:**
   * Use labeled data to train your model.
   * This teaches the model to classify new data based on learned patterns.
4. **View Model:**
   * After training, review the model's performance.
   * Performance is scored between 0 and 1 based on precision and recall.
5. **Improve Model:**
   * Identify misclassified data and areas for improvement.
   * Add more examples and retrain to improve accuracy.
   * Example: Add more examples for "Adventure" and "Strategy" if they are often confused.
6. **Deploy Model:**
   * Once satisfactory, deploy your model to make it accessible via API.
   * Example: Deploy a model named "GameGenres".
7. **Classify Text:**
   * Use the deployed model to classify new text data.
   * Example: Use the API to classify new game summaries into genres.

**How to Split Datasets for Training**

* **Training Dataset:**
  + Used to train the model.
  + Should be about 80% of your labeled data.
* **Testing Dataset:**
  + Used to test the model after training.
  + Compares model predictions against actual labels to evaluate performance.

**Training Options**

* **Automatic Split:**
  + Azure randomly splits your data into training and testing datasets.
  + Suitable for larger, consistent datasets.
* **Manual Split:**
  + Manually specify which files are for training and which are for testing.
  + Useful for smaller datasets to ensure correct class distribution and variation.

**Deployment Options**

* **Multiple Models and Deployments:**
  + Allows creation of multiple models and deployments with unique names.
  + Benefits include testing models side by side, comparing dataset splits, and deploying multiple versions.

Custom named entity recognition (NER), otherwise known as custom entity extraction, is one of the many features for natural language processing (NLP) offered by Azure AI Language service. Custom NER enables developers to extract predefined entities from text documents, without those documents being in a known format - such as legal agreements or online ads.

An entity is a person, place, thing, event, skill, or value.

**Understand custom named entity recognition**

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

Custom NER, which is the focus of the rest of this module, 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.

Examples of when you'd want custom NER include specific legal or bank data, knowledge mining to enhance catalog search, or looking for specific text for audit policies. Each one of these projects requires a specific set of entities and data it needs to extract.

**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.

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.

A screenshot of a computer

Description automatically generated

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.

There are many commonly used languages throughout the world, and the ability to exchange information between speakers of different languages is often a critical requirement for global solutions.

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

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).

**Define custom translations**

While the default translation model used by Azure AI Translator is effective for general translation, you may need to develop a translation solution for businesses or industries in that have specific vocabularies of terms that require custom translation.

To solve this problem, you can create a custom model that maps your own sets of source and target terms for translation. To create a custom model, use the Custom Translator portal to:

* Create a workspace linked to your Azure AI Translator resource.
* Create a project.
* Upload training data files and train a model.
* Test your model and publish your model.
* Make translation calls to the API.

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

After creating your Azure 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.

**Using Azure AI Speech to Text API**

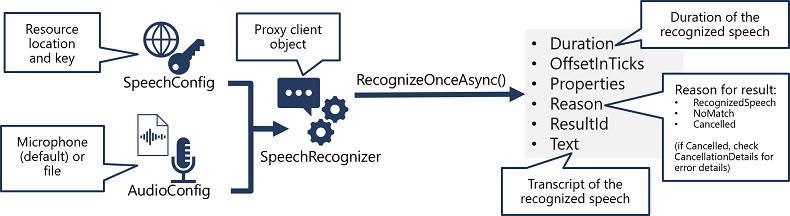
The Azure AI Speech service offers two REST APIs for speech recognition:

1. **Speech to Text API:** The primary API for performing speech recognition.
2. **Speech to Text Short Audio API:** Optimized for short audio streams (up to 60 seconds).

These APIs support both interactive speech recognition and batch transcription of multiple audio files. Detailed documentation for these REST APIs can be found in the Speech to Text REST API documentation. However, most applications use a language-specific SDK for ease of use.

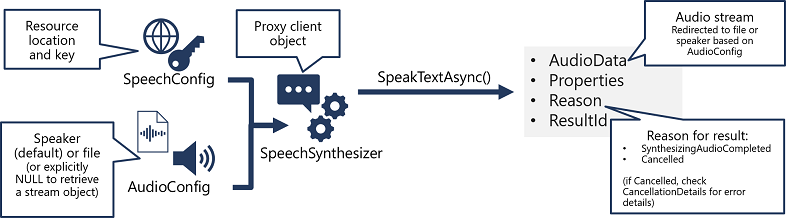
**Using the Azure AI Speech SDK**

The process for using the Speech to Text API with an SDK (such as Python, C#, etc.) generally follows this pattern:



1. **Create a SpeechConfig Object:**
   * This object contains information required to connect to your Azure AI Speech resource, such as the location and key.
2. **Create an AudioConfig Object (Optional):**
   * Defines the input source for the audio to be transcribed.
   * By default, this is the system's default microphone.
   * You can also specify an audio file as the input source.
3. **Create a SpeechRecognizer Object:**
   * Use the SpeechConfig and AudioConfig objects to create a SpeechRecognizer.
   * This object acts as a client for the Speech to Text API.
4. **Call API Functions Using SpeechRecognizer Methods:**
   * Example: RecognizeOnceAsync() method to asynchronously transcribe a single spoken utterance.
5. **Process the Response:**
   * The response is a SpeechRecognitionResult object with properties such as:
     + Duration
     + OffsetInTicks
     + Properties
     + Reason
     + ResultId
     + Text
   * If successful, the Reason property is RecognizedSpeech, and the Text property contains the transcription.
   * Other possible values for Reason include:
     + NoMatch: Audio parsed but no speech recognized.
     + Canceled: Error occurred (check Properties for CancellationReason).

**Key Points for Using the Azure AI Text to Speech API**

* **APIs Available:**
  + **Text to Speech API:** Primary API for speech synthesis.
  + **Batch Synthesis API:** For batch operations, converting large volumes of text to audio.
* **SDK vs. REST API:**
  + Most interactive applications use the Azure AI Speech SDK for convenience.
* **Steps to Implement Speech Synthesis Using Azure AI Speech SDK:**
* 
  + **Create SpeechConfig Object:**
    - Encapsulates information required to connect to Azure AI Speech resource.
    - Requires location and subscription key.
  + **Create AudioConfig Object (Optional):**
    - Defines the output device for synthesized speech.
    - Can be system speaker, an audio file, or null for processing the audio stream directly.
  + **Create SpeechSynthesizer Object:**
    - Uses SpeechConfig and AudioConfig to act as a client for the Text to Speech API.
  + **Call Methods of SpeechSynthesizer:**
    - For example, SpeakTextAsync() method converts text to spoken audio.
  + **Process the Response:**
    - Response is a SpeechSynthesisResult object with properties:
      * AudioData
      * Properties
      * Reason
      * ResultId
    - Check if the Reason property is SynthesizingAudioCompleted for successful synthesis.
    - AudioData contains the audio stream.

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

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

**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.

**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.
* <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>

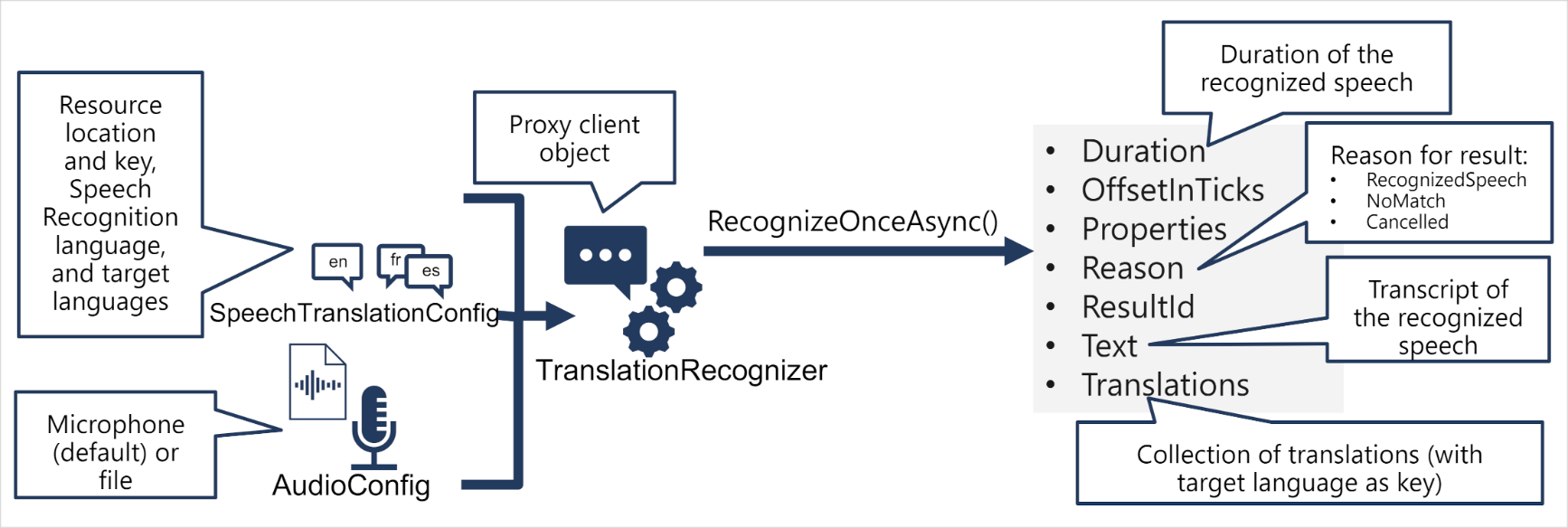
Translation of speech builds on speech recognition by recognizing and transcribing spoken input in a specified language, and returning translations of the transcription in one or more other languages.

After creating your Azure 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.

**Speech Translation Using Azure AI Speech SDK**

* **APIs and SDK:**
  + Azure AI Speech SDK is used for speech translation.
* **Steps to Implement Speech Translation:**



1. **Create SpeechTranslationConfig Object:**
   * Encapsulates information required to connect to Azure AI Speech resource.
   * Requires subscription key and region.
   * Specifies source language (input speech) and target languages (translation).
2. **Create AudioConfig Object (Optional):**
   * Defines the input source for the audio.
   * Default is the system microphone, can also be an audio file.
3. **Create TranslationRecognizer Object:**
   * Uses SpeechTranslationConfig and AudioConfig to act as a client for the Speech translation API.
4. **Call Methods of TranslationRecognizer:**
   * For example, RecognizeOnceAsync() method translates a single spoken utterance.
5. **Process the Response:**
   * Response is a SpeechRecognitionResult object with properties:
     + Duration
     + OffsetInTicks
     + Properties
     + Reason
     + ResultId
     + Text (transcription in the original language)
     + Translations (dictionary of translations)

**Synthesizing Translations in Azure AI Speech SDK**

**Goal:** Convert translated text into spoken audio, creating a speech-to-speech translation solution.

**Methods:**

1. **Event-based Synthesis:** For 1:1 translation (source language to single target language).
2. **Manual Synthesis:** For translating into multiple target languages.

**Event-based Synthesis**

**Steps:**

1. **Specify Desired Voice:**
   * Set the desired voice in the TranslationConfig.
2. **Create Event Handler:**
   * Create an event handler for the TranslationRecognizer's Synthesizing event.
   * Use the GetAudio() method to retrieve the byte stream of translated audio.

**Manual Synthesis**

**Steps:**

1. **Translate Spoken Input:**
   * Use TranslationRecognizer to get text transcriptions in target languages.
2. **Iterate Through Translations:**
   * Use SpeechSynthesizer to synthesize audio for each target language.

 **Event-based Synthesis:** Best for single target language translation with real-time synthesis.

 **Manual Synthesis:** Flexible for multiple target languages, iterating over translations and synthesizing each one separately.