# AZURE AI SERVICES

SUCCINCTLY®

BY ALESSANDRO DEL SOLE

SUCCINCTLY® EBOOK SERIES

Syncfusion®

# Azure AI Services Succinctly

**Alessandro Del Sole**

Foreword by Daniel Jebaraj

2501 Aerial Center Parkway
Suite 111
Morrisville, NC 27560
USA

ISBN: 978-1-64200-248-5



**Technical Reviewer:** James McCaffrey

**Copy Editor:** Courtney Wright

**Acquisitions Coordinator:** Graham High, content team lead, Syncfusion, Inc.

**Proofreader:** Jacqueline Bieringer, content producer, Syncfusion, Inc.

# Table of Contents

# The *Succinctly* Series of Books

Daniel Jebaraj
CEO of Syncfusion, Inc.

When we published our first *Succinctly* series book in 2012, *jQuery Succinctly*, our goal was to produce a series of concise technical books targeted at software developers working primarily on the Microsoft platform. We firmly believed then, as we do now, that most topics of interest can be translated into books that are about 100 pages in length.

We have since published over 200 books that have been downloaded millions of times. Reaching more than 2.7 million readers around the world, we have more than 70 authors who now cover a wider range of topics, such as Blazor, machine learning, and big data.

Each author is carefully chosen from a pool of talented experts who share our vision. The book before you and the others in this series are the result of our authors' tireless work. Within these pages, you will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

We are absolutely thrilled with the enthusiastic reception of our books. We believe the *Succinctly* series is the largest library of free technical books being actively published today. Truly exciting!

Our goal is to keep the information free and easily available so that anyone with a computing device and internet access can obtain concise information and benefit from it. The books will always be free. Any updates we publish will also be free.

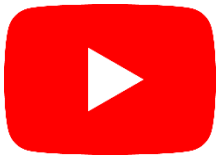
**Let us know what you think**

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctlyseries@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on social media and help us spread the word about the *Succinctly* series!



# About the Author

Alessandro Del Sole is a Xamarin Certified Mobile Developer and former Microsoft MVP. Awarded MVP of the Year in 2009, 2010, 2011, 2012, and 2014, he is internationally considered a Visual Studio expert and .NET authority. Alessandro has authored many printed books and ebooks on programming with Visual Studio, including *.NET MAUI Succinctly*, *Visual Basic 2015 Unleashed*, and *Xamarin.Forms Succinctly*. He has written many technical articles about .NET, Visual Studio, and other Microsoft technologies in Italian and English for many developer portals. He has been a frequent speaker at Italian conferences, and he has also produced many instructional videos in both English and Italian.

Alessandro works as a senior software engineer for Fresenius Medical Care, focusing on building mobile apps with .NET MAUI in the healthcare market. You can follow him on LinkedIn, and support him with a coffee.

# Introduction

Artificial intelligence (AI) is revolutionizing every aspect of modern life, transforming industries, businesses, and daily experiences. This technology is dramatically accelerating business processes, and companies that are not embracing AI will soon put their businesses at risk. As one of the most important technological advancements of the last decades, AI is enabling machines to learn from data and recognize patterns. It may soon make decisions with minimal human intervention. This capability is driving innovation across sectors, from healthcare and finance to manufacturing and entertainment. There are many impacts in terms of automation and efficiency because AI systems can automate routine and complex tasks, freeing up human work for more creative and strategic roles. In industries like manufacturing, logistics, and customer service, AI-powered robots and virtual assistants are boosting productivity and efficiency. This is strictly connected with the healthcare industry, where AI is enabling early diagnosis, personalized treatments, and better patient care. AI-driven tools analyze medical images, predict patient outcomes, and even assist in drug discovery, ultimately saving lives and reducing costs.

Not limited to this, AI's ability to process massive amounts of data in real time is helping organizations make smarter, data-driven decisions. In finance, AI-driven algorithms detect fraud, optimize trading strategies, and assess credit risk with unparalleled speed and accuracy. All these aspects together help deliver improved customer experiences, which are enhanced with chatbots, recommendation systems, and personalized marketing.

Cloud platforms like Microsoft Azure are making AI accessible to businesses and developers of all sizes. Azure AI services fit into the broader AI era by offering powerful tools and services that allow organizations to take advantage of AI without needing deep expertise in machine learning or data science. This is called AI democratization, and Microsoft Azure brings AI services to everyone by providing prebuilt models, APIs, and development tools. This allows developers to integrate AI capabilities like computer vision, natural language processing, and speech recognition into their applications easily. Azure’s cloud infrastructure ensures that these AI services can scale to meet the demands of businesses, from startups to enterprises.

Additionally, Azure AI services seamlessly integrate with the Microsoft ecosystem, including Office 365, Dynamics 365, and the full Azure offering. This integration allows businesses to leverage AI within familiar environments, enhancing existing workflows and systems with AI capabilities. As AI continues to shape the future, platforms like Azure AI services are playing a critical role in making AI accessible and impactful.

Using Azure AI services can dramatically increase the power and effectiveness of your applications; however, you need to make important considerations. Like with any other cloud-based service, applications must be connected to the internet, so you establish an external dependency that is not under your control, and you need to carefully evaluate the benefits compared to the costs.

This ebook provides an overview of the Microsoft Azure AI services suite from the point of view of developers working with .NET. Since .NET is available on multiple operating systems, the examples will be discussed using Visual Studio Code. Obviously, you are totally free to use Microsoft Visual Studio 2022 if you work on Windows. The companion source code is available on GitHub.

## Assumptions about the reader

This ebook is for software developers working with .NET and the Microsoft stack. Ideally, the reader should have intermediate knowledge of .NET technology and good knowledge of at least one of the Microsoft development platforms based on the Extensible Application Markup Language (XAML), such as Windows Presentation Foundation (WPF), .NET MAUI, or Universal Windows Platform. This is because most of the code examples will be created using WPF to provide a convenient user interface that makes it easier to load files that will be analyzed by AI services and to display the analysis result.

However, the focus of this publication is Azure AI services, so the explanations of the code will explicitly target this topic. It's not possible to also provide the basics of WPF and of other .NET common types, except where strictly required to explain concepts related to AI services.

Syncfusion® offers a full collection of free ebooks, including *C# Succinctly* and *WPF Succinctly*, which you can use as a reference when something is not clear. Also, you should already have basic knowledge of NuGet, the package manager for .NET, which will be widely used across this ebook to install libraries from the Azure SDK.

# Chapter 1 Introducing Azure AI Services

Microsoft Azure AI services, formerly Azure Cognitive services, provide a set of APIs and tools that allow developers to add intelligent features to their applications without needing in-depth knowledge of AI or data science. These services cover various aspects of artificial intelligence, such as vision, speech, language, and decision-making. By abstracting the complexity of AI development, Azure AI services make it easier to integrate AI capabilities into applications, offering services that are reliable, scalable, and easy to use. This chapter provides an overview of the services that will be covered in detail in the next chapters.

## Overview of Azure AI services

Azure AI services include a wide range of AI capabilities integrated into the Microsoft Azure cloud platform, as well as into Microsoft 365 and other products. The goal of these services is to democratize AI, making advanced tools available to enterprises, developers, and users, regardless of their technical expertise. Microsoft's AI solutions are built on Azure, its cloud computing platform, leveraging deep learning, machine learning, and cognitive services to drive automation, insights, and enhanced productivity.

These services are designed to address diverse needs, from natural language processing (NLP) and computer vision to conversational AI, data analytics, and custom AI model development. They target different industries such as healthcare, finance, retail, and manufacturing, offering solutions that can integrate with existing business processes.

The Azure offerings regarding AI are very large, and not all of the platforms can be covered in this ebook. The next paragraphs provide an introduction to the Azure AI services that will be covered in the coming chapters.

### The Microsoft Responsible AI Standard and Principles

Artificial intelligence is extremely powerful, and it can help solve business problems in a much faster way. The applications of AI can target different critical sectors—not only in people's daily lives, but also in most countries' public services. In fact, AI is already present in many processes of healthcare systems, military and defense systems, transportation, politics, and so on.

On the other hand, democratized AI is also available to people that do not have good intentions. As a common example, generative AI can be easily used to create completely fake content and news to influence the masses.

For this reason, Microsoft has developed the Responsible AI Standard and the Responsible AI Principles, a set of guidelines for companies and individuals that take advantage of AI in their applications. You can find the details of these principles on the official page, but the summary is: make responsible use of AI in your applications and do not use AI in a way that can violate the law or the privacy of the people.

When setting up Azure AI services in the Azure Portal, for some of the services (such as Computer Vision) you will be asked to accept a notice of responsible use of AI so that you take responsibility for the use you make of Microsoft services.

## Azure AI Search

Azure AI Search is a cloud-based search service that adds AI-driven search capabilities to your applications. It combines powerful indexing, natural language processing, and AI enrichment capabilities to provide relevant and personalized search results. It provides indexing to extract insights from unstructured content, such as identifying key phrases, language detection, and sentiment analysis. It also offers full control over the ranking, filtering, and faceting of search results; and it implements semantic search capabilities to understand user intent and context. This allows for more accurate and nuanced search results, similar to web search engines. Chapter 3 discusses this service and provides examples that will help you get started.

## Azure AI Language services

Language services use natural language processing (NLP) to implement conversational user interfaces, so that users can interact with applications in the same way they would do with other people. Language services include the following:

- Language Understanding (LUIS): Builds NLP models to understand user intentions.
- Text analytics: Extracts insights from text, such as sentiment analysis, key phrase extraction, and entity recognition.
- Translator: Provides machine translation services for multiple languages.
- QnA Maker: Creates a question-and-answer layer over your data, such as FAQs or manuals.

The Language understanding (LUIS) and QnA Maker are very complex services, and vast topics that cannot be summarized with a few paragraphs here. For this reason, this ebook only covers the Text Analytics and Translator services in Chapter 4.

## Azure AI Document Intelligence

Azure AI Document Intelligence services (formerly known as Form Recognizer) use AI to automate data extraction from documents, such as forms, invoices, receipts, and contracts. These services can process both structured and unstructured data, transforming documents into usable data quickly and accurately. It features prebuilt models to extract key-value pairs easily from common document types like invoices, receipts, and business cards.

In addition, it provides the possibility to create and train custom models to handle unique document types or industry-specific forms. Finally, Document Intelligence services can automatically extract and structure tables from documents and recognize and process documents in multiple languages, making it suitable for global use cases.

Chapter 5 describes this service thoroughly and shows how to build an application that analyzes invoices.

## Azure AI Decision services

Azure AI Decision services allow for analyzing data and contents in order to provide recommendations based on the criteria provided by developers. More specifically, the following services are available:

- Personalizer: Delivers personalized user experiences by making recommendations.
- Content Safety: Automatically filters inappropriate content in text, images, and videos.
- Anomaly Detector: Detects anomalies in time-series data to identify potential problems early.

Actually, Anomaly Detector and Personalizer are scheduled to be retired in late 2026, and it is not possible to create new resources. Their successors have not been announced at the time of writing. For these reasons, they will not be covered here; instead, we will cover the Content Safety service in Chapter 6.

## Azure AI Translator

Azure AI Translator is a cloud-based machine translation service that supports real-time language translation for more than 100 languages. It enables businesses and developers to translate text, documents, or even entire websites, providing localized content and fostering better communication. It features instantaneous translation for text, documents, and speech, and it allows translating entire documents while preserving the original format, such as Word, PDF, and PowerPoint.

In addition, it is also possible to create custom translators to fine-tune translation models with domain-specific data to improve accuracy and relevance. Chapter 7 provides detailed information on this service, and it explains how to translate both text and formatted documents, leveraging other Azure services, such as the Blob Storage.

## Azure AI Speech services

Azure AI Speech services allow us to work with spoken text and language translation. The following services are included:

- Speech-to-Text: Converts speech into text in real-time.
- Text-to-Speech: Synthesizes speech from text, with customizable voices.
- Speech Translation: Provides real-time speech translation.
- Speaker Recognition: Identifies and authenticates speakers based on their voice.

Chapter 8 provides detailed explanations and code examples for each feature.

## Azure AI Computer Vision

Azure AI Computer Vision services allow for image, video, and form analysis. More specifically:

- Computer Vision: Provides tools for image analysis, including object detection, text recognition (OCR), and content moderation.
- Face API: Detects and recognizes human faces, along with facial attributes such as emotions, age, and gender.
- Custom Vision: Allows developers to create their own image classification models.

Custom Vision is actually another vast topic, and it is not possible to cover it in detail in this ebook. If you want to create custom vision models, refer to the official documentation. Due to the Face API's ability to identify biometric human characteristics, Microsoft decided to open it only to customers and organizations that submit a registration form and comply with specific legal requirements. All the necessary information for it can be found on the Limited Access to Face API page.

As a consequence, this ebook will only describe the source code to implement Face API, but no image will be analyzed. In summary, Chapter 9 describes Computer Vision in detail, but code examples only relate to image analysis and OCR.

## Other AI services

The Azure AI services suite includes additional services that will not be covered in this ebook for different reasons explained later. The following paragraphs summarize the other available services.

### Azure OpenAI service

Azure OpenAI service offers access to advanced language models developed by OpenAI, such as GPT (generative pretrained transformer) models. This service enables developers to leverage state-of-the-art natural language understanding and generation capabilities for various applications, including chatbots, content generation, summarization, and code completion. These include GPT models, which allow the deployment of language models capable of generating human-like text based on input prompts, used in a range of scenarios like writing, summarization, and customer interaction.

***Note: Syncfusion has published a dedicated ebook about the OpenAI service, so this will not be covered here.***

### Azure AI Bot service

The AI Bot service is the most recent update in the Microsoft Azure offering related to building conversational applications. With this service, you can build chatbots that leverage the most sophisticated Azure AI-driven technologies. This service is quite complex, and it is not possible to summarize its opportunities in one chapter. For this reason, it is not discussed in this ebook.

## Programming with Azure AI services

Azure AI services are exposed through RESTful APIs, so it is possible to send web requests and receive a response over the HTTPS protocol. However, there are many convenient client libraries on the market, developed to make it easier to work with Azure resources from various development platforms and programming languages.

Developers working with Microsoft technologies can leverage the Azure SDK, a collection of .NET libraries available via NuGet that allow for interacting with all the Azure cloud services via .NET and managed code.

In this ebook, you will work with C#, .NET, and the Azure SDK to interact with the Microsoft Azure AI services. The appropriate libraries from the Azure SDK will be mentioned as required. In the next chapter, you will instead set up your development environment to have all the necessary common .NET tools.

## Chapter summary

Azure AI services provide a robust suite of AI-driven tools that enable developers to integrate sophisticated intelligence into their applications. By leveraging these services, businesses can enhance their capabilities in global communication, intelligent search, automated data processing, natural language understanding, computer vision, and content moderation. These services streamline workflows, reduce operational complexity, and allow for the seamless integration of AI into various business processes, from customer support and document management to content safety and localization.

Before writing code to fully understand the potential, you need to set up your development environment. This is discussed in the next chapter.

# Chapter 2 Setting Up the Development Environment

This chapter explains how to set up the development environment to build .NET applications that use Microsoft Azure AI services, regardless of the computer type and operating system you use. In fact, .NET runs on Windows, macOS, and Linux systems, so the steps described in this chapter apply to all systems. Figures in the ebook are based on Windows, but you will get the same results on other operating systems.

***Note: If you work with Microsoft Visual Studio 2022 on Windows and prefer to use this environment rather than VS Code, you are totally free to do so. The choice of Visual Studio Code in this ebook aims to reach the widest audience possible.***

## Registering for an Azure subscription

To set up a development environment for Microsoft Azure AI services using Visual Studio Code, the first requirement is having a Microsoft Azure subscription. If you do not have one, you can register for a 30-day trial. You will need to register using a Microsoft account.

Complete the registration process by providing your details, verifying your identity with a credit card (you will not be charged unless you upgrade your subscription), and accepting the terms. Once you register, you will receive $200 in free credit, which can be used for various Azure services, including Azure AI services. This is a perfect start to walk through the examples described in this ebook. If you create a free Azure subscription for the purposes of following along in this ebook, it's relatively easy to cancel the subscription.

For now, nothing else is required on the Azure platform. All the required configurations and service generations through the Azure Portal user interface will be discussed where appropriate in the next chapters.

### Locating the Azure AI services

Once you have created an Azure subscription, you need to log in to the Azure Portal, which is the place where you manage all the available Azure services, not just AI services. The main page of the Azure Portal is a dashboard where you find shortcuts to the most popular Azure services. You will find a shortcut called **AI Services**. If you click on this shortcut, you will access a page that contains the full list of available Azure AI services, as shown in Figure 1.

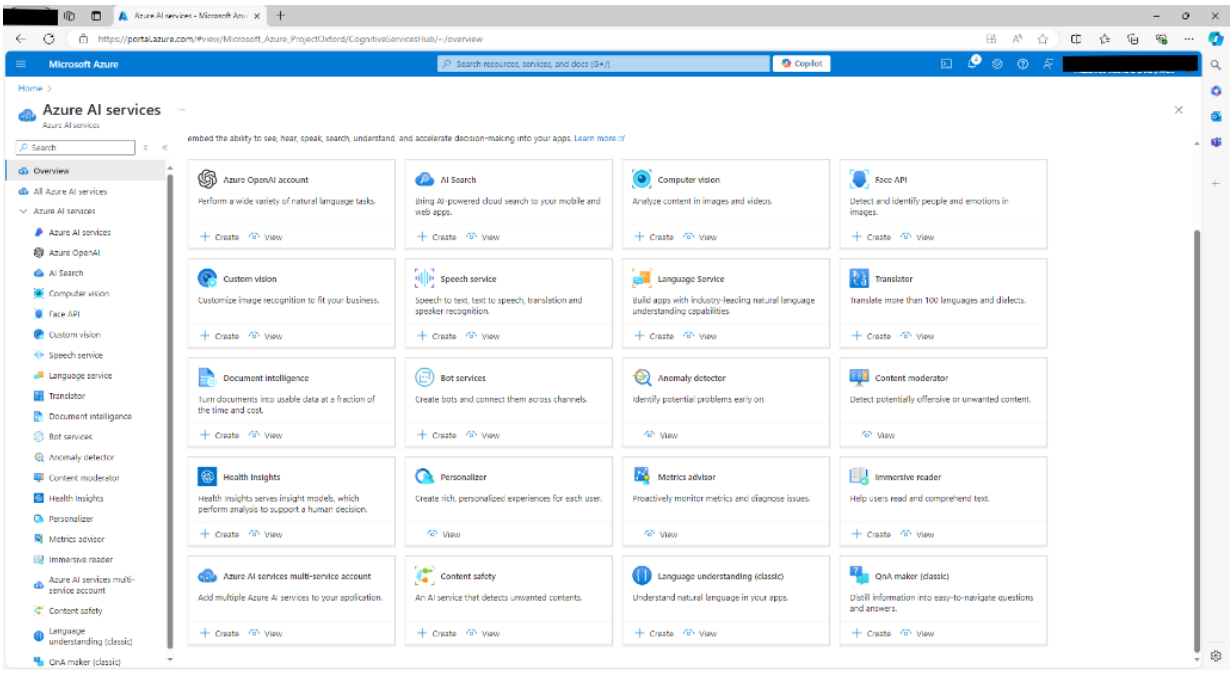



*Figure 1: Full list of Azure AI Services*

In the next chapters, you will be asked to create new service instances for the AI service targeted by each chapter. Keep Figure 1 as a reference to find the discussed service quickly and remember that you can access this page by clicking **AI Services** in the Azure Portal home page.

## Creating a resource group

As the name implies, an Azure resource group is a container for cloud services. It provisions all the resources common to the services it contains. A resource group is also needed to complete the code examples discussed in this ebook.

To accomplish this, once you have logged in to the Azure Portal, type **Resource Group** in the search bar and click the **Resource Groups** item that appears. This will open the Resource groups page, as shown in Figure 2.

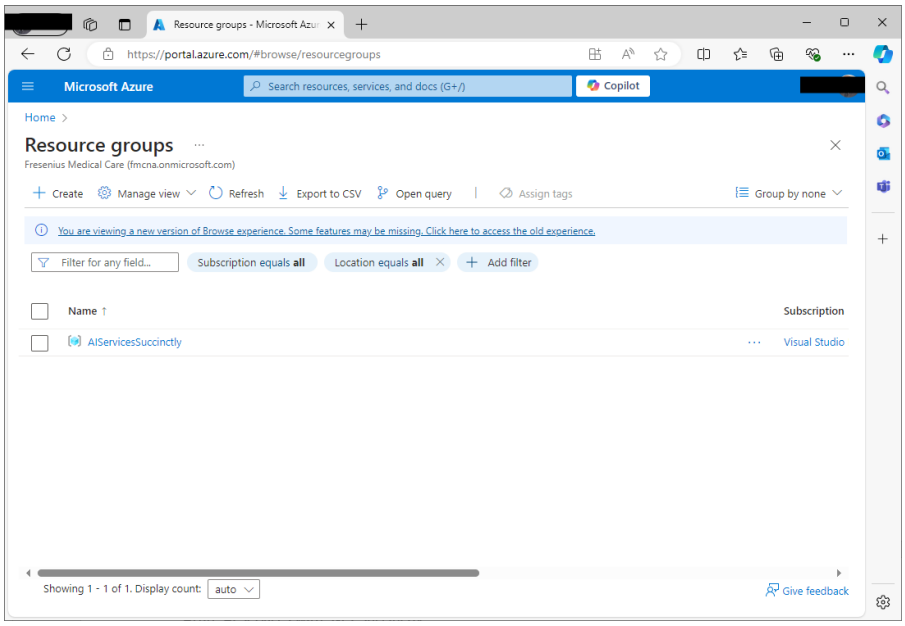



*Figure 2: Locating resource groups*

Next, click **Create**. In the Create a resource group page, select your Azure subscription, then enter a name for the resource group, such as aiservicessuccinctly, keeping the name lowercase.

> ***Note: Every time you specify a name for an Azure resource, it must be lowercase. Optionally, you can append the -rg literal to the resource group names to distinguish them from other Azure resources.***

Figure 3 demonstrates this.

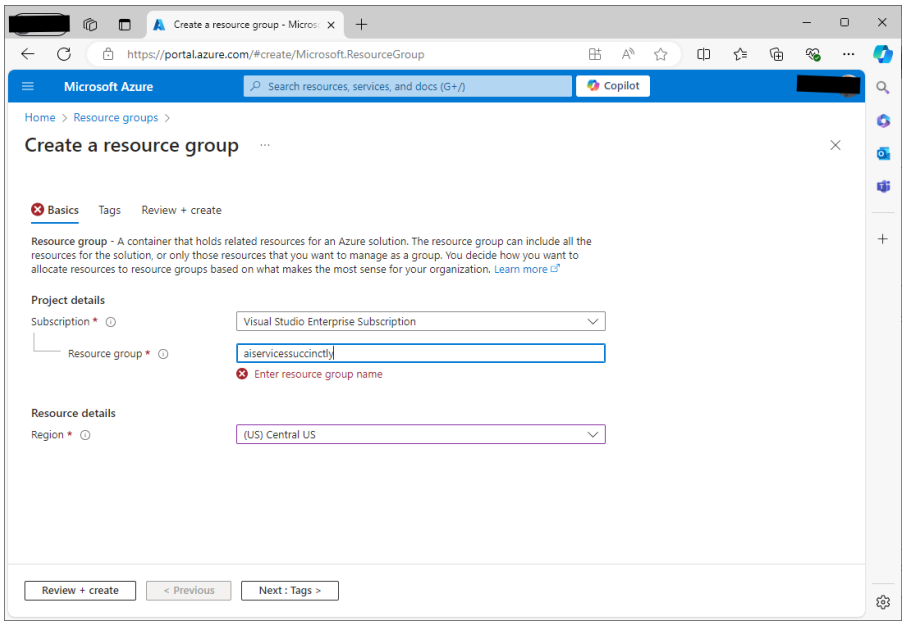



*Figure 3: Creating a resource group*

In the Region dropdown, select the closest region to your location. When ready, click **Review + Create** > **Create**, and wait for the resource group to be deployed.

***Note: To avoid extra costs or credit consumption, remember to completely delete the Azure resources that you no longer use.***

# Installing .NET and .NET SDK

***Note: If you have already installed .NET 8 on your machine, you can skip this step.***

In Chapter 1, you learned that Azure AI services are available as RESTful APIs and that Microsoft provides the Azure SDK to work against such services with convenient .NET client libraries. The goal of this ebook is leveraging the Azure AI services in C# and .NET, so before taking advantage of the Azure SDK, you need to set up your development machine to have all the necessary .NET tools.

The next step is installing the latest stable release of .NET and the .NET software development kit (SDK) on your development machine. The SDK also includes the .NET command line interface (CLI), which is required to launch command lines from a command prompt. At the time of writing this, the current stable release is .NET 8. You can download the installer from the official download page, obviously making sure you select the installer that targets your system.

Launch the installer and follow the on-screen instructions. When the installation is complete, you can open a command prompt (or Terminal, on macOS and Linux) and type the following command line:

```
> dotnet --version
```

If the installation was successful, this line will display the current .NET version number.

## Installing and configuring Visual Studio Code

Visual Studio Code (often shortened to VS Code or Code) is a very popular, open-source, cross-platform evolved code editor that works well with Azure services, as well as with others. If you have not installed VS Code already, go to the official download page and download the appropriate installer for your operating system (Windows, macOS, or Linux). Launch the installer and follow the on-screen instructions. When you're ready, launch VS Code.

> ***Note: It is not possible to summarize all the powerful features of Visual Studio Code in this publication, which has a different focus. For this reason, we'll only discuss the features that are required to set up the environment and understand the examples. For further information, you can read Visual Studio Code Succinctly.***

The final step is installing an extension for Visual Studio Code called C# DevKit, which extends the development environment with a rich C# development experience and a debugger for .NET. Open the **Extensions** view, and search for the **C# DevKit** extension, as shown in Figure 4.

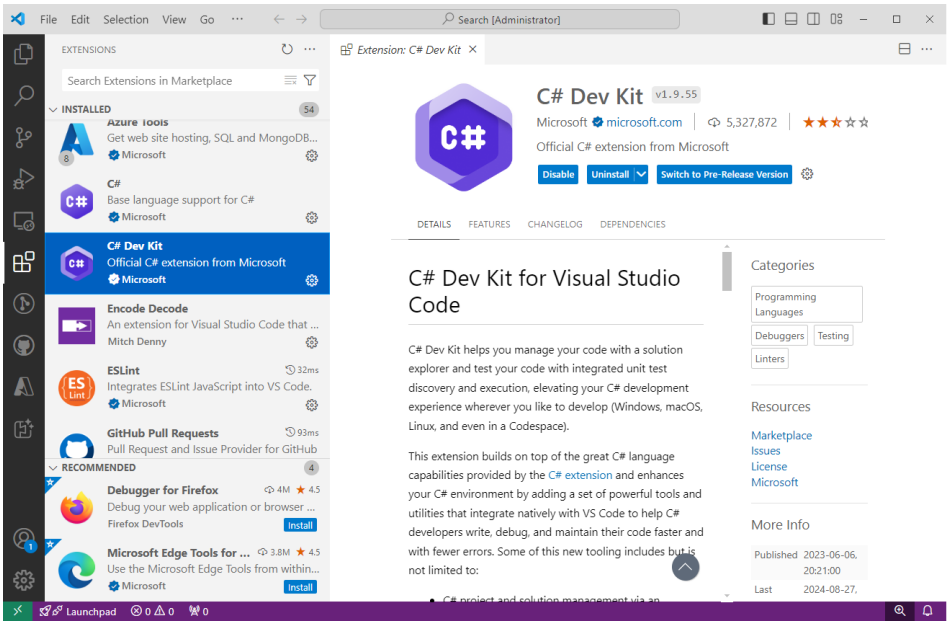



*Figure 4: Installing the C# DevKit extension for VS Code*

Click **Install** and close the Extensions view when you're ready.

***Note: Microsoft has published many Visual Studio Code extensions that make it easier to work against Azure resources from within the development environment. However, these extensions do not currently target Azure AI services, which is why we aren't discussing them here.***

# Additional configuration

The last step of your local configuration involves creating a new folder that will contain all the code examples. If you want to be consistent with this ebook, you can name the folder **AIServices**, but you are free to choose a different name. Using a command prompt or a Terminal instance in VS Code, you can simply create the new folder as follows:

```
> md c:\AIServices
```

Now you are ready to start writing intelligent apps with .NET.

## Creating applications with the .NET CLI

The .NET command line interface (CLI) provides a system-agnostic way to create and manage .NET applications. In the next chapters, you will create two types of .NET apps: console apps and WPF apps. The following command line generates a new C# project for a console app:

```
> dotnet new console
```

The generated project takes the name from the current folder. The following command line, instead, generates a new C# project for a WPF app:

```
> dotnet new wpf
```

***Note: When you create a C# project with the .NET command line, the tool also generates a Visual Studio solution (.sln) file whose name consists of the project name, plus the `.generated` literal. This allows you to also open the project in Visual Studio 2022. If you instead work with VS 2022, the solution name and main project name are the same.***

The steps you will follow to create sample projects are the following:

1. Create a new subfolder for the new project inside the AIServices folder created previously.
2. Make the new subfolder the current folder.
3. Generate a new project.
4. Install the necessary libraries from the Azure SDK in the form of NuGet packages.

For example, the following sequence of commands is used in Chapter 3 to create a sample WPF project:

```
> md c:\AIServices\AppSearchWpfApp
> cd c:\AIServices\AppSearchWpfApp
> dotnet new wpf
> dotnet add package Azure.Search.Documents
> dotnet add package Newtonsoft.Json
```

The **`dotnet add package`** command is used to install NuGet packages in the project. More specifically, the command installs the latest version available. If you need to install a specific version, you can use the following command:

```
> dotnet add package PackageName -v 1.0.0
```

**`PackageName`** is the name of the NuGet package, and **`1.0.0`** will be replaced with the version number you need.

## Opening projects in Visual Studio Code

Once you have created a project, open Visual Studio Code and then select **File** > **Open Folder** or click **Open Folder** in the Start page. Select the folder that contains the project you just created and wait for it to be available in VS Code.

Remember that VS Code is folder-based, not project-based. So, you do not open a C# project file (.csproj) or Visual Studio solution file (.sln) directly; instead, you open the folder that contains the project. Visual Studio Code will then set up the environment accordingly. We'll cover more details later in this book.

## Common errors and exceptions

When you work with Azure AI services from .NET, you can receive exceptions and error messages if something goes wrong. Some exceptions are common to all the services, and others are specific. Table 1 describes exceptions that are common to all the services.

*Table 1: Common AI Services .NET exceptions*

| | |
|---|---|
| **`AuthenticationFailedException`** | This occurs when the provided API keys or tokens are invalid, expired, or missing. The service will not be able to authenticate your request, and you'll receive this error in response. |

| | |
|---|---|
| **`ServiceRequestException`** | This occurs when there's an issue with the request to the Azure service, such as malformed payloads or incorrect endpoint usage. It's a general exception thrown for request validation errors. |
| **`RequestFailedException`** | This is a general exception class used across Azure SDKs when a request to the server fails for any reason (e.g., network failure, server issues, or client misconfiguration). |
| **`TimeoutException`** | If the service takes too long to respond or the network request times out, this exception is thrown. This can be caused by network issues or server-side delays. |

You can surround the code that interacts with Azure AI services with a **`try..catch`** block and catch and handle the most appropriate exceptions. Other exceptions that are specific to each service will be discussed where appropriate.

## Chapter summary

In this chapter, you have seen how to configure the development environment to work with .NET, Visual Studio Code, and the Microsoft Azure AI services. On the Azure side, you have seen how to access the Azure Portal, where to locate the Azure AI services, and how to create resource groups. On the desktop side, you have seen how to set up Visual Studio Code as the development environment and how to leverage the .NET CLI to create Console and WPF applications.

Now you have everything you need, and you are ready to start building intelligent applications.

# Chapter 3 Azure AI Search

The Microsoft Azure AI Search is a cloud-based search service with built-in AI capabilities that enrich the information retrieval experience. AI Search goes beyond traditional keyword-based search by integrating cognitive skills, such as natural language processing, image recognition, and machine learning, to provide more intelligent and context-aware search results. This chapter discusses AI Search in a way that makes it easier to understand its purpose.

## Introducing Azure AI Search

Microsoft Azure AI Search is a cloud-based search service that integrates AI capabilities to offer advanced search features. It provides an enriched search experience by combining traditional full-text searching with advanced natural language processing (NLP), cognitive skills, and machine learning models. Azure AI Search can be used to develop custom search solutions that offer users accurate, fast, and meaningful search results across various data types such as structured data, unstructured text, and media files.

Azure AI Search is particularly useful for organizations that need to search large datasets efficiently. It can be employed in scenarios such as enterprise search, e-commerce, catalog search, document discovery, and knowledge management. It is built on the foundation of modern search technology, powered by Elasticsearch, and enhanced with Microsoft's proprietary AI services to make sense of complex datasets.

Azure AI Search is a fully managed service, which means it abstracts the complexities of setting up, maintaining, and scaling search infrastructure. The core functionality of Azure AI Search revolves around indexing, querying, and refining data to make it searchable in a way that aligns with user intent. Key features include:

- **Search indexing**: Indexes can be created from multiple data sources such as Azure SQL Database, Cosmos DB, Blob Storage, and custom data sources. During the indexing process, data is transformed and enriched using AI capabilities, such as language detection, image analysis, and entity recognition.
- **Search queries**: The service supports powerful query syntax that allows for full-text search, filtering, faceting, and sorting. It also provides suggesters and autocomplete features to improve the search experience.
- **Cognitive skills**: Through integration with other Azure AI services, you can enrich search indexes with insights extracted from raw content, such as detecting named entities, sentiment analysis, image extraction, key phrases, and more. These are known as cognitive skills, which are applied to the data during the indexing process.
- **Security and scalability**: The service supports enterprise-grade security with Azure Active Directory (Azure AD) integration, role-based access control (RBAC), and encryption of data at rest and in transit. It is also highly scalable, allowing you to adjust resources based on your search workload.

Azure AI Search includes several subservices and components that work together to enable comprehensive search capabilities:

- **Indexers** automate the process of ingesting data from various sources into the search index. Indexers can connect to multiple Azure data sources like Azure SQL Database, Cosmos DB, and Blob Storage, or they can use a custom data source. You can schedule indexers to update the search index periodically as the underlying data changes. During the indexing process, AI enrichment can be applied via cognitive skills.
- **Cognitive skills** allow you to enrich your data using various AI services. These skills range from text analysis, image recognition, and entity extraction to more advanced tasks such as document translation. These skills are applied to documents or text as they are indexed, transforming raw content into searchable and insightful information.
- **Synonym** maps allow you to define lists of equivalent terms for search queries. For example, car and automobile can be treated as synonyms.
- **Analyzers** determine how data is tokenized and processed during indexing and querying. They support different languages and customizations, helping to improve search relevance.
- **Search units** are the scalable compute resources of Azure AI Search. Based on your workload and performance requirements, you can scale search units up or down. Search units handle query processing, indexing, and storage operations. They ensure that the service can accommodate large volumes of data and provide rapid query responses.

Each of these components works together to build a sophisticated search solution that can handle complex search scenarios with ease.

# Creating a sample application

You will now create a Windows Presentation Foundation (WPF) application that allows users to search a dataset using Azure AI Search. The app will interact with a locally created dataset (a JSON file) that simulates an e-commerce product catalog. The purpose of this example is to demonstrate how to query the Azure AI Search service, retrieve results, and display them in the app. You will also see how to implement faceted search, filtering, and autocomplete.

Generally speaking, creating applications that work against Azure AI services first requires generating the appropriate cloud resources in the Azure Portal. You will repeat almost the same steps through all the next chapters, just targeting different services, so it is important that you follow the next steps with particular attention.

***Note: Most of the examples in this ebook are represented by WPF applications. This allows for implementing a convenient user interface that simplifies the management of data and files, making it more interactive compared to Console apps.***

## Creating the Azure resources

Before writing code, you need to set up the necessary Azure resources, which include an instance of the Azure AI Search service and a search index. Since this is the first time you will create an instance of an AI service, the required steps will be explained in more detail. In the next chapters, they will be summarized, and only the service name will be highlighted.

Once you're logged into the Azure Portal, you will see the Azure dashboard, as shown in Figure 5.

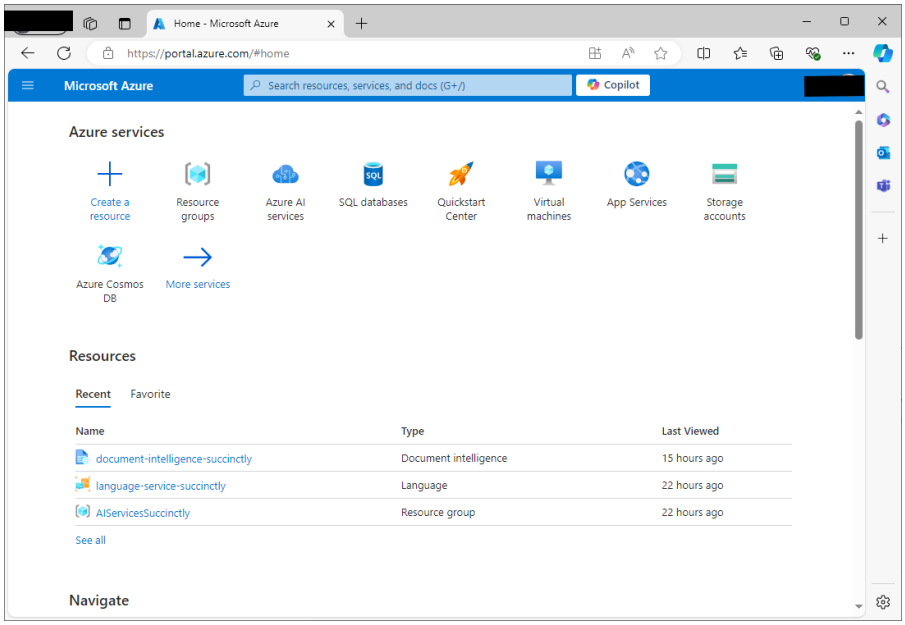



*Figure 5: The dashboard of the Azure Portal*

At this point, click **AI Services**. This will open the full list of available Azure AI services, including those not discussed in this ebook (see Figure 6).

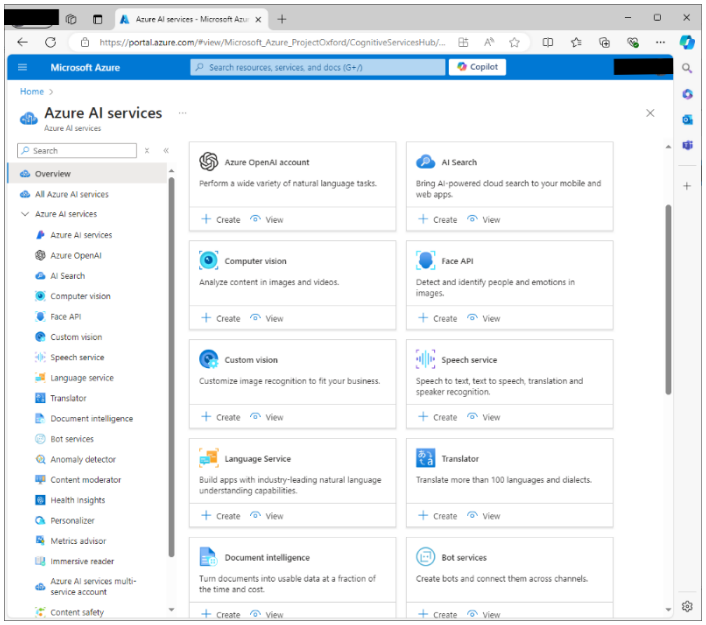



*Figure 6: Accessing the list of Azure AI services*

Locate the **AI Search** service (in Figure 6 it is at the top-right corner) and click **Create** inside its card. On the **Create a search service** page (see Figure 7), select your subscription and the resource group created previously.

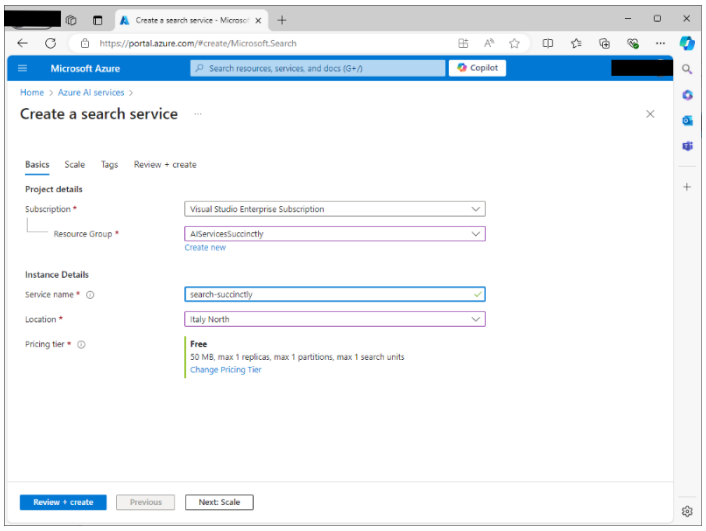



*Figure 7: Creating a search service*

In the **Service name** text box, enter a unique identifier for your service, such as **search-succinctly**. In the **Location** dropdown, select the region that is closest to you to avoid network latency. Finally, in the **Pricing tier** dropdown, select the **Free** plan (F0).

When ready, click **Review + Create**, and on the summary page, click **Create**. The service will now be created and deployed. You can adjust the service settings before creating it, but this will not be covered here. Once the deployment is complete, a confirmation page appears (see Figure 8).

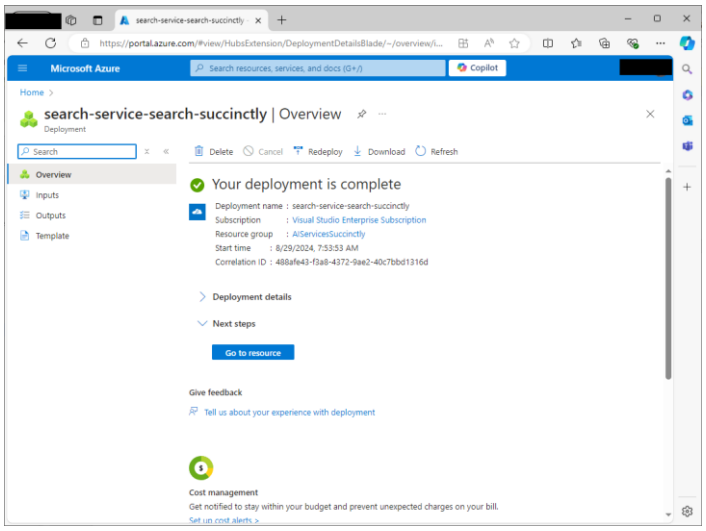



*Figure 8: Deployment completion for the AI Search service*

Click **Go to resource**. This will open the main page for the new service, as shown in Figure 9.

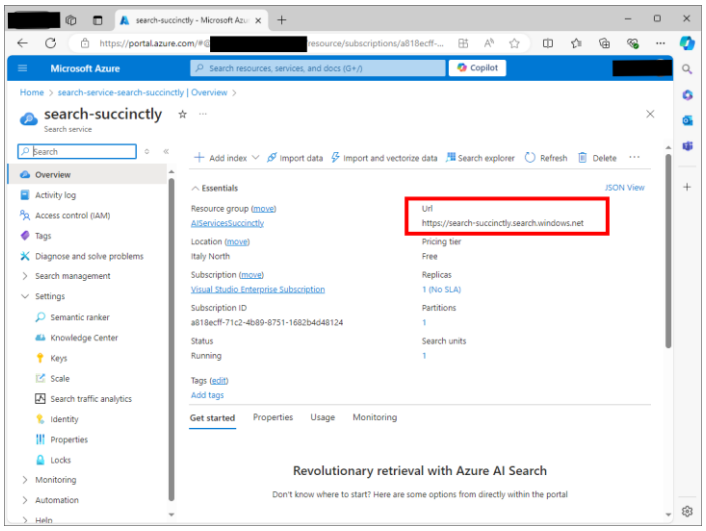



*Figure 9: The main page of the AI Search service*

In Figure 9, you can see that the URL has been highlighted in red. Take note of the service URL; it represents the access point to the service, and it will be required shortly in the source code. The other relevant information you need is the API key, which is required for authenticating against the service. Click the **Keys** menu item located at the left of the page to open the Keys page (see Figure 10).

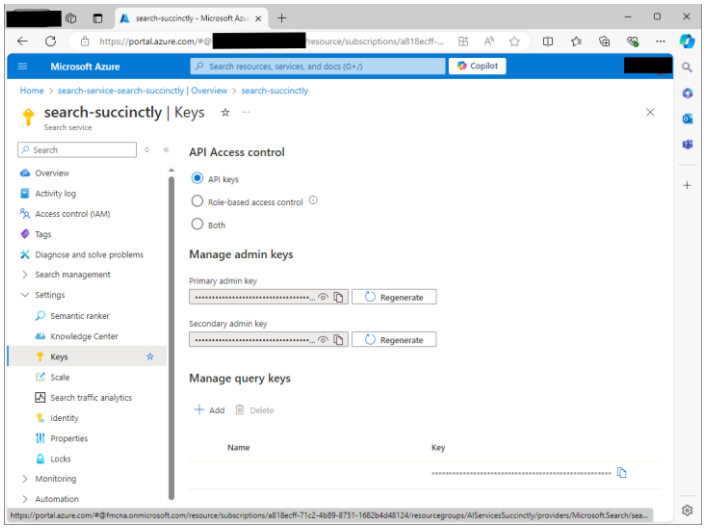



*Figure 10: Managing API keys*

By default, Azure generates two API keys, primary and secondary, but for development, you only need one. They are hidden by default for security reasons. Click the **Copy** button close to the primary API key text box and securely store the API key for later use in the source code.

***Tip: For each Azure AI service discussed in the next chapters, these are the common steps that you will need to perform.***

With Azure AI Search, you need to perform an additional step, which is configuring indexes.

## Creating an index

Creating indexes is a crucial step in configuring Azure AI Search, as indexes determine how your data will be stored, searched, and retrieved. In Azure AI Search, an index is a collection of fields, each representing a piece of information within your searchable documents, such as titles, descriptions, or metadata. Each field has specific attributes, such as whether it is searchable, filterable, sortable, or *facetable* (allows users to refine a search).

The index that you will create in the next step relates to a collection of products, and each product is mapped to the following C# class:

```
public class Product
{
    public string Id { get; set; }
    public string ProductName { get; set; }
    public string Category { get; set; }
    public double Price { get; set; }
}
```

When you create an index in Azure, you will need to map every property of your data objects to a field of the index. A practical example will clarify your doubts. In the left-hand menu of your search service's overview page, click the **Indexes** item, located under the Search management node. This is where you can view existing indexes or create new ones.

To create a new index, click **Add Index** at the top of the indexes page. Make sure you select the **Add index** option, since **Add index (JSON)** requires writing indexes in JSON format. This will open the index creation page, where you can define the structure and attributes of your index (see Figure 11).

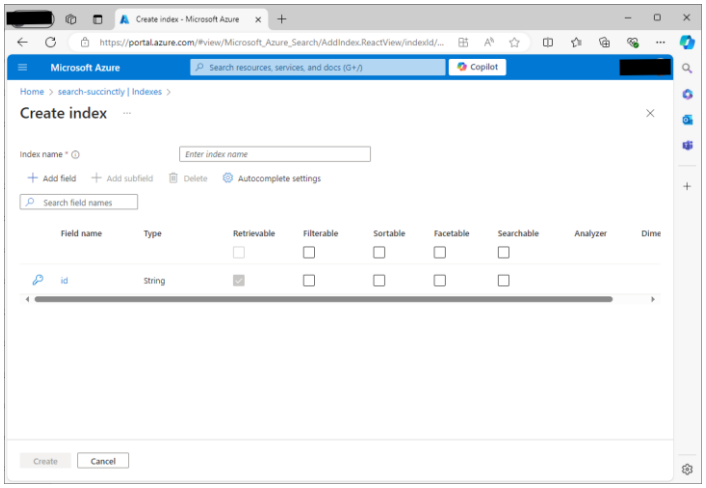



*Figure 11: The index creation page*

Here you have to define the index properties:

- Index Name: A unique name for your index. This name will be used to reference the index in your queries, API calls, and C# code.
- Fields: The fields you define will represent the schema of your data within the index. Each field corresponds to a specific property or piece of information in your documents (considering the **`Product`** class: **`ProductName`**, **`Category`**, **`Price`** for an e-commerce index).

Click **Add field** to start adding fields. At this point, you will enter into edit mode for a new field. Figure 12 shows an example based on the **`ProductName`** field.

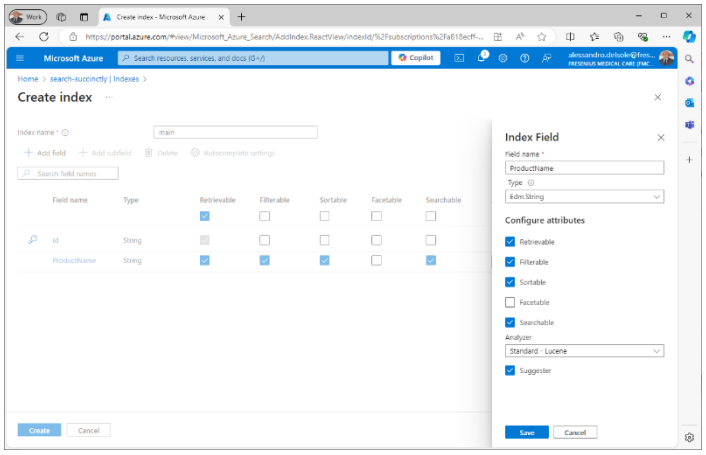



*Figure 12: Editing an index field*

The first detail you set is the Field Name, which represents the name of the field as it will appear in your index. Then, you set the data type (Type box): the appropriate data type for each field, such as **`Edm.String`** for text, **`Edm.Int32`** for integers, **`Edm.Double`** for floating-point numbers, or **`Edm.DateTimeOffset`** for dates. **`Edm`** stands for entity data model.

For collections, you can select a collection type such as **`Collection(Edm.String)`**. You can also assign one or more of the following attributes:

- **`Searchable`**: Determines whether the field's content can be searched. Typically, you would enable this for text fields like **`ProductName`** or **`Description`**.
- **`Filterable`**: Allows filtering on the field's content. For example, you might want to filter products by **`Category`** or **`Price`**.
- **`Sortable`**: Allows the results to be sorted based on this field, which is useful for fields like **`Price`**.
- **`Facetable`**: Enables faceting, which is useful for generating search facets or categories based on the values in this field, such as showing the number of products by category or price range.

Notice that one field must be designated as the key field, which serves as the unique identifier for documents in your index. By default, Azure generates a field called **`id`**. Every time you add and assign a field, click the **Save** button. Repeat the steps to add fields for the **`Category`** and **`Price`** properties, to be set with **`Edm.String`** and **`Edm.Double`**, respectively.

When you're finished, click **Save** on the index page. Indexes are very flexible and allow for implementing complex operations, which are beyond the scope of this ebook. For further information, you can read the official Azure documentation about indexes.

Now you are ready to write code.

## Creating a WPF application

To create a WPF application, open Visual Studio Code and open an instance of the Terminal by selecting **Terminal > New Terminal**. When the Terminal is ready, type the following commands:

```
> md c:\AIServices\AppSearchWpfApp
> cd c:\AIServices\AppSearchWpfApp
> dotnet new wpf
> dotnet add package Azure.Search.Documents
> dotnet add package Newtonsoft.Json
```

The following is a description of the aforementioned commands:

- The **`md`** command creates a new subfolder for a new app called **AppSearchWpfApp** inside the **AIServices** folder created in Chapter 2.
- The **`cd`** command sets the new folder as the current folder.
- The **`dotnet new wpf`** command creates a new WPF project, whose name is the same as the containing folder.
- The first **`dotnet add package`** command installs the Azure.Search.Documents NuGet package, a library from the Azure SDK that allows for interacting with the AI Search service from .NET code. The command line installs the Newtonsoft.Json library, probably the most popular for JSON manipulation, required to manage sample JSON data.

When you're ready, open the project in Visual Studio Code. Remember that VS Code is folder-based, not project-based, so you have to select **File** > **Open Folder** and open the **AppSearchWpfApp** folder created previously.

In the Explorer view, you will see the full project structure, as shown in Figure 13.

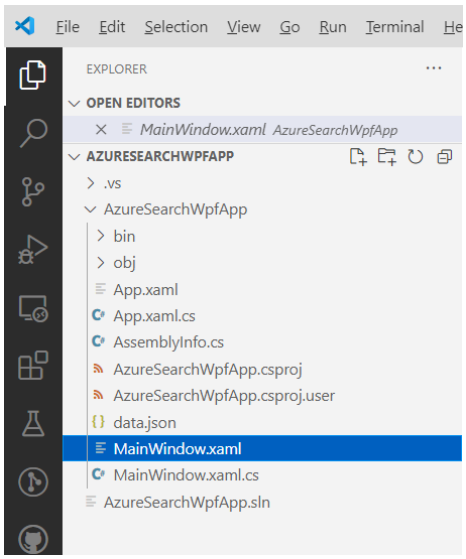



*Figure 13: The new project opened in VS Code*

Click the **New File** button on the toolbar available on the line of the project name. A new file is added, so rename it **data.json**.

In real-world scenarios, the data you process is originated from a database or from an API service, but for demonstration purposes, you will create a JSON file containing a list of products that matches the **`Product`** class shown previously. The following is the content of the JSON file that you need to add:

```
[
  {
    "Id": "1",
    "ProductName": "Laptop",
    "Category": "Electronics",
    "Price": 999.99
  },
  {
    "Id": "2",
    "ProductName": "Smartphone",
    "Category": "Electronics",
    "Price": 699.99
  },
  {
    "Id": "3",
    "ProductName": "Desk Chair",
    "Category": "Furniture",
    "Price": 149.99
  }
]
```

It is also a good idea to work with a small JSON file, because the data must be sent to the Azure AI Search service in order to be processed, and the Free pricing tier only offers 50Mb. Now that you have a project and some data, it is time to write code.

## Defining the user interface

The user interface of the sample app must include a search box, a button that launches searches, and a **`ListView`** that displays the search result in the form of a collection of products. In the MainPage.xaml file, add the code shown in Code Listing 1.

*Code Listing 1*

```
<Window x:Class="AzureSearchWpfApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:AzureSearchWpfApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800" Loaded="Window_Loaded">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal">
            <TextBox x:Name="SearchBox"
                     Width="300" Height="30"
                     VerticalAlignment="Top" Margin="10" />
            <Button x:Name="SearchButton" Content="Search"
                    Margin="10" Click="SearchButton_Click" />
        </StackPanel>
        <ListView x:Name="ResultsList"
                  Margin="10,10,10,10" Grid.Row="1">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="Product Name"
                                    DisplayMemberBinding="{Binding ProductName}"
                                          Width="200"/>
                    <GridViewColumn Header="Category"
                                    DisplayMemberBinding="{Binding Category}"
                                          Width="100"/>
                    <GridViewColumn Header="Price"
                                    DisplayMemberBinding="{Binding Price}"
                                    Width="100"/>
                </GridView>
            </ListView.View>
```

```
        </ListView>
    </Grid>
</Window>
```

Notice how each **`GridViewColumn`** object in the **`ListView`** is bound to a property of each **`Product`** instance in the retrieved collection. Now it is time to write the C# code that performs searching and generates the search result.

## Performing intelligent search in C#

The C# code needs to read the content of the data.json file, upload it to the Azure AI Search service for indexing, and query the data based on the specified search terms. This is accomplished with the code shown in Code Listing 2 (comments will follow shortly).

*Code Listing 2*

```
using Azure;
using Azure.Search.Documents;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
using System.IO;
using System.Windows;

namespace AzureSearchWpfApp
{
    public partial class MainWindow : Window
    {
        private SearchClient searchClient;
        private ObservableCollection<Product> _products;
        private ObservableCollection<Product> Products
        {
            get { return _products; }
            set { _products = value; }
        }

        public MainWindow()
        {
            InitializeComponent();
            InitializeSearchClient();
            Products = new ObservableCollection<Product>();
        }

        private void InitializeSearchClient()
        {
            string serviceEndpoint =
                "your-endpoint-here";
            string indexName = "main";
```

```
            string apiKey =
                "your-api-key";

            var credential = new AzureKeyCredential(apiKey);
            searchClient = new SearchClient(
                new Uri(serviceEndpoint), indexName, credential);
        }

        private async Task PerformSearch(string query)
        {
            try
            {
                var options = new SearchOptions
                {
                    IncludeTotalCount = true
                };
                var response = await searchClient.
                    SearchAsync<Product>(query, options);
                Products.Clear();
                var result = response.Value.GetResults();
                foreach (var item in result)
                {
                    Products.Add(item.Document);
                }
                ResultsList.ItemsSource = Products;
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error performing search: {ex.Message}");
            }
        }

        private async Task LoadDataAsync()
        {
            // Read the JSON file.
            string jsonData = File.ReadAllText("data.json");
            var products = JsonConvert.
                DeserializeObject<List<Product>>(jsonData);

            // Upload data to Azure AI Search index.
            await UploadDataToIndex(products);
        }

        private async Task UploadDataToIndex(List<Product> products)
        {
            try
            {
                // Upload the documents to the search index.
                await searchClient.UploadDocumentsAsync(products);
```

```
                MessageBox.
                    Show("Data successfully uploaded to Azure Search.");
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error uploading data to Azure Search: " +
                    $"{ex.Message}");
            }
        }

        private async void Window_Loaded(object sender,
            RoutedEventArgs e)
        {
            await LoadDataAsync();
        }

        private async void SearchButton_Click(object sender,
            RoutedEventArgs e)
        {
            string query = SearchBox.Text;
            if (!string.IsNullOrWhiteSpace(query))
            {
                await PerformSearch(query);
            }
        }
    }

}
```

The Azure SDK is central to this application, enabling interaction with the search service hosted in Azure. The following is an explanation about the .NET objects and members that are relevant for AI Search:

- The **`SearchClient`** class provides a way to interact with a specific search index in Azure AI Search. It allows performing search queries and managing documents within that index. The **`SearchClient`** object is instantiated with the service endpoint (**`Uri`**), index name (**`string`**), and **`AzureKeyCredential`** for authentication. In the code, the **`UploadDocumentsAsync<T>`** method is used to asynchronously upload a batch of documents (in this case, **`Product`** objects) to the index.

- This method returns a **`Response<IndexDocumentsResult>`** object, which contains information about the indexing operation. The **`SearchAsync<T>`** method is used to execute search queries asynchronously, returning a **`SearchResults<T>`** object containing the search results. Additionally, methods like **`MergeDocumentsAsync<T>`** and **`DeleteDocumentsAsync<T>`** allow updating and removing documents from the index, respectively, and could be useful for scenarios where you need to modify or delete indexed data.

- The **`SearchIndexClient`** class enables you to manage the structure of your search indexes. Although it is not heavily used in the code, this client is essential for creating, updating, or deleting indexes. The client is instantiated similarly to **`SearchClient`** with the service endpoint and **`AzureKeyCredential`**. The **`CreateIndexAsync`** method can be used to create a new search index programmatically, while **`DeleteIndexAsync`** allows for deleting an existing index. The **`GetIndexAsync`** method retrieves the schema of an index, which can be useful for inspecting or modifying the structure.

- The **`SearchOptions`** class configures the behavior of search queries performed with the **`SearchClient`**. In the code, **`SearchOptions`** is used to specify that the total count of results should be included in the search response (**`IncludeTotalCount = true`**). Other available properties include **`Filter`**, which can be used to apply filters to the search query, and **`Facets`**, which allows you to specify fields for faceted navigation. Additionally, **`OrderBy`** can be set to control the sort order of the search results.

- The **`SearchResults<T>`** class represents the collection of search results returned by the **`SearchAsync<T>`** method. It includes properties such as **`TotalCount`**, which gives the total number of documents that match the search query, and **`Results`**, which contains a collection of **`SearchResult<T>`** objects. The **`GetResults`** method can be used to retrieve the collection of individual search results. This class encapsulates not only the results themselves, but also metadata about the query, which can be useful for building pagination or providing more detailed insights into search performance.

- The **`SearchResult<T>`** class represents an individual document retrieved from the search index. Each **`SearchResult`** object includes properties like **`Score`**, which represents the relevance score assigned by the search engine, and **`Highlights`**, which contains the highlighted portions of the document that match the query, if highlighting is enabled. The **`Document`** property gives access to the actual document (in this case, a **`Product`** object) stored in the index. The **`SearchResult<T>`** class also exposes methods like **`GetDouble`**, **`GetString`**, and other type-specific accessors to retrieve individual fields from the result when working with dynamic schemas.

- The **`IndexDocumentsResult`** object is returned by document indexing methods like **`UploadDocumentsAsync<T>`** and **`MergeDocumentsAsync<T>`**. This object contains information about the result of the indexing operation, such as the status and number of documents successfully processed. It helps in verifying whether the documents were indexed properly, and if there were any errors.

## Errors and exceptions

If a search fails, the Azure AI services can throw the following exceptions:

- **`IndexNotFoundException`**: Thrown when trying to query a non-existent or incorrect search index.
- **`InvalidSearchQueryException`**: This exception occurs when the search query syntax is malformed or includes unsupported features.

You can implement a **`try..catch`** block for exception handling and take the appropriate actions.

## Running the application

Start the application by pressing F5. The main window will first upload the JSON data to the Index service of Azure AI Search, and then it will allow typing into the search box. The application will send the queries to Azure AI Search, retrieve the results, and display them in the **`ListView`**. Figure 14 shows an example.

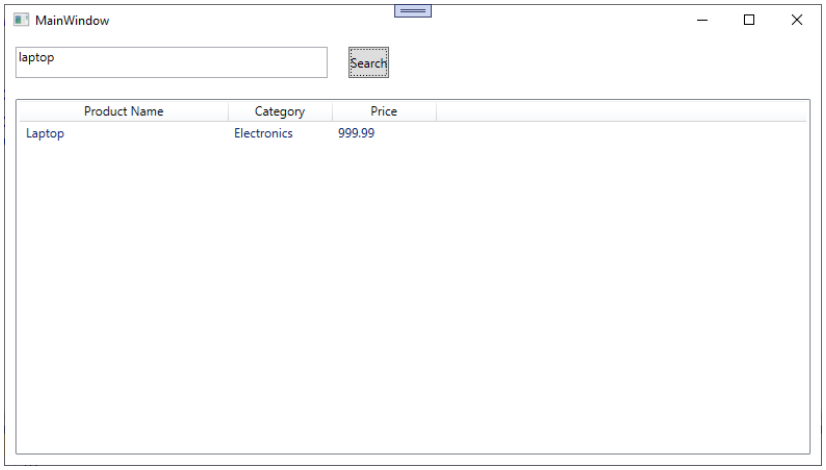



*Figure 14: The application is ready to leverage Azure AI Search*

This is obviously a simple example, but it should at least give you an idea of Azure AI Search's powerful features.

# Chapter summary

The Microsoft Azure AI Search service is a powerful, cloud-based search solution that combines traditional full-text search capabilities with advanced AI features, such as cognitive skills and machine learning models. Through this research, we explored how Azure AI Search can be set up and utilized to build sophisticated search applications.

The provided code example demonstrated how to query a search index from a WPF application, showcasing the integration of Azure AI Search into desktop applications. By leveraging the Azure AI Search SDK, developers can create rich search experiences with minimal infrastructure overhead.

# Chapter 4 Azure AI Language

The Microsoft AI Language service focuses on natural language processing (NLP) and is designed to analyze unstructured text and provide deep insights, such as language detection, sentiment analysis, key phrase extraction, named entity recognition (NER), and text summarization. This chapter describes this service in detail, providing code examples that target all the language processing features.

## Introducing Azure AI Language service

Azure AI Language specializes in the processing and understanding of human language, which makes it an essential tool for businesses seeking to extract actionable insights from large amounts of textual data. As companies accumulate vast amounts of unstructured text data, from social media interactions, emails, surveys, and documents, the challenge becomes turning this data into useful information. Azure AI Language allows businesses to automate and scale the analysis of this data by leveraging advanced NLP models built on Microsoft's extensive research in artificial intelligence. Azure AI Language offers several functionalities that cater to different aspects of language processing:

- **Sentiment Analysis**: This feature analyzes the sentiment of text data, classifying it as positive, neutral, or negative. This can be helpful in various scenarios, such as gauging customer satisfaction from feedback or analyzing public sentiment about a product.
- **Key Phrase Extraction**: This feature extracts the most important phrases from a document, highlighting the key concepts or ideas within the text. This can be particularly useful for summarizing large documents or extracting main topics from customer reviews.
- **Named Entity Recognition** (NER): NER identifies and classifies entities within text such as people, organizations, locations, and dates. This is useful for data categorization and enhancing search functionalities within applications.
- **Language Detection**: The service can automatically detect the language of the text input, which is helpful when dealing with multilingual data sources.
- **Text Summarization**: This advanced feature provides an automated summary of the content by distilling long documents into a concise summary that captures the essence of the text.
- **Text Translation**: Azure AI Language can translate text from one language to another, leveraging the same powerful models used in Microsoft Translator. This feature is actually empowered by the Azure AI Translator service, which is discussed thoroughly in Chapter 7.

The service is integrated into the Azure ecosystem, which allows seamless connection with other Azure resources like Blob Storage, Azure Cognitive Search, and Power BI, enhancing its capabilities in data-driven applications. Azure AI Language is designed to handle data privacy and security, adhering to stringent compliance standards, including GDPR and ISO/IEC certifications.

***Note: Text summarization might not be available in all Azure regions. Make sure your region supports this feature when you create the AI Language service instance in the Azure Portal.***

# Creating a sample application

In this example, you will create a WPF (Windows Presentation Foundation) application using .NET that interacts with the Azure AI Language service to perform text analysis over a text file. The application will allow users to select a file from their local machine, read its content, and analyze text using Azure AI Language. The code example focuses on all the text analysis features described previously, so it is a comprehensive example. As it normally happens for Azure AI services, you will first set up the appropriate Azure resources, and then you will create an application with Visual Studio Code.

***Tip: The companion code for this chapter ships with a text file that will be used for the coming discussions. This text file describes the story of Microsoft Corp. in English, and it has been generated with Azure OpenAI's ChatGPT. It is written in a way that all the NLP features can be demonstrated here. It also includes some sentences in German to demonstrate the services' ability to work over mixed languages. Figures for the sample application are based on this text file.***

## Setting up the Azure AI Language resources

To set up the Azure AI Language service, sign into the Azure Portal with your Microsoft account. Following the lesson learned in the previous chapter, click **AI Services** and locate the **Language** service. At this point, follow these steps:

1. Click **Create** under the **Language** card.
2. When the creation page appears, choose your subscription and the resource group created previously.
3. Provide a name for your **Language** resource (for example, **language-service-succinctly**), choose a region, and select the **Free** price tier.
4. Click **Review + create** > **Create**.

After the resource is created and deployed, click **Go to resource**. From the left-hand menu, select **Keys and Endpoint**. Copy the key and endpoint URL (either to the clipboard or to a text editor), which you will need for authentication in your application.

## Creating a WPF sample project

The goal of the sample application is analyzing the text contained inside a .txt file, which is included in the companion solution that you can use as a reference. The application will leverage the Azure AI Language service to detect sentiment, extract key phrases and named entities, detect language, and perform text summarization.

Open Visual Studio Code and then open a new Terminal window. Type the following, self-explanatory commands:

```
> md \AIServices\TextAnalysisApp
> cd \AIServices\TextAnalysisApp
> dotnet new wpf
> dotnet add package Azure.AI.TextAnalytics
```

The Azure.AI.TextAnalytics NuGet package is the library from the Azure SDK that exposes all the necessary objects you will use to interact with the Azure AI Language service. When you're ready, open the folder containing the new solution. The code will be now split into two parts: the user interface, and the code that interacts with the AI Language service.

## Designing the user interface

In WPF, the user interface is defined via the eXtensible Application Markup Language (XAML). Code Listing 3 shows the definition of the user interface for the current sample application. Comments will follow shortly.

*Code Listing 3*

```
<Window x:Class="textanalysisapp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:textanalysisapp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Button Content="Open File" HorizontalAlignment="Left"
                VerticalAlignment="Top" Width="100"
                Margin="10,10,0,0" Click="OpenFileButton_Click"/>
        <TextBox Grid.Row="1" Name="FileContentTextBox"
                 HorizontalAlignment="Left" Height="100"
                 Margin="10,50,0,0" VerticalAlignment="Top" Width="560"
```

```
                TextWrapping="Wrap"/>
        <TextBlock Grid.Row="2" Name="ResultLabel" Height="150"
                   TextWrapping="WrapWithOverflow"
                   Text="Result:" HorizontalAlignment="Left"
                   Margin="10,10,0,0" VerticalAlignment="Top"/>
        <StackPanel Grid.Row="3" Orientation="Horizontal"
                    HorizontalAlignment="Left"
                    VerticalAlignment="Top" Margin="10,10,0,0">
            <Button Content="Analyze Sentiment"
                    Width="120" Click="AnalyzeSentimentButton_Click"/>
            <Button Content="Extract Key Phrases" Width="120"
                 Margin="10,0,0,0" Click="ExtractKeyPhrasesButton_Click"/>
            <Button Content="Recognize Entities" Width="120"
                    Margin="10,0,0,0"
                    Click="RecognizeEntitiesButton_Click"/>
            <Button Content="Detect Language" Width="120" Margin="10,0,0,0"
                    Click="DetectLanguageButton_Click"/>
            <Button Content="Summarize Text" Width="120" Margin="10,0,0,0"
                    Click="SummarizeTextButton_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

The user interface definition is very simple. At the top, there is a button that allows for loading a text file. A **`TextBox`** shows the content of the text file, while a **`TextBlock`** displays the analysis results. Finally, there are several button controls, one per analysis type. At this point, it is time to implement natural language analysis in C#.

## Natural language processing in C#

Natural language processing in C# with the Azure AI Language service can be performed using the **`TextAnalyticsClient`** class and its members. Code Listing 4 shows the full implementation, and individual code blocks will be discussed next.

*Code Listing 4*

```
using Azure;
using Azure.AI.TextAnalytics;
using Microsoft.Win32;
using System.IO;
using System.Windows;

namespace TextAnalysisApp;

/// <summary>
/// Interaction logic for MainWindow.xaml.
/// </summary>
public partial class MainWindow : Window
```

```
{
    private static readonly
        string endpoint =
        "your-endpoint";
    private static readonly
        string apiKey = "your-api-key";
    private readonly TextAnalyticsClient _client;

    public MainWindow()
    {
        InitializeComponent();
        var credentials =
            new AzureKeyCredential(apiKey);
        _client = new TextAnalyticsClient(
            new Uri(endpoint), credentials);
    }

    private void OpenFileButton_Click(object sender, RoutedEventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Text files (*.txt)|*.txt";
        if (openFileDialog.ShowDialog() == true)
        {
            string fileContent = File.ReadAllText(openFileDialog.FileName);
            FileContentTextBox.Text = fileContent;
        }
    }

    private async void AnalyzeSentimentButton_Click(object sender,
        RoutedEventArgs e)
    {
        string fileContent = FileContentTextBox.Text;
        if (!string.IsNullOrEmpty(fileContent))
        {
            DocumentSentiment sentiment =
                await _client.AnalyzeSentimentAsync(fileContent);
            ResultLabel.Text = $"Sentiment: {sentiment.Sentiment}";
        }
    }

    private async void ExtractKeyPhrasesButton_Click(object sender,
        RoutedEventArgs e)
    {
        string fileContent = FileContentTextBox.Text;
        if (!string.IsNullOrEmpty(fileContent))
        {
            KeyPhraseCollection keyPhrases =
                await _client.ExtractKeyPhrasesAsync(fileContent);
            ResultLabel.Text =
```

```
                $"Key Phrases: {string.Join(", ", keyPhrases)}";
        }
    }

    private async void RecognizeEntitiesButton_Click(object sender,
        RoutedEventArgs e)
    {
        string fileContent = FileContentTextBox.Text;
        if (!string.IsNullOrEmpty(fileContent))
        {
            CategorizedEntityCollection entities =
                await _client.RecognizeEntitiesAsync(fileContent);
            string entitiesResult = "";
            foreach (var entity in entities)
            {
                entitiesResult +=
                    $"{entity.Text} ({entity.Category}), ";
            }
            ResultLabel.Text =
                $"Entities: {entitiesResult.TrimEnd(',', ' ')}";
        }
    }

    private async void DetectLanguageButton_Click(object sender,
        RoutedEventArgs e)
    {
        string fileContent = FileContentTextBox.Text;
        if (!string.IsNullOrEmpty(fileContent))
        {
            DetectedLanguage language =
                await _client.DetectLanguageAsync(fileContent);
            ResultLabel.Text =
                $"Detected Language: {language.Name}";
        }
    }

    private async void SummarizeTextButton_Click(object sender,
                        RoutedEventArgs e)
    {
        string fileContent = FileContentTextBox.Text;

        if (!string.IsNullOrEmpty(fileContent))
        {
            try
            {
                // Start the text summarization operation and wait
                // until it completes.
                var summaryOperation =
                    await _client.ExtractiveSummarizeAsync(
```

```
                        WaitUntil.Completed,
                        new List<string> { fileContent });

                // Initialize a string to store the summary.
                string summarizedText = "";

                // Process the results asynchronously.
                await foreach (var document in summaryOperation.Value)
                {
                    var firstDocument = document.FirstOrDefault();

                    if (firstDocument != null && !firstDocument.HasError)
                    {
                        foreach (var sentence in firstDocument.Sentences)
                        {
                            summarizedText += sentence.Text + " ";
                        }
                    }
                    else
                    {
                        summarizedText =
                            "An error occurred during text summarization.";
                    }
                }

                // Display the summarized text in the UI.
                ResultLabel.Text = $"Summarized Text: {summarizedText}";
            }
            catch (Exception ex)
            {
                ResultLabel.Text = $"An error occurred: {ex.Message}";
            }
        }
    }
}
```

At the top of the code implementation, you need to initialize the **`endpoint`** and **`apiKey`** variables with the service endpoint and API key that you previously stored from the Azure Portal. Following is a list of relevant objects from the Azure SDK used in the code for Azure AI Language:

- The **`TextAnalyticsClient`** client class is used for interacting with the Azure AI Language service. It provides various methods to analyze text data, such as **`ExtractiveSummarizeAsync`** to perform text summarization, **`AnalyzeSentimentAsync`** to perform sentiment analysis on text data, **`RecognizeEntitiesAsync`** to retrieve named entities, **`DetectLanguageAsync`** to detect the predominant language inside text, and **`ExtractKeyPhrasesAsync`** to extract key phrases from text.

- The **`ExtractiveSummarizeOperation`** object is returned by the **`ExtractiveSummarizeAsync`** method and represents the long-running summarization operation. The **`Value`** property contains the summarization results, which can be iterated over to access individual summarized documents. Additionally, methods like **`WaitForCompletionAsync`** can be used to await the completion of the operation, making it easier to manage asynchronous workflows.

- The **`ExtractiveSummarizeResultCollection`** holds the results of a text summarization operation. Each result in the collection corresponds to one document that was summarized. The **`Count`** property allows for iterating over the collection, and each document result can be accessed through its index.

- The **`ExtractiveSummarizeResult`** class represents the summarized result of a single document. It includes the **`Sentences`** property, which is a collection of **`SummarySentence`** objects representing the sentences that make up the summary. The **`HasError`** property indicates whether there was an error processing the document, and the **`Id`** property identifies the document in the collection.

- The **`SummarySentence`** class represents a single sentence in a summarized document. The **`Text`** property contains the actual text of the sentence, and the **`RankScore`** property indicates the sentence's relevance within the document, with higher values indicating greater importance. Other properties, like **`Offset`** and **`Length`**, provide information about the position of the sentence within the original document.

- The **`DocumentSentiment`** object is returned by the **`AnalyzeSentimentAsync`** method and represents the result of sentiment analysis. It contains the **`Sentiment`** property, which indicates the overall sentiment of the text, and **`ConfidenceScores`**, which provides confidence levels for each sentiment category (positive, neutral, and negative).

- The **`CategorizedEntityCollection`** collection is returned by the **`RecognizeEntitiesAsync`** method. It contains categorized entities recognized in the text, such as people, organizations, or locations. Each entity in the collection is described by its category, subcategory, and confidence score, allowing for detailed entity recognition and analysis.

- The **`DetectedLanguage`** object is returned by the **`DetectLanguageAsync`** method and represents the detected language in a document. It includes the **`Name`** property, which indicates the name of the detected language, and the **`ConfidenceScore`** property, which shows the confidence level of the detection. This is useful for determining the primary language of a document or set of documents in multilingual applications.

By combining these objects and methods, developers can extend their applications to include various text analytics capabilities such as summarization, sentiment analysis, entity recognition, key phrase extraction, and language detection using the Azure AI Language service. This enables more comprehensive text processing in real-world applications.

## Errors and exceptions

If language analysis fails, the Azure AI Language service can throw the following exceptions:

- **`InvalidDocumentException`**: Thrown when the document provided for analysis (e.g., for sentiment or text analytics) doesn't meet format or content requirements.
- **`DocumentLimitExceededException`**: Thrown when the batch size exceeds the limit allowed by the service for a single request.

Do not forget to implement a **`try..catch`** block as a best practice for exception handling.

# Running the application

Press F5 to run the application. Click **Open File** and select a .txt file that contains some text suitable for a full language analysis. The companion source code contains a file called text.txt that you can use for simplicity; it's the base for the next figures. The first analysis type is sentiment analysis. Click the **Analyze Sentiment** button, and you will get the result shown in Figure 15.

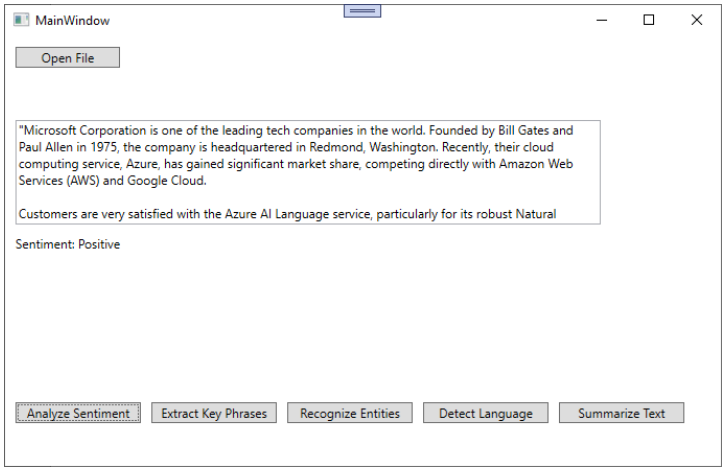



*Figure 15: Analyzing the sentiment from the text*

As you can see, the language analysis returned a positive sentiment. Now click **Extract Key Phrases**. You will get the result shown in Figure 16.

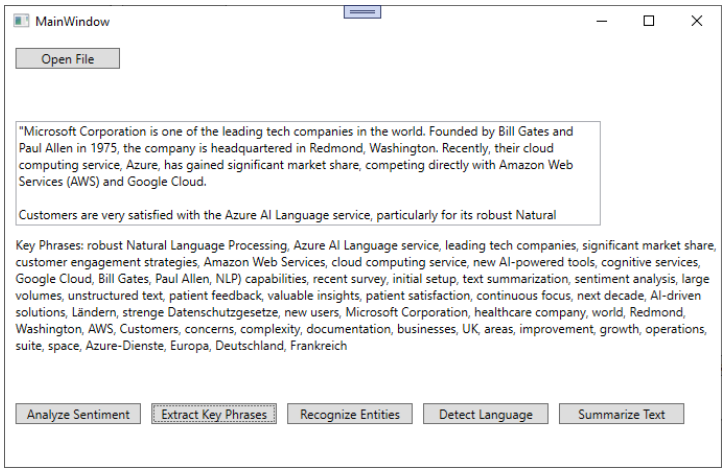



*Figure 16: Extracting key phrases*

As you can see, the list is quite long, but you can quickly understand the work done by the language analysis engine in collecting the relevant key phrases. Now, click **Recognize Entities** to get the result shown in Figure 17.

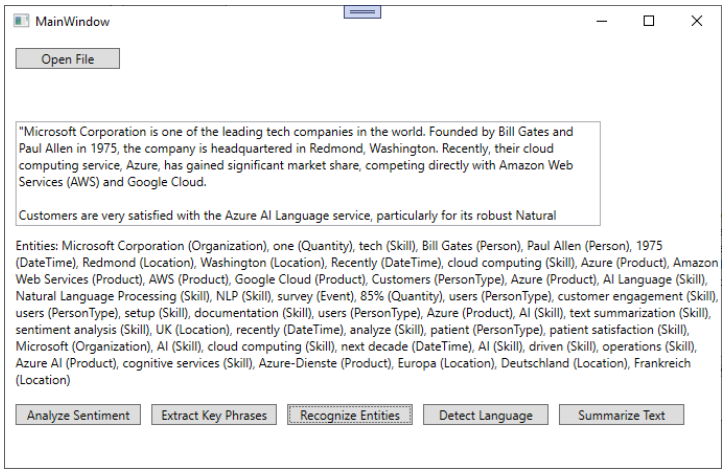



*Figure 17: Extracting named entities*

As you can see, the analysis engine returns a long list of named entities detected in the text, also specifying the type of entity (for example, person, quantity, DateTime). This is extremely valuable because it allows you to filter entities by type.

Now, click **Detect Language**. If you look at Figure 18, you can see that the language detected is English, even if there are some sentences in German. The reason is that the analysis engine returns the prominent language.

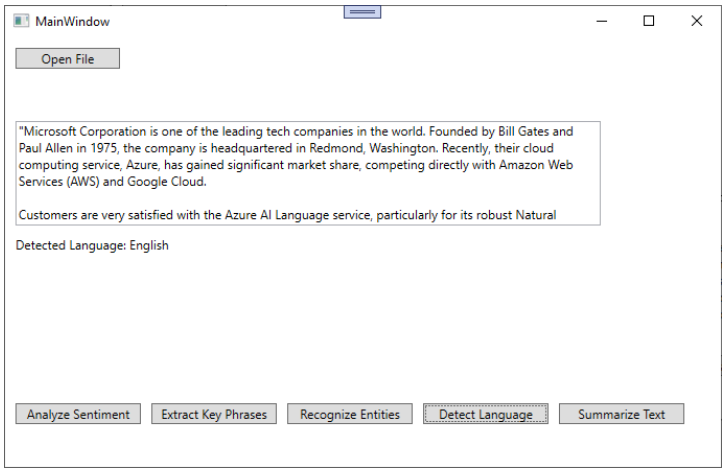



*Figure 18: Detecting languages*

Finally, click **Summarize Text**. As you can see in Figure 19, the AI Language service returns a summary of the original text.

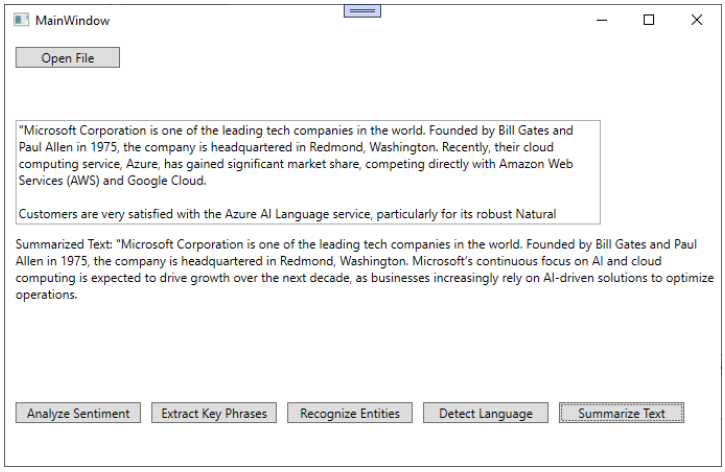



*Figure 19: Extracting text summaries*

With extremely limited effort, you have been able to perform advanced text analysis taking advantage of natural language processing.

## Chapter summary

The Microsoft Azure AI Language service offers robust capabilities for processing and analyzing unstructured text data, enabling businesses to extract meaningful insights. This service integrates smoothly with applications using Azure SDKs, and by following the steps in this chapter, developers can create powerful applications that leverage these AI capabilities.

The sample application demonstrated how to use the Azure AI Language service in a WPF application, focusing on all the text analysis features for text files. By implementing this service, businesses can automate text analysis workflows and gain insights into customer sentiment, document summarization, and more, while benefiting from the scalability and security of Azure's cloud infrastructure.

# Chapter 5 Azure AI Document Intelligence

Microsoft Azure AI Document Intelligence is a specialized AI service that focuses on automating the extraction, analysis, and processing of information from various document types. This chapter describes the different features provided by this service, with code examples that will help you work with forms and documents leveraging the power of AI.

## Introducing Azure AI Document Intelligence

Azure AI Document Intelligence is designed to intelligently extract text, key-value pairs, tables, and other structured information from documents like invoices, receipts, contracts, or any form requiring data entry. Azure AI Document Intelligence leverages machine learning models to identify and structure data, minimizing the need for manual intervention. It offers capabilities to customize and train models specific to your business documents, allowing for improved accuracy and adaptability. The service is especially useful in scenarios involving high volumes of documents, where manual data entry can be time-consuming, expensive, and error-prone.

Azure AI Document Intelligence provides prebuilt and custom models to extract information from structured, semistructured, and unstructured documents. These models are trained to understand the format of specific documents, like invoices, identity cards, or business forms. The service supports multiple languages and formats, including PDFs, scanned images, and photographs. More specifically, it offers:

- **Prebuilt models**: Prebuilt models are designed to recognize and extract information from common document types, such as receipts, invoices, and business cards. These models are trained by Microsoft and can be used immediately without the need for further training.
- **Custom models**: Custom models allow users to train AI models tailored to their specific document layouts. Using labeled data, users can train models to recognize key-value pairs and other structured information in documents that do not conform to standard templates.
- **General document model**: This subservice extracts content and layout information from any document without the need for training or labeling. It can handle a wide variety of documents, extracting text, tables, and other data.
- **Layout model**: This is a base model used to extract document layout information, including text, tables, selection marks, and the overall structure. It is often used as a preprocessing step before further processing with other models.

Azure AI Document Intelligence exposes the aforementioned models through a number of services, described in the next section.

## Services of Azure AI Document Intelligence

In terms of AI services that you can use in your applications, Azure AI Document Intelligence provides the following:

- **Form Recognizer**: Form Recognizer is the core component of Azure AI Document Intelligence. It allows the extraction of information from forms, invoices, and other documents through prebuilt or custom models. It can recognize fields, checkboxes, signatures, and tables from scanned or digital documents. The form processing models within Form Recognizer support various forms such as receipts, invoices, and business cards.
- **Invoice Recognizer**: This is a specialized prebuilt model that extracts information from invoices, including fields like invoice number, date, total amount, and due date. It helps automate the processing of accounts payable by accurately extracting and categorizing invoice data.
- **Receipt Recognizer**: The receipt recognizer model extracts information from sales receipts, such as the transaction date, total amount, items purchased, and merchant details. This model is useful in automating expense reporting and tracking.
- **Business Card Recognizer**: This prebuilt model extracts contact details from business cards, including name, phone number, email address, and company name. It is particularly useful for customer relationship management (CRM) systems to automate the entry of new contacts.
- **ID Document Recognizer**: This model is used to extract key fields from identification documents, such as driver's licenses, passports, and ID cards. The fields it extracts typically include name, date of birth, and document number.

Each of these subservices is built to handle specific document types and extract information in a way that can be easily integrated into other business processes and workflows. In the next sections, you will get some code examples about document analysis via Azure AI Document Intelligence services.

# Configuring the Azure resources

Before writing code, you need to set up the Azure AI Document Intelligence service on the Azure Portal. This is going to be quite fast, since you will perform the same steps you did with the previous services.

Once logged in, click **AI Services**. Locate the **Document Intelligence** service and click **Create**. In the service creation page, select the resource group created previously and choose the closest region to your location. Specify **document-intelligence-succinctly** as the service name and select the **Free** pricing tier. Finally, click **Review + Create** > **Create**. As usual, retrieve and store the service endpoint and API key for later use.

## Sample application: processing invoices

The goal of the example is showing how to create a WPF application in Visual Studio Code that uses the Azure AI Document Intelligence service to process invoices. The application will allow the user to select a local PDF file of an invoice and then extract key information, such as the invoice number, total amount, and due date. The companion code contains a prebuilt PDF document that you can use, with the following structure:

```
-----------------------------------------------
Invoice #:  INV-1001
Invoice Date:  August 25, 2024
Due Date:  September 25, 2024
Bill To:
Alessandro Del Sole
123 Main St
Seattle, WA 98000
Item Description                    Amount
-----------------------------------------------
Website Development Services        $1,500.00
Monthly Hosting (August 2024)         $100.00
-----------------------------------------------
Total Amount Due:                   $1,600.00
-----------------------------------------------
```

This is more than enough to leverage the power of the AI Document Intelligence service.

> ***Tip: You can create your own invoice sample in Microsoft Word, using a similar structure, and then save the document as PDF.***

Following the lessons learned in the previous chapters, create a new WPF project in Visual Studio Code called **InvoiceProcessorApp**. Make sure you install the Azure.AI.FormRecognizer NuGet package, which is the library that allows for interacting with the AI Document Intelligence service.

In summary, these are the command lines you need to run:

```
> md \AIServices\InvoiceProcessorApp
> cd \AIServices\InvoiceProcessorApp
> dotnet new wpf
> dotnet add package Azure.AI.FormRecognizer
```

When finished, open the new project in Visual Studio Code.

## Defining the user interface

In the MainWindow.xaml file, add the code shown in Code Listing 5 to implement a simple UI with a button that loads a PDF document, and a **`TextBlock`** that shows the results of the document processing.

*Code Listing 11*

```
<Window x:Class="InvoiceProcessorApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:InvoiceProcessorApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Button Name="BtnSelectFile" Content="Select Invoice"
                HorizontalAlignment="Left" VerticalAlignment="Top"
                Margin="10,10,10,0" Width="100" Height="30"
                Click="BtnSelectFile_Click"/>
        <TextBlock Grid.Row="1" Name="TxtInvoiceData"
                   HorizontalAlignment="Left" VerticalAlignment="Top"
                   Margin="10,10,10,0" Width="500" TextWrapping="Wrap" />
    </Grid>
</Window>
```

The next step is about adding the document processing logic to the C# code.

## Document analysis in C#

When the application runs, the user can select a PDF invoice from the local file system using a dialog. The application will then send the selected file to Azure AI Document Intelligence, specifically using the prebuilt invoice model. Once the invoice is processed, the extracted information, such as the invoice number, total amount due, and due date, will be displayed in the text block within the WPF window.

To accomplish this, add the code shown in Code Listing 6 to the MainPage.xaml.cs file, with comments following shortly.

*Code Listing 6*

```
using Azure;
using Azure.AI.FormRecognizer.DocumentAnalysis;
using Microsoft.Win32;
using System.IO;
using System.Windows;

namespace InvoiceProcessorApp
```

```
{
    public partial class MainWindow : Window
    {
        private readonly string
            endpoint = "your-endpoint";
        private readonly string
            apiKey = "your-api-key";

        public MainWindow()
        {
            InitializeComponent();
        }

        private async void BtnSelectFile_Click(object sender,
                RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "PDF files (*.pdf)|*.pdf";
            if (openFileDialog.ShowDialog() == true)
            {
                string filePath = openFileDialog.FileName;
                await ExtractInvoiceData(filePath);
            }
        }

        private async Task ExtractInvoiceData(string filePath)
        {
            var credential = new AzureKeyCredential(apiKey);
            var client = new DocumentAnalysisClient(new Uri(endpoint),
                            credential);

            using var stream = new FileStream(filePath, FileMode.Open);
            AnalyzeDocumentOperation operation =
                await client.AnalyzeDocumentAsync(
                    WaitUntil.Completed, "prebuilt-invoice", stream);

            AnalyzeResult result = operation.Value;

            string invoiceNumber =
                result.Documents[0].Fields["InvoiceId"].Content;
            string totalAmount =
                result.Documents[0].Fields["AmountDue"].Content;
            string dueDate =
                result.Documents[0].Fields["DueDate"].Content;

            TxtInvoiceData.Text =
                $"Invoice Number: {invoiceNumber}\nTotal Amount: " +
                $"{totalAmount}\nDue Date: {dueDate}";
        }
```

```
    }
}
```

The following is an explanation of .NET objects used in the code to interact with Azure AI Document Intelligence:

- The **`DocumentAnalysisClient`** class is the core client used to interact with Azure AI Document Intelligence. This class facilitates document analysis by enabling the submission of documents to prebuilt models or custom models. In the current example, the **`AnalyzeDocumentAsync`** method is invoked to analyze an invoice. Additionally, this client provides other methods, like **`StartAnalysisAsync`**, for handling larger documents in a more granular way. The constructor requires the service endpoint and an instance of **`AzureKeyCredential`** to authenticate API requests.
- The **`AnalyzeDocumentOperation`** class represents an asynchronous operation that processes documents using a specific model. In the sample code, it has been used to handle the document analysis request, waiting for the operation to complete with await. The **`AnalyzeDocumentOperation`** can be polled periodically for long-running document processing tasks, which is particularly useful when working with documents that contain multiple pages or complex layouts. While the code invoked **`AnalyzeDocumentAsync`** for a straightforward analysis, other methods in this class, like **`GetDocumentResult`**, allow for retrieving the analysis results directly, which can be useful when integrating with workflows that need finer control over document processing.
- The **`AnalyzeResult`** object holds the result of a document analysis. It contains extracted content, such as key-value pairs, tables, and text. The **`Documents`** property is of type **`IReadOnlyList<AnalyzedDocument>`**, which represents the analyzed documents with fields and their corresponding values. Besides basic extraction, the **`AnalyzeResult`** class supports complex scenarios. For instance, if your document contains tables, you can access them via the **`Tables`** property, which provides structured table data. The **`Pages`** property also provides detailed information about the layout of each page, such as lines of text, selection marks, and bounding boxes, making it easier to implement custom rendering or export logic based on the extracted data.
- Each field extracted from the document is represented by the **`DocumentField`** class. This class provides access to the field's content, confidence score, and bounding box (if available). In our example, we used the **`Content`** property to extract the recognized values, such as invoice numbers and total amounts. Depending on the type of field, the **`DocumentField`** class has several properties that provide strongly typed access to data, such as **`ValueType`**, **`ValueString`**, **`ValueDate`**, and **`ValueCurrency`**. These specialized properties allow for the type-safe extraction of data, reducing the need for manual parsing and enhancing the reliability of the extracted information.

Notice how the **`Fields`** property requires specifying a conventional identifier that the AI service can use to detect the various document parts quickly, such as **`InvoiceId`**, **`AmountDue`**, and **`DueDate`**. This is based on the prebuilt invoice model (**`prebuilt-invoice`**), and the full list of conventional identifiers is available in the official documentation, where you will also find the identifier of the other prebuilt models.

## Errors and exceptions

If language analysis fails, the Azure AI Language service can throw the following exceptions:

- **`InvalidDocumentFormatException`**: Thrown when the document submitted for analysis, such as OCR or form recognition, is in an unsupported format or is unreadable.
- **`UnsupportedLanguageException`**: Raised when the document contains text in a language not supported by the Document Intelligence service.

Do not forget to implement a **`try..catch`** block as a best practice for exception handling.

## Running the application

Press F5 to run the application. When the main window appears, click **Select Invoice**. As you can see in Figure 20, the AI Document Intelligence service has retrieved the information specified in the C# code from the loaded PDF document.


*Figure 20: Detecting invoice elements with AI Document Intelligence*

As you can easily imagine, with limited effort you can automate the analysis and data extraction from complex documents. Especially with PDFs, this is extremely valuable.

# Hints about training and analyzing custom models

To create a sample app using Azure AI Document Intelligence against custom models, you must follow a structured approach that includes defining a custom model, training it with labeled data, and integrating it into your application. Custom models are especially useful when dealing with specific document layouts that prebuilt models may not fully support, allowing for more tailored and accurate data extraction.

In the Azure Portal, you navigate to your Azure AI Document Intelligence resource, and then you will upload a set of documents (PDFs, images) for manually labelling key fields that you want the model to recognize. The labeling process involves marking areas of interest (e.g., invoice numbers, dates, amounts) in the document. Once labeled, train your custom model by specifying the set of labeled documents as the training data. The service uses these annotations to learn the structure and recognize similar patterns in future documents. You can find detailed guidance on training custom models in the official documentation.

After training, you'll be provided with a model ID, which uniquely identifies your custom model. To use this model in a .NET app, follow a similar process to the example outlined previously. The difference lies in invoking the custom model by using its model ID instead of prebuilt model IDs. In your code, replace the call to the prebuilt model (e.g., **`prebuilt-invoice`**) with your custom model ID when using the **`AnalyzeDocumentAsync`** method of the **`DocumentAnalysisClient`** class. This allows the application to process new documents using the tailored extraction logic of your custom model. For more detailed steps and best practices for custom models, you can refer to the official Azure AI Document Intelligence documentation for composing custom models. This resource covers everything from creating and training custom models to integrating them into applications.

## Chapter summary

Azure AI Document Intelligence simplifies the extraction and processing of data from various types of documents, from invoices to business cards, by leveraging powerful AI models. The service supports both prebuilt models for common document types and custom models for specific layouts, making it a versatile solution for automating document workflows. By integrating this service into a WPF application using Visual Studio Code, you have seen how to create a practical, real-world example that processes invoices, extracts critical data, and presents it in an intuitive user interface. This approach can significantly reduce manual data entry and enhance productivity across various industries.

# Chapter 6 Azure AI Content Safety

Microsoft Azure AI Content Safety is a cloud-based service that leverages machine learning and AI techniques to help organizations monitor, review, and moderate user-generated content. This chapter describes the AI Content Safety service and provides examples to analyze text and images, with hints about video analysis.

## Introducing Azure AI Content Safety

Microsoft Azure AI Content Safety is a powerful, AI-driven service designed to help businesses and organizations maintain a safe and legally compliant digital environment by automatically identifying harmful, inappropriate, or offensive content. As user-generated content continues to proliferate on platforms such as social media, forums, and gaming communities, the need to monitor and moderate this content effectively has become extremely important. Azure AI Content Safety provides an automated, scalable solution that utilizes advanced AI models to flag content that violates community guidelines or legal regulations. Replacing the deprecated Azure AI Content Moderator, Azure AI Content Safety enhances the accuracy of detection by leveraging cutting-edge AI models that have been trained on diverse datasets to recognize various forms of harmful content, including hate speech, violent imagery, and more.

### Summary of Azure AI Content Safety capabilities

The service allows developers to integrate moderation capabilities directly into their applications and websites via a set of APIs, ensuring that harmful content can be intercepted before it reaches a broader audience. It provides real-time moderation capabilities, making it ideal for live platforms that require immediate action, such as social media networks, streaming services, and online forums. The service uses a range of machine learning techniques to identify harmful content, allowing users to set up rules and thresholds according to their needs.

Azure AI Content Safety can detect various types of undesirable content, including hate speech, sexually explicit material, violence, harassment, and more. This detection can be fine-tuned with custom settings, giving businesses the flexibility to align the moderation engine with their specific guidelines.

One of the most interesting features of Azure AI Content Safety is its ability to handle multilingual content, making it a viable solution for global platforms. Additionally, it integrates seamlessly with human review workflows, ensuring that flagged content can be reviewed manually when needed. This combination of AI and human moderation helps improve the overall effectiveness and fairness of the moderation process.

## Services of Azure AI Content Safety

Azure AI Content Safety is broken down into several services, each tailored to a specific type of content: text safety, image safety, and video safety. These subservices allow developers to implement moderation based on the type of content their platforms handle:

- **Text safety** focuses on analyzing textual content to detect inappropriate or harmful language. This includes identifying hate speech, threats, sexually explicit language, and offensive comments. The service can analyze text in multiple languages, making it a versatile tool for global content moderation. Additionally, it can detect personally identifiable information (PII) in text, which helps businesses comply with data protection regulations.
- **Image safety** analyzes images to detect offensive or inappropriate visual content. The AI models are trained to recognize explicit content, such as nudity or graphic violence, and flag these images accordingly. The service can also detect potentially risky content in various categories, allowing businesses to manage visual content more effectively. For custom use cases, it is possible to configure custom image lists for the service to reference during moderation.
- **Video safety** is built upon image safety by allowing for the moderation of video content. The service analyzes video frames to detect harmful content, such as graphic violence or explicit scenes, and flags them for review. Video safety can moderate both recorded and live video streams, making it suitable for a variety of use cases, including video-sharing platforms, live-streaming services, and online conferencing tools. The next section provides code examples to help you understand how these services work.

# Configuring the Azure resources

Before diving into the code, you need to set up the Azure AI Content Safety service on the Azure Portal. Once logged in, and following the lesson learned in Chapter 3, click **AI Services** and then locate the **Content Safety** service (usually at the bottom of the dashboard page).

Click **Create** under the service card and fill out the necessary details, such as subscription, resource group, and region, as you did for the previous services. Assign **content-safety-succinctly** as the service name for consistency with the current example. If the name is not available, provide one of your choosing. Select the **Free** pricing tier and click **Review + Create**. Finally, click **Create** to deploy the service.

Once the service is deployed, click **Go to resource** and retrieve your API key and endpoint URL, which will be required for the code examples.

***Note: In the Azure Portal, you will also see the Content Moderator service, which is the previous version of Azure AI Content Safety. Because Content Moderator has been deprecated, it is available in read-only mode, and you can only view existing resources.***

## Sample application: text safety

The purpose of the first example is to create a .NET console application that integrates with the Azure AI Content Safety API to analyze and moderate text input. The goal is to detect harmful or offensive language in user-generated content, such as social media posts or forum comments. To accomplish this, open Visual Studio Code and an instance of the Terminal, where you type the following commands:

```
> md c:\AIServices\TextSafetyExample
> cd c:\AIServices\TextSafetyExample
> dotnet new console
> dotnet add package Azure.AI.ContentSafety
```

The Azure.AI.ContentSafety NuGet package is a library from the Azure SDK that allows for interaction with the Content Safety service from .NET code. It will be required in all the examples in this chapter. When the code editor is ready, add the content of Code Listing 7.

*Code Listing 7*

```
using System;
using System.Threading.Tasks;
using Azure;
using Azure.AI.ContentSafety;

namespace TextSafetyExample
{
    class Program
    {
        private static readonly
                string contentSafetyEndpoint = "your-endpoint";
        private static readonly
                string contentSafetyApiKey = "your-api-key";

        static async Task Main(string[] args)
        {
            var client = new ContentSafetyClient(
                new Uri(contentSafetyEndpoint),
                new AzureKeyCredential(contentSafetyApiKey));

            string textToModerate =
            "I think AI Services Succinctly is a good learning resource.";
            Console.WriteLine("Text to moderate:");
            Console.WriteLine(textToModerate);

            var moderationRequest = new AnalyzeTextOptions(textToModerate);
            Response<AnalyzeTextResult> result =
```

```
            await client.AnalyzeTextAsync(moderationRequest);

            Console.WriteLine("\nModeration Results:");
            if (result.Value.BlocklistsMatch != null)
            {
                Console.WriteLine("Blocked text match found:");
                foreach (var block in result.Value.BlocklistsMatch)
                {
                    Console.WriteLine($"Name: {block.BlocklistName}, term:
                    {block.BlocklistItemText}");
                }
            }

            if (result.Value.CategoriesAnalysis != null)
            {
                Console.WriteLine("\nCategory Analysis:");
                foreach (var category in result.Value.CategoriesAnalysis)
                {
                    Console.WriteLine($"Category: {category.Category},
Confidence: {category.Severity}");
                }
            }
        }
    }
}
```

The code uses the following objects and members from the Azure AI Content Safety SDK:

- **`ContentSafetyClient`**: This is the core class that facilitates interaction with the Azure AI Content Safety service. It is used to send text content for moderation. The client requires a **`Uri`** for the service endpoint and an **`AzureKeyCredential`** for authentication. For text content moderation, the **`AnalyzeTextAsync`** method is employed, which takes in an **`AnalyzeTextOptions`** object and returns a **`Response<AnalyzeTextResult>`**. This client is not limited to text moderation, as it also supports analyzing images and videos via methods like **`AnalyzeImageAsync`** and **`AnalyzeVideoAsync`**, making it versatile for different content moderation scenarios.

- **`AnalyzeTextOptions`**: This class encapsulates the configuration for a text moderation request. It requires the text to be moderated as a string. The constructor also allows you to include optional parameters, such as blocklist IDs, if you want to moderate content against custom-defined blocklists. This flexibility makes it possible to tailor text moderation to the specific needs of your application, such as filtering out sensitive language or enforcing community guidelines.

- **`AnalyzeTextResult`**: This class represents the result of a text moderation operation, and it exposes properties described in the next points.

- **`BlockListMatch`**: This class is part of the **`AnalyzeTextResult`** and encapsulates information about blocked terms detected in the analyzed text. Each **`BlockListMatch`** object contains properties like **`Term`**, which indicates the exact term that matched, and **`ListId`**, which refers to the blocklist that triggered the match. This is particularly useful when multiple blocklists are in use, allowing you to differentiate between different types of violations or content concerns. You can also retrieve additional blocklist details using methods like **`GetBlockListDetailsAsync`** if more context is needed.

- **`CategoriesAnalysis`**: This property is a collection of **`TextCategoriesAnalysis`** objects. Each object in the collection exposes a **`Category`** property of type **`TextCategory`**, an enumeration described in Table 2, and labels harmful content categories detected in the text. Each category may appear with a confidence score, giving insight into the AI's level of certainty. The categories span a wide range of harmful content types, such as hate speech, adult content, and violence. For even more refined moderation, developers can call methods like **`GetCategoryDetailsAsync`**, which offers a deeper breakdown of the content categories.

- **`Response<T>`**: This is a generic class that wraps the result of the **`AnalyzeTextAsync`** method. It provides access to the actual moderation result (**`AnalyzeTextResult`**) through the **`Value`** property. Additionally, it contains HTTP response information like status codes and headers, which can be helpful for debugging or logging purposes.

*Table 2: Values from the TextCategory enumeration*

| | |
|---|---|
| **`Sexual`** | Indicates the content contains adult or sexually explicit material. |
| **`Hate`** | Indicates the content contains hate speech or offensive language targeting individuals or groups. |
| **`Violence`** | Indicates the content contains violent or graphic imagery. |
| **`SelfHarm`** | Indicates the content promotes self-harm or suicidal ideation. |

For the current example, there is intentionally no offensive or harmful content, since this is a professional publication. However, you can try writing different text to see the results.

## Running the application

Press F5 to start debugging the application. When you run a Console app for the first time, Visual Studio Code will ask you to specify a debugger, as shown in Figure 21. Make sure you click the **C#** option.


*Figure 21: Selecting the C# debugger for Console apps*

At this point, Visual Studio Code will ask you to specify the project that you want to debug from a similar dropdown. You have only one project, so click its name. Now the application will be built and started with an attached instance of the debugger. The application output is sent to the DEBUG CONSOLE panel. Figure 22 shows the result of the text analysis for content safety.


*Figure 22: The result of the text analysis for content safety*

There is nothing offensive in the source text, and this is why nothing is found. Try on your machine with different sentences to see how the result changes.

# Sample application: image safety

The goal of the second example is creating a WPF app that allows users to select an image from their local machine and send it to the Azure AI Content Safety service for moderation. The goal is to analyze the selected image for harmful content, such as explicit images or graphic violence, and display the results in the user interface. You do not need to set up a new resource in Azure, you can reuse the existing service with its endpoint and key. What you instead need to do is create a new WPF project with the following commands to be run in Visual Studio Code's Terminal:

```
> md c:\AIServices\ImageSafetyApp
> cd c:\AIServices\ImageSafetyApp
```

```
> dotnet new wpf
> dotnet add package Azure.AI.ContentSafety
```

As you can see, you will also use the same NuGet package from the Azure SDK, which implies reusing some of the objects described in the first example.

## Defining the user interface

The user interface for the sample application is very simple. It contains a button that allows for loading an image, an **`Image`** control that displays the selected image, and a **`TextBlock`** that will show the analysis result. Code Listing 8 contains the code that you need to add to the MainPage.xaml file.

*Code Listing 8*

```
<Window x:Class="ImageSafetyApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:ImageSafetyApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto"/>
            <RowDefinition />
        </Grid.RowDefinitions>
        <Button x:Name="SelectImageButton" Content="Select Image"
         Width="200" Click="SelectImageButton_Click" Margin="10" />
        <Image x:Name="SourceImage" Grid.Row="1"
         Margin="10" Width="320" Height="240"/>
        <TextBlock x:Name="ModerationResult"
         Grid.Row="2" Margin="10" />
    </Grid>
</Window>
```

Now that you have a convenient user interface, you are ready to analyze an image for content safety in C#.

## Image safety in C#

The imperative code for this example needs to load an image, analyze it for content safety, and display the results. This is accomplished with the code shown in Code Listing 9 (comments will follow shortly). For the endpoint and API key, reuse the same ones from the previous example.

*Code Listing 9*

```
using Azure;
using Azure.AI.ContentSafety;
using Microsoft.Win32;
using System;
using System.IO;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Media.Imaging;

namespace ImageSafetyApp
{
    public partial class MainWindow : Window
    {
        private static readonly string
            contentSafetyEndpoint = "your-endpoint";
        private static readonly string
            contentSafetyApiKey = "your-api-key";

        public MainWindow()
        {
            InitializeComponent();
        }

        private async void SelectImageButton_Click(object sender,
           RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter =
            "Image files (*.jpg, *.png)|*.jpg;*.png";

            if (openFileDialog.ShowDialog() == true)
            {
                string imagePath = openFileDialog.FileName;
                SourceImage.Source =
                   new BitmapImage(new Uri(imagePath));
                await ModerateImage(imagePath);
            }
```

```
        }

        private async Task ModerateImage(string imagePath)
        {
            var client = new ContentSafetyClient(
                new Uri(contentSafetyEndpoint),
                new AzureKeyCredential(contentSafetyApiKey));
            string moderationResult = string.Empty;

            using (var imageStream = File.OpenRead(imagePath))
            {
                BinaryData imageData =
                    BinaryData.FromStream(imageStream);
                var contentSafetyImageData =
                    new ContentSafetyImageData(imageData);
                var moderationRequest =
                    new AnalyzeImageOptions(contentSafetyImageData);
                Response<AnalyzeImageResult> result =
                    await client.AnalyzeImageAsync(moderationRequest);
                foreach(var item in result.Value.CategoriesAnalysis)
                {
                    moderationResult = string.Concat(
                        moderationResult, $"\nCategory: {item.Category},
                                          Severity: {item.Severity}");
                }
                ModerationResult.Text = moderationResult;
            }
        }
    }
}
```

For the image analysis example, the following is the list of relevant objects and members from the Azure AI Content Safety SDK:

- **`ContentSafetyClient`**: As in the text analysis example, this class is the primary way to communicate with the Azure AI Content Safety service. However, in this case, the focus is on analyzing images. The **`AnalyzeImageAsync`** method is used to send image content for moderation. This method requires an **`AnalyzeImageOptions`** object and returns a **`Response<AnalyzeImageResult>`**. This client can also handle text and video moderation, making it adaptable for various content moderation needs. Beyond basic moderation, developers can also utilize other client methods, such as starting moderation jobs or fetching moderation summaries for more complex scenarios.

- **`AnalyzeImageOptions`**: This class represents the configuration for an image moderation request. It requires an image stream (such as a file stream or a memory stream) and provides options for configuring the moderation settings. You can define additional parameters like blocklists or metadata, depending on the specific needs of your application. This flexibility allows for different types of images to be moderated, from user avatars to more complex media, ensuring that all visual content adheres to your platform's guidelines.

- **`AnalyzeImageResult`**: This class encapsulates the result of an image moderation operation. The primary property, **`IsHarmfulContent`**, is a Boolean value indicating whether the image contains harmful or inappropriate content.

- **`CategoriesAnalysis`**: This property of the **`AnalyzeImageResult`** class provides a list of harmful content categories detected in the image, each associated with a confidence score. It is of type **`ImageCategory`**, an enumeration that exposes the same values shown in Table 2 for the **`TextCategory`** enumeration but targeting images. The categories cover a broad spectrum of harmful content, and the confidence scores allow developers to adjust moderation thresholds based on how certain the AI is about the content. For advanced use cases, the **`GetCategoryDetailsAsync`** method can provide a more granular analysis of the categories detected in the image.

Now that you have learned the meaning of the relevant objects, you can run the sample application.

## Running the application

Press F5 to run the application. Select an image on your local machine and wait for the content safety analysis to be completed. Figure 23 shows an example based on an image that does not contain any offensive content.

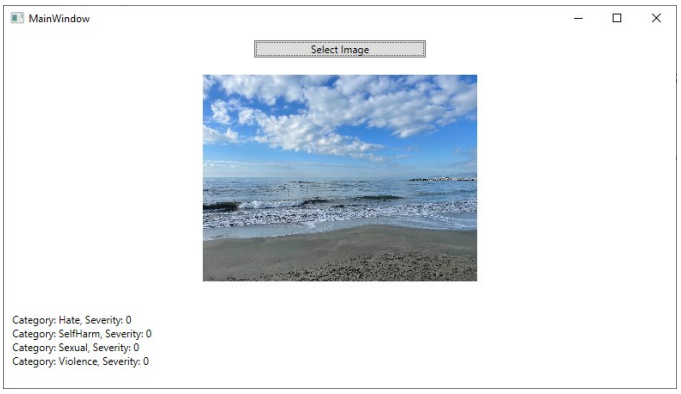



*Figure 23: Analyzing an image for content safety*

As you can see, the application displays the severity level for each category of offensive content. You can try yourself with different images to see the result.

### Hints about content safety analysis on videos

If you want to perform content safety analysis on videos, you need to implement code that extracts individual frames from the video and then analyzes such frames as images. While for the image analysis you can reuse the code shown in Code Listing 9, frame extraction from videos depends on the input format and the libraries you want to use, so it is not possible to provide an example here. However, you know how to approach content safety analysis for videos, too.

## Errors and exceptions

If content analysis fails, the Azure AI Content Safety service can throw the following exceptions:

- **`UnsupportedContentTypeException`**: If the content type provided (such as an unsupported image or text format) is invalid or not supported, this exception is thrown.
- **`ContentViolationException`**: Raised if the provided content violates Azure Content Safety policies, potentially flagging inappropriate or restricted content.

Do not forget to implement a **`try..catch`** block as a best practice for exception handling.

## Chapter summary

The Azure AI Content Safety service is a highly effective tool for automating the moderation of user-generated content across various formats, including text, images, and videos. By leveraging advanced AI models, the service helps businesses and platforms maintain compliance with legal regulations and community guidelines, while also enhancing user safety.

The code examples provided demonstrate how to integrate and use these services in real-world applications using Visual Studio Code. Whether moderating text for offensive language or analyzing video frames for graphic content, Azure AI Content Safety empowers developers to build safer online environments with minimal manual intervention.

# Chapter 7 Azure AI Translator

Microsoft Azure AI Translator is a powerful, cloud-based machine translation service that enables businesses and developers to integrate multilingual translation capabilities into their applications, websites, and business workflows. This chapter describes Azure AI Translator, explaining the available services and providing different code examples.

***Tip: The Azure AI Translator service supports many languages, represented by language codes. The full list of languages and their codes is available in the documentation. In this chapter, only the language codes that are relevant for the code examples will be explained.***

## Introducing Azure AI Translator

The Azure AI Translator service offers a wide range of features for text and document translation in real-time or batch processes. It supports numerous languages and allows for both generic and specialized translation scenarios, such as translating conversational text, technical documents, or entire websites. The service leverages advanced neural machine translation models that continuously improve, providing high-quality translations with better contextual understanding. The service is divided into two main offerings:

- Text translation: This is ideal for real-time scenarios where small text inputs need to be translated quickly and efficiently. It supports features such as transliteration (converting text from one script to another) and dictionary lookup (offering multiple translation options for individual words or phrases). Text translation is suitable for applications like live chat, email translation, and translating short text blocks on websites. This subservice is designed for speed and performance, making it a preferred choice for dynamic content.
- Document translation: This is designed to handle more complex translation scenarios involving entire files or sets of files. It supports a variety of formats, including Word documents, Excel spreadsheets, PowerPoint presentations, and PDFs. This subservice preserves the original document's structure, formatting, and layout during translation, making it ideal for legal contracts, research papers, and marketing materials. It operates on a batch processing model, where files are uploaded to Azure Blob Storage, and then processed in bulk. This makes it suitable for enterprise applications where large volumes of documents need to be translated efficiently.

Azure AI Translator can be accessed through REST APIs, SDKs (like in the case of this ebook), and integrations with other Microsoft products. It offers deployment flexibility with options for cloud-based use and containerization for on-premises environments. Azure AI Translator's capabilities are designed to address various use cases, from simple text translation to handling complex documents requiring contextual translation. The service has a robust infrastructure that enables developers to build multilingual applications for diverse audiences. The primary deployment models include cloud-based translation, where all processes are handled on

Azure's infrastructure, and containerized translation, allowing for local deployment and control. This service also supports the following scenarios:

- Transliteration: This allows converting text from one script to another. For instance, it can convert Arabic script to Latin script, or Devanagari script to Latin script, making the content readable by people who understand the language but use a different script.
- Dictionary Lookup: This feature provides detailed translation options for individual words or short phrases. Instead of translating a full sentence, the dictionary lookup service returns multiple translation options, definitions, and contextual meanings.
- Custom Translator: This allows users to fine-tune Azure's translation models with their own data. This is useful for businesses or applications that need domain-specific terminology, such as in legal, medical, or technical fields.

In the next sections, you will get examples about text and document translations, plus examples about transliteration and dictionary lookup. Custom translators would require an additional publication; for this reason, if you are interested in this topic, the official documentation is a great starting point.

## Configuring the Azure resources

Before diving into the code, you need to set up the Azure AI Translator service on the Azure portal. Once logged in, and following the lesson learned in Chapter 3, click **AI Services** and locate the **Translator** service. Click **Create** under the service card and fill out the necessary details, such as subscription, resource group, and region, as you did for the previous services.

Assign **translator-succinctly** as the service name for consistency with the current example. If the name is not available, provide one of your choosing. Select the **Free** pricing tier and click **Review + Create**. Finally, click **Create** to deploy the service. Once the service is deployed, click **Go to resource** and retrieve your API key and endpoint URL, which will be required for the code examples.

***Note: For the Azure AI Translator service, you will see two endpoints: one with a common URL that should be used for text translation, and one with a custom name, which should be used for document translation. However, using the common URL for text translation will result in a "Forbidden" error. You will then need to use the custom endpoint also for this scenario.***

Actually, you will also need to set up an Azure Blob Storage account for the example about document translation, but this will be explained in the appropriate section.

## Sample application: text translation

The goal of the first example is demonstrating how to translate text from English to Spanish using Azure AI Translator's Text Translation service. For this particular scenario, it is enough to create a Console application.

Start VS Code and open a new instance of the Terminal. Then, type the following commands to create a new Console app inside the root folder for the ebook examples:

```
> md \AIServices\TextTranslationApp
> cd \AIServices\TextTranslationApp
> dotnet new console
> dotnet add package Azure.AI.Translation.Text
```

The last command installs the Azure.AI.Translation.Text NuGet package, a library from the Azure SDK that provides interaction with the Azure AI Translation service from .NET code. When ready, write the code shown in Code Listing 10 (comments will follow shortly).

*Code Listing 10*

```
using Azure.AI.Translation.Text;
using Azure;

class Program
{
    private static readonly string endpoint =
        "your-endpoint";
    private static readonly string apiKey =
        "your-api-key";

    static async Task Main(string[] args)
    {
        var client = new TextTranslationClient(
            new AzureKeyCredential(apiKey),
            new Uri(endpoint));

        Console.WriteLine("Enter text to translate:");
        string inputText = Console.ReadLine();

        var response = await client.TranslateAsync("es", inputText);

        foreach (TranslatedTextItem translation
                 in response.Value)
        {
             Console.WriteLine(
                $"Detected languages of the input text:
                {translation?.DetectedLanguage?.Language}
                with score: {translation?.DetectedLanguage?.Confidence}.");

             Console.WriteLine(
                $"Text was translated to: '{translation?.
                Translations?.FirstOrDefault().TargetLanguage}'
                and the result is: '{translation?.Translations?.
                FirstOrDefault()?.Text}'.");
        }
    }
```

```
}
```

Understanding the code should be quite simple. The **`TextTranslationClient`** class allows for interacting with Azure AI Translator's Text Translation API. You still pass an instance of the **`AzureKeyCredential`** class to provide authentication via the API key.

The **`TranslateAsync`** method translates text from one language to another; the first argument is the target language, whereas the second argument is the input text. The return type is an object of type **`Response<IReadOnlyCollection<TranslatedTextItem>>`**, and the actual collection is stored inside the **`Value`** property. Each item in the collection is a **`TranslatedTextItem`** object, which exposes the **`Translations`** property.

This is an **`IReadOnlyList<TranslationText>`** interface, and the **`TranslationText`** class actually contains the translated text and the detected language with the level of confidence. For simple translations like in the current example, it is sufficient to retrieve the string contained in the **`Text`** property of the first element in the collection.

It is worth mentioning that you are not limited to one language. In fact, the **`TranslateAsync`** method offers an overload that takes an object of type **`TextTranslationTranslateOptions`** as an argument. That allows for specifying multiple languages and multiple strings to be translated. Following is an example that translates a string into Czech, Spanish, and German:

```
TextTranslationTranslateOptions options =
    new TextTranslationTranslateOptions(
            targetLanguages: new[] { "cs", "es", "de" },
            content: new[] { inputText });

var response = await client.TranslateAsync(options);
```

You can then iterate the result like you did previously to get the translation for each language, and you can pass multiple strings to the **`content`** named parameter.

## Running the application

When you run a Console application in VS Code in debug mode, both the debugger output and the application result are shown in the DEBUG CONSOLE panel. This might create confusion, especially if the app is expecting input. For this reason, for this particular example, open an instance of the Terminal and run the application without the debugger, typing the following command:

```
> dotnet run
```

This will first build and then run the application in a clean Terminal window. Obviously, you will still be able to press F5 and start debugging if something is not working as expected. Type in some text in English and press **Enter**. After a few milliseconds, you will get the Spanish translation, as shown in Figure 24.

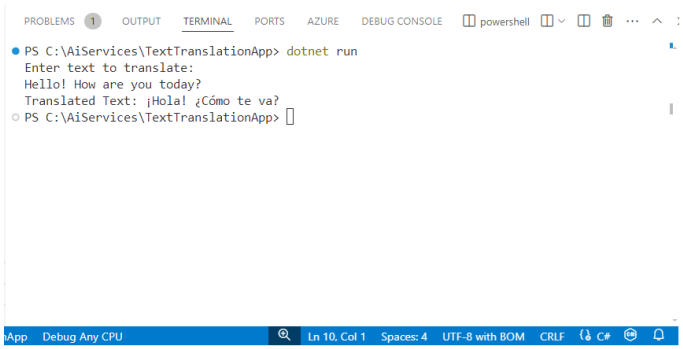



*Figure 24: Text translation with Azure AI Translator*

It is easy to understand the potential of this service, which can translate complex sentences from one language to another in milliseconds.

## Sample application: transliteration

As mentioned at the beginning of this chapter, transliteration is the process of converting text from one script to another. Converting Arabic script to Latin script, or Devanagari script to Latin script, are examples of transliteration. This allows for making the content readable by people who understand the language but use a different script.

Now, use the well-known commands to create a new Console app called **TextTransliterationApp** and install the Azure.AI.Translation.Text NuGet package as you did for the first example. The goal of this example is transliterating text from Vietnamese script to Latin script using the Azure AI Translator service.

***Note: To avoid mistakes due to language knowledge, the source script has been taken from the official Microsoft documentation.***

To accomplish this, write the code shown in Code Listing 11. As usual, explanations will follow shortly.

*Code Listing 11*

```
using Azure.AI.Translation.Text;
using Azure;

class Program
{
    private static readonly string endpoint =
        "your-endpoint";
    private static readonly string apiKey =
        "your-api-key";

    static async Task Main(string[] args)
    {
```

```
        Console.OutputEncoding = System.Text.Encoding.UTF8;

        var client = new TextTranslationClient(
            new AzureKeyCredential(apiKey), new Uri(endpoint));

        // Vietnamese text taken from the MS documentation
        // Translates to "This is the problem" as per Google Translator.
        string inputText = "这是个测试。";

        Response<IReadOnlyList<TransliteratedText>> response =
            await client.TransliterateAsync(language: "zh-Hans",
            fromScript: "Hans",
            toScript: "Latn", inputText);
        IReadOnlyList<TransliteratedText> transliterations =
            response.Value;
        TransliteratedText transliteration =
            transliterations.FirstOrDefault();
        Console.WriteLine($"Transliterated Text: " +
            $"{transliteration.Text}");
        Console.ReadLine();
    }
}
```

The approach is very similar to what you saw for text translation. The **`TextTranslationClient`** class allows for interacting with Azure AI Translator's Text Translation API. You still pass an instance of the **`AzureKeyCredential`** class to provide authentication via the API key. The **`TransliterateAsync`** method transliterates from one script to another; the first argument is the source language, the second argument is the source script, the third argument is the target script, and the fourth argument is the input text.

The return type is an object of type **`Response<IReadOnlyCollection<TransliteratedText>>`**, and the actual collection is stored inside the **`Value`** property. For simple transliterations like in the current example, it is sufficient to retrieve the string contained in the **`Text`** property of the first element in the collection. For this scenario, as well as for text translation, you are not limited to one language and script. In fact, the **`TransliterateAsync`** method offers an overload that takes an object of type **`TextTranslationTransliterateOptions`** as an argument and that allows for specifying multiple strings to be transliterated. Following is an example:

```
TextTranslationTransliterateOptions options =
    new TextTranslationTransliterateOptions(language: "zh-Hans",
    fromScript: "Hans",
    toScript: "Latn",
    new string[] { inputText });
var transliteration = await client.TransliterateAsync(options);
```

You can then parse the result like you did previously to get the transliteration targets.

***Tip: Notice how the code assigns the UTF-8 encoding to the console output. This ensures that text coming from other scripts is formatted properly.***

## Running the application

With the same purpose of avoiding confusion in the output panel as the first example, open an instance of the Terminal and run the application without the debugger, typing the following command:

```
> dotnet run
```

This will first build and then run the application in a clean Terminal window. You can always press F5 and start debugging if something is not working. The text to be transliterated is hard-coded, so you will directly get the result shown in Figure 25.

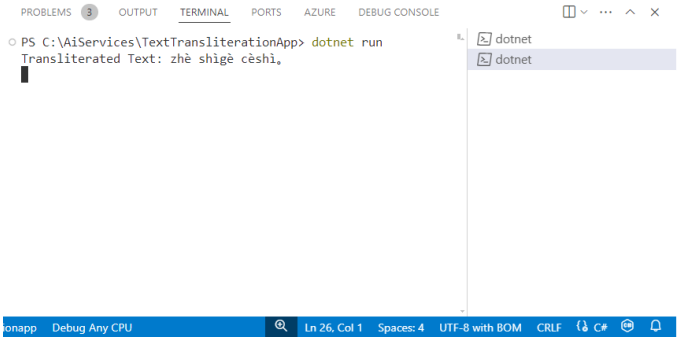



*Figure 25: Text transliteration with Azure AI Translator*

This is another very powerful feature, because it allows people with different linguistic backgrounds to better understand the context of a written conversation.

# Sample application: dictionary lookup

At the beginning of this chapter, we mentioned that dictionary lookup is a feature that provides detailed translation options for individual words or short phrases. This feature is designed to provide these detailed translations along with example sentences and multiple translation options instead of translating a full sentence. The dictionary lookup is made of two parts: entry search, which allows for finding entries related to the given words, and example search, which finds examples based on the retrieved translations.

The goal of the next example is combining both parts by searching the Azure dictionary to find Spanish translations for a single English word and then to find sample sentences based on the most relevant translation. To accomplish this, open a new Terminal instance in VS Code and create a new Console application called **DictionaryLookupApp** with the following commands:

```
> md \AIServices\DictionaryLookupApp
> cd \AIServices\DictionaryLookupApp
> dotnet new Console
> dotnet add package Azure.AI.Translation.Text
```

As you can see, you still install the same NuGet package of the previous examples. When ready, enter the code shown in Code Listing 12, which contains comments and is followed by explanations.

*Code Listing 12*

```
using Azure;
using Azure.AI.Translation.Text;

class Program
{
    private static readonly string endpoint =
        "your-endpoint";
    private static readonly string apiKey =
        "your-api-key";

    static async Task Main(string[] args)
    {
        // Create a client for the Azure AI Translator Text service.
        var client = new TextTranslationClient(
            new AzureKeyCredential(apiKey), new Uri(endpoint));

        string sourceLanguage = "en";
        string targetLanguage = "es";
        string inputText = "fly";

        // Step 1: Look up dictionary entries for the
        // given word (from English to Spanish).
        Response<IReadOnlyList<DictionaryLookupItem>> entriesResponse =
            await client.LookupDictionaryEntriesAsync(sourceLanguage,
            targetLanguage, inputText).ConfigureAwait(false);
        IReadOnlyList<DictionaryLookupItem> dictionaryEntries =
            entriesResponse.Value;
        DictionaryLookupItem dictionaryEntry =
            dictionaryEntries.FirstOrDefault();

        if (dictionaryEntry != null &&
            dictionaryEntry.Translations.Any())
        {
            Console.WriteLine($"For the given input '{inputText}', " +
                $"{dictionaryEntry.Translations.Count} " +
                $"entries were found in the dictionary.");

            // Display the first entry and its confidence.
```

```
            var firstTranslation = dictionaryEntry.
                Translations.FirstOrDefault();
            Console.WriteLine($"First entry: '
                {firstTranslation?.DisplayTarget}', " +
                $"confidence: {firstTranslation?.Confidence}.");

            // Step 2: Fetch example sentences for the first translation.
            IEnumerable<InputTextWithTranslation> inputTextElements = new[]
            {
                new InputTextWithTranslation(inputText,
                firstTranslation.DisplayTarget)
            };

            Response<IReadOnlyList<DictionaryExampleItem>>
                examplesResponse =
                await client.LookupDictionaryExamplesAsync(
                sourceLanguage,
                targetLanguage,
                inputTextElements
            ).ConfigureAwait(false);

            IReadOnlyList<DictionaryExampleItem> examples =
                examplesResponse.Value;
            DictionaryExampleItem exampleEntry = examples.FirstOrDefault();
            if (exampleEntry != null && exampleEntry.Examples.Any())
            {
                Console.WriteLine(
                    $"\nExamples for translation '
                    {firstTranslation.DisplayTarget}':");
                foreach (var example in exampleEntry.Examples)
                {
                    Console.WriteLine($" - Source:
                        $"{example.SourcePrefix} " +
                        $"{inputText} {example.SourceSuffix}");
                    Console.WriteLine(
                        $"   -> Target: {example.TargetPrefix} " +
                        $"{firstTranslation.
                           DisplayTarget} {example.TargetSuffix}");
                }
            }
            else
            {
                Console.WriteLine("No examples found.");
            }
        }
        else
        {
            Console.WriteLine("No translations found.");
        }
```

```
        Console.ReadLine();
    }
}
```

Following is an explanation of the types and members used in Code Listing 12:

- The **`TextTranslationClient`** is the primary class for interacting with the Azure AI Translator service, and it works like in the previous examples, including authentication.
- The **`LookupDictionaryEntriesAsync`** method retrieves dictionary entries for a given word or phrase, translating it from one language to another. The method signature includes the following parameters: **`sourceLanguage`**, the language code of the input text (e.g., **`en`** for English); **`targetLanguage`**, the language code of the output text (e.g., **`es`** for Spanish); **`inputText`**, the word or phrase for which translations are requested. The method returns a **`Response<IReadOnlyList<DictionaryLookupItem>>`**, where **`DictionaryLookupItem`** contains the possible translations for the input text, along with additional information such as part of speech, confidence scores, and grammatical details. This allows the application to present multiple translation options for the user.
- The **`DictionaryLookupItem`** class represents a single dictionary entry returned by the **`LookupDictionaryEntriesAsync`** method. Each **`DictionaryLookupItem`** exposes the following properties: **`Translations`**, a list of possible translations, represented by a **`TranslationItem`** object. Each translation includes the translated text, part of speech, and confidence score; **`DisplaySource`**, the original word or phrase in the source language; **`DisplayTarget`**, the translated word or phrase in the target language. This class is essential for handling and organizing the translation results, which can include multiple translations for a single input word.
- The **`TranslationItem`** class represents a single translation within a **`DictionaryLookupItem`**. It exposes the following members: **`DisplayTarget`**, the translated word or phrase; **`PosTag`**, the part of speech of the translated word (such as noun, verb); **`Confidence`**, a numerical value indicating the confidence of the translation, typically ranging from 0 to 1. This class allows us to extract and display detailed information about each translation, including linguistic nuances like part of speech and the certainty of the translation.
- The **`LookupDictionaryExamplesAsync`** method retrieves example sentences for a given word or phrase, showing how the word is used in context. It requires the following arguments: **`sourceLanguage`**, the language code of the input text; **`targetLanguage`**, the language code of the translation; **`inputTextElements`**, a collection of **`InputTextWithTranslation`** elements that contain both the source text and its translation. The method returns a **`Response<IReadOnlyList<DictionaryExampleItem>>`**, where each **`DictionaryExampleItem`** contains examples of how the word and its translation are used in real sentences. This method helps provide context for the translation, which can be valuable in understanding nuances and proper usage.
- The **`InputTextWithTranslation`** class represents a pairing of a source text and its translation. When calling the **`LookupDictionaryExamplesAsync`** method, you provide an **`InputTextWithTranslation`** for the word you are querying and its corresponding translation. This ensures that the examples are specific to the chosen translation.

- The **`DictionaryExampleItem`** class represents a set of example sentences returned by the **`LookupDictionaryExamplesAsync`** method. Each **`DictionaryExampleItem`** exposes the following properties: **`Examples`**, a list of **`DictionaryExample`** objects, each representing an individual example sentence; **`SourcePrefix`** and **`SourceSuffix`**, representing text that appears before and after the target word in the source language; **`TargetPrefix`** and **`TargetSuffix`**, representing text that appears before and after the target word in the target language.
- The **`DictionaryExample`** class represents a single example sentence from the list in **`DictionaryExampleItem`**. It exposes the following properties: **`SourcePrefix`**, **`SourceSuffix`**, **`TargetPrefix`**, and **`TargetSuffix`**, with the same purpose as described in the previous point. This class allows you to see how a word or phrase is used in context, which can be particularly helpful when learning how to properly use a new language.

Now that you have a clear understanding of the objects used for dictionary lookup, it is time to run the application and see the result of this complex work.

## Running the application

With the same purpose of avoiding mixing the debugger output with the application output, run the application by typing **`dotnet run`** inside an instance of the Terminal. Figure 26 shows the result of the previous code.

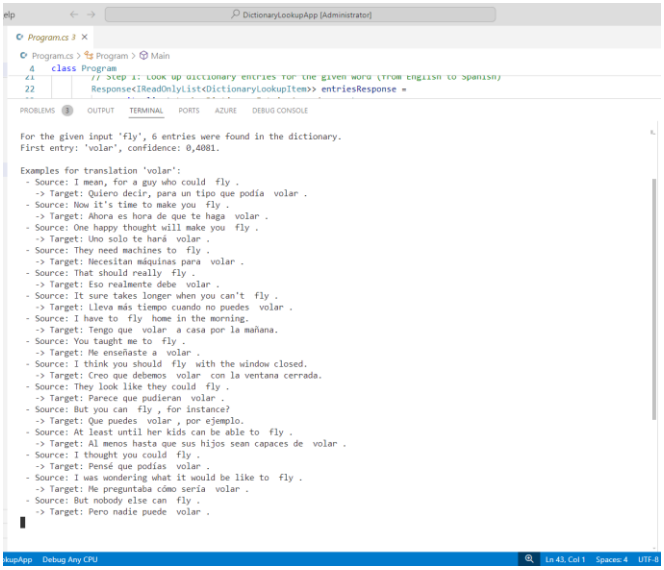



*Figure 26: Dictionary lookup with Azure AI Translator*

As you can see, the application provides possible translations for the English word “fly,” and then it shows example sentences for the most relevant translation.

# Sample application: document translation

Document translation allows for translating entire documents while preserving the original formatting, structure, and images. It is particularly useful when you need to translate complex documents like contracts, reports, or presentations in various formats (such as .docx, .pdf, and .pptx). Document translation is powered by the same neural machine translation technology used in the text translation API, but it processes whole documents rather than individual pieces of text. The service is typically designed to work with Azure Blob Storage for uploading, storing, and processing documents. The process involves uploading source documents to a Blob Storage container and specifying the target languages. Then, the translated documents are stored in another container for efficiency purposes.

In this section, you will create a sample Console application that invokes the Azure AI Translator API to translate an English document into Spanish. For consistency with the example, you can use the Word document attached to the companion solution, which contains the unedited first page of Chapter 2 of this ebook with all the formatting. This allows for demonstrating how the translation process preserves the original formatting, other than translating the document. You will also learn to set up an Azure Blob Storage account as the document store. This is explained in the next paragraphs.

## Brief introduction to Azure Blob Storage

***Note: BLOB is the acronym for Binary Large Object, and it has become a common term in the information technology to represent complex binary data such as large files.***

Azure Blob Storage is a cloud storage solution designed for storing large amounts of unstructured data, such as text or binary data. It is highly scalable, secure, and accessible via HTTP/HTTPS. Blob Storage is optimized for handling data like documents, images, video files, backups, and more. It's commonly used for scenarios such as storing files for distributed access, streaming video and audio, serving images or documents directly to a browser, and storing data for backup. Azure Blob Storage supports three main types of blobs:

- Block blobs: Used to store text and binary data. This is the most commonly used blob type for general-purpose storage.
- Append blobs: Optimized for append operations, making them ideal for logging.
- Page blobs: Used for storing virtual hard disk (VHD) files.

For document translations, you will use block blobs. For more information, visit the Azure Blob Storage documentation page.

## Setting up the Azure Blob Storage

In order to take advantage of the document translation feature from Azure AI Translator, you need to create an Azure Blob Storage where documents are uploaded for translation and saved after translation. To accomplish this, open the Azure Portal, and in the dashboard, locate the **Storage accounts** service (see Figure 27).

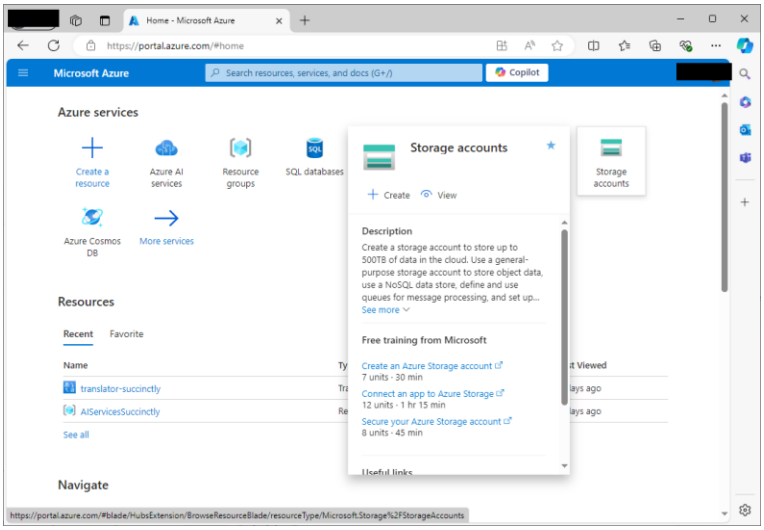



*Figure 27: Locating the Storage accounts service in the Azure Portal dashboard*

Click **Create**. When the Storage accounts page opens, click **Create**. At this point, keep Figure 28 as a reference and enter the following information:

- Subscription: Choose your subscription.
- Resource group: Select the existing resource group created previously.
- Storage account name: Enter a unique name, which must be lowercase and a maximum of 24 characters long.
- Region: Choose the region that is closest to you.
- Performance: Choose Standard for general-purpose usage.
- Redundancy: Select **Locally-redundant storage** (LRS) unless you need higher availability options, like geo-redundant storage (GRS).

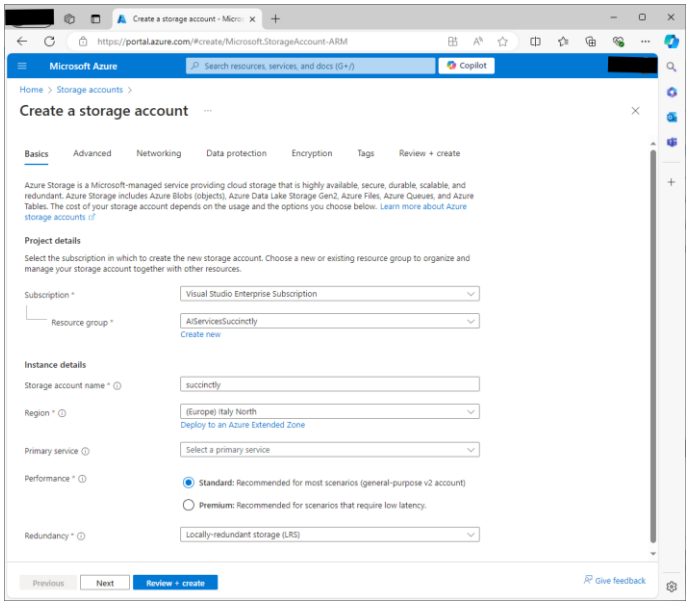



*Figure 28: Assigning settings to the new Azure Storage account*

When ready, click **Review + Create** > **Create**. After a few minutes, the service is deployed, so click **Go to resource** when the deployment completion page appears. At this point, expand the **Security + networking** node in the left-handed menu, and then click **Access keys**. When the Access keys page appears (see Figure 29), locate the **Key** text box, click **Show**, and then copy it to the clipboard for later reuse.

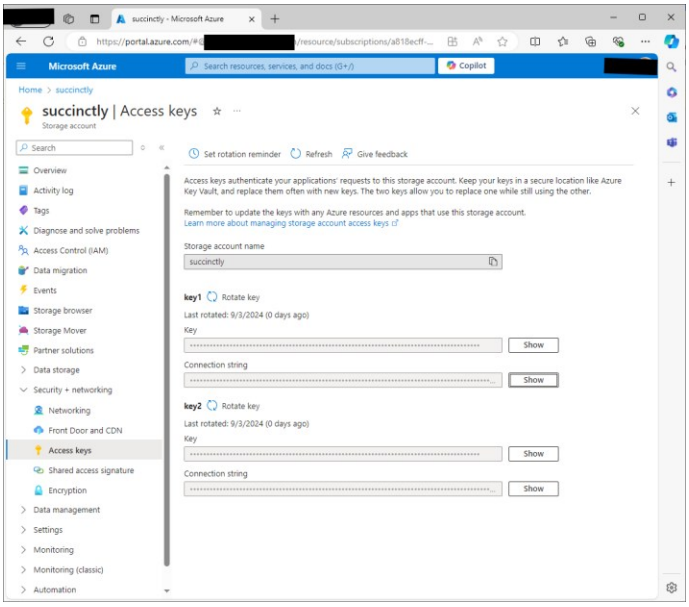



*Figure 29: The security options for the Azure Blob Storage*

The API key will be needed in the code example to connect to the Azure Blob Storage from C#. Now you need to create two additional resources: containers, which represent the actual storage for your documents, and shared access signatures (SAS) to grant temporary access to documents via tokens.

## Creating containers

Containers in the Azure Blob Storage organize your blobs into logical units. For the document translation example, you need two containers: one for the source document, and one for the translated document. In the left-hand menu, navigate to **Containers** (see Figure 30) and click **+Container**.

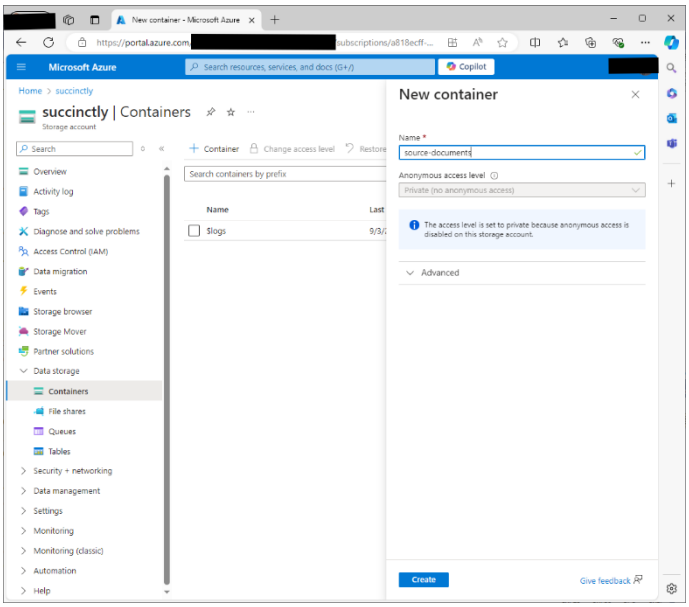



*Figure 30: Creating a new container*

Enter a name for the container that stores source documents, for example **source-documents**. Make sure that the access level is set as private (see Figure 30), and then click **Create**. When the new container appears in the list, click its name. At this point, you will be able to see its details and to upload files by clicking the Upload button.

A convenient user interface will help you upload the documents you want to translate. For consistency with this ebook, you can upload the sourcedoc.docx file included in the companion solution (see Figure 31).

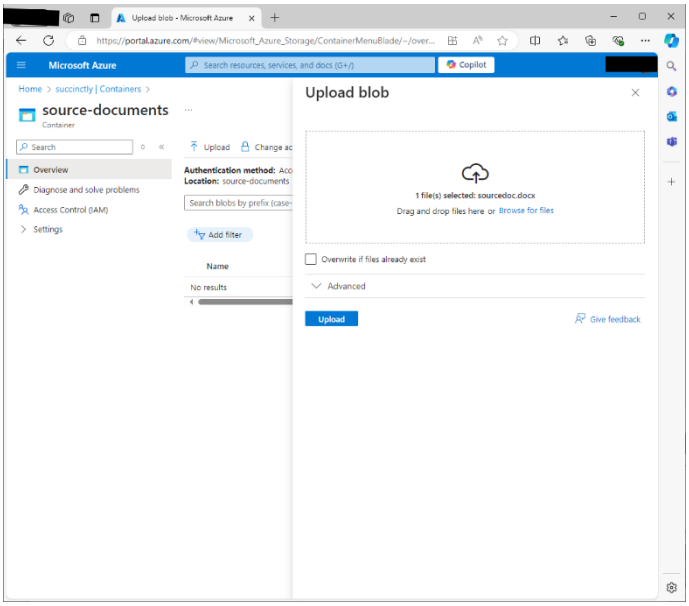



*Figure 31: Uploading documents to the Blob Storage*

Once uploaded, navigate back to the containers page and repeat the steps to create a new container called translated-documents. For now, this is all you need for file management. The next step is about setting up access keys.

## Defining shared access signatures

To allow Azure AI Translator to access your containers, you will need to generate shared access signature (SAS) tokens for both containers. These tokens provide secure and temporary access to your blobs. In order to create SAS tokens, navigate again to the **Containers** page. Then, follow these steps:

1. Click the **source-documents** container.
2. When the container page appears, expand the **Settings** node and click **Shared access tokens** in the left-hand menu.
3. In the **Shared access tokens** page (see Figure 32), leave unchanged the proposed options. More specifically, the source-documents container should have read-only permissions, and permissions should have limited time; with the default options, it is 24 hours.
4. For the current example, do not enter any IP address. This is something that your network administrator will do in the real world.

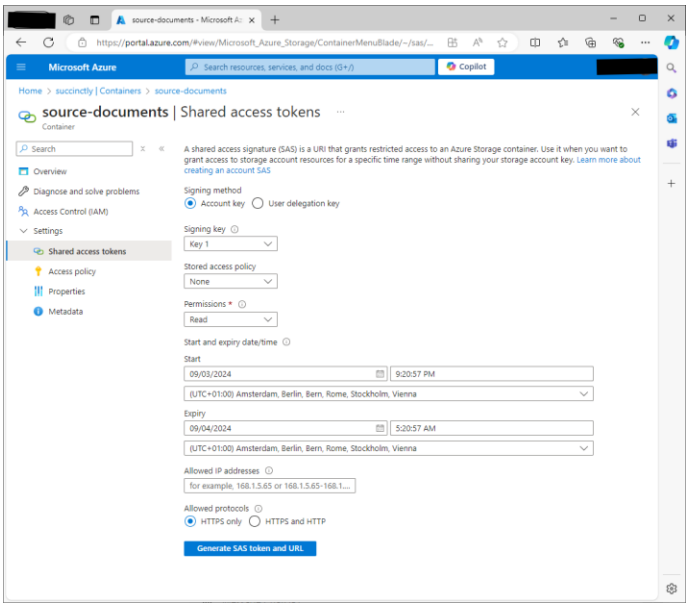



*Figure 32: Creating SAS tokens*

When finished, click **Generate SAS token and URL**. You will get a token and a service URL, which enable permissions over containers. You need to store the generated URL, as it will be needed in the source code to identify and reach the resource. Repeat the same steps for the translated-documents container, but make sure to also add the write permission. Now you have all the necessary resources, and you can create a sample application.

## Creating a Console app

Open a new Terminal instance in VS Code and create a new Console app called **DocumentTranslatorApp** with the following commands:

```
> md \AIServices\DocumentTranslatorApp
> cd \AIServices\DocumentTranslatorApp
> dotnet new console
> dotnet add package Azure.AI.Translation.Document
```

The Azure.AI.Translation.Document NuGet package is required to work with document translation features. When ready, open the project folder in Visual Studio Code and add the code shown in Code Listing 13 to the Program.cs file. Comments will follow shortly.

*Code Listing 13*

```
using Azure;
using Azure.AI.Translation.Document;

class Program
{
    // Define constants for your Azure Translator resource.
    private static readonly string endpoint =
        "your-ai-translator-endpoint";
    private static readonly string apiKey =
        "your-ai-translator-api-key";
    static async Task Main(string[] args)
    {
        // Create the Document Translation client.
        var client = new DocumentTranslationClient(
            new Uri(endpoint), new AzureKeyCredential(apiKey));

        Uri sourceSasUri = new Uri("your-source-sas-url");
        Uri targetSasUri = new Uri("your-target-sas-url");

        // Create a translation operation.
        DocumentTranslationInput input =
            new DocumentTranslationInput(sourceSasUri,
            targetSasUri, "es"); // Translate to Spanish.

        Console.WriteLine("Translation started...");

        DocumentTranslationOperation operation =
            await client.StartTranslationAsync(input);

        // Wait for the translation to complete.
```

```
        await operation.WaitForCompletionAsync();

        Console.WriteLine("Document translation completed.");
        Console.ReadLine();
    }
}
```

The **`DocumentTranslationClient`** class allows you to interact with the Azure Document Translation service, and it takes an instance of the **`AzureKeyCredential`** class as an argument to provide authentication. This client is responsible for initiating and managing translation operations. The **`DocumentTranslationInput`** class encapsulates all the essential information required to perform a document translation. It holds references to the source and target Blob Storage containers, specified as URIs, and the language into which the documents should be translated.

The **`sourceSasUri`** and **`targetSasUri`** variables represent the URLs of the source document container and translated document container, respectively. These variables will be filled with the URLs with the appended token that you generated previously in the **Defining Shared Access Signatures** paragraph.

The **`StartTranslationAsync`** method of **`DocumentTranslationClient`** initiates the translation by starting a long-running operation. It returns an object of type **`DocumentTranslationOperation`** that allows you to track the status of the translation as it progresses. The sample code invokes the **`WaitForCompletionAsync`** method to asynchronously wait for the translation operation to complete.

## Running the application

The output of the sample application is very simple, as it shows two informational messages before and after the translation. You are then free to run the application by either pressing F5 (with the debugger), or by typing **`dotnet run`** in the Terminal. Figure 33 shows the simple output from the application.

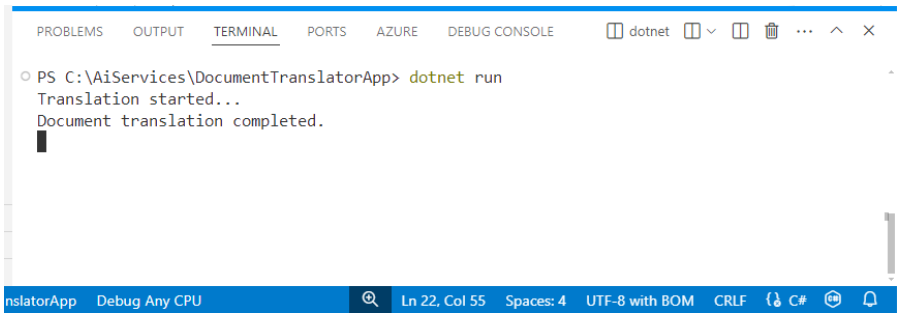



*Figure 33: Running the sample application*

If everything succeeds, you will find your translated document in the translated-documents container that you created previously. Figure 34 shows how to retrieve the translated document.

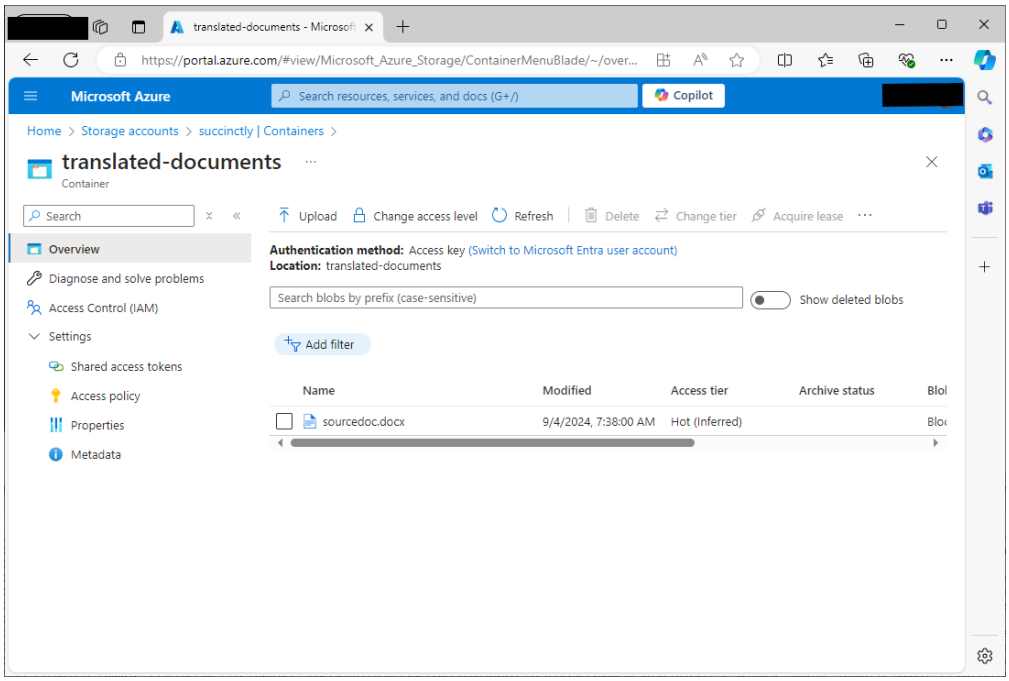



*Figure 34: The translated document is available*

You can click the file name to access a details page, where you will also find a Download button. The result of the document translation is shown in Figure 35.

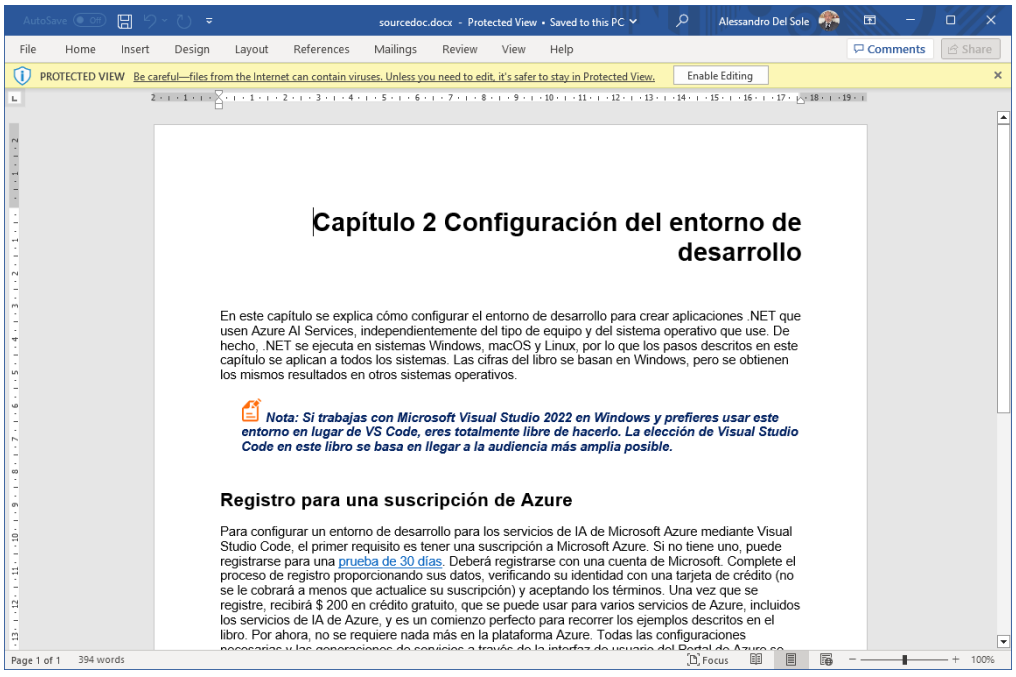



*Figure 35: The translated document preserving formatting*

As you can see, not only has the Azure document translator feature translated the document into a different language, but it also has preserved all the source formatting (including hyperlinks). As you can imagine, this is an extremely powerful feature that can literally revolutionize the way companies manage documents, especially if they have customers abroad.

## Errors and exceptions

If translation fails, the Azure AI Translation service can throw the following exceptions:

- **`TranslationLanguageNotSupportedException`**: Raised when the language you are trying to translate to or from is not supported by the Translator service.
- **`QuotaExceededException`**: Thrown when the service usage exceeds the subscription's quota, such as the character or document limit for translation.

Do not forget to implement a **`try..catch`** block as a best practice for exception handling.

## Chapter summary

In this chapter about Azure AI Translator, you have discovered two main services: text translation and document translation. Text translation is ideal for the real-time translation of small text snippets, while document translation is designed for translating larger documents while preserving their formatting.

The code examples showed practical implementations of text translation, transliteration, dictionary lookup, and document translation, all within Visual Studio Code. By using Azure AI Translator, developers can enhance their applications with advanced multilingual capabilities, making them accessible to a global audience.

# Chapter 8 Azure AI Speech

Microsoft Azure AI Speech is a powerful service that focuses on delivering cutting-edge speech recognition, synthesis, and translation capabilities. In this chapter, you will learn about all the available services and how to implement them in your .NET projects so that you will be able to create voice-enabled applications.

## Introducing Azure AI Speech

The Microsoft Azure AI Speech service leverages advanced machine learning models to convert speech into text, synthesize natural-sounding speech from text, and translate speech into various languages in real-time. This service is designed to integrate seamlessly into various applications, providing developers with the tools they need to build voice-enabled applications, create conversational AI experiences, and enhance accessibility with speech-to-text and text-to-speech functionalities.

The service supports a wide range of use cases, including voice-activated assistants, transcription services, real-time language translation, and the generation of audio content from textual data. Azure AI Speech offers high accuracy and is continually updated to support new languages, dialects, and scenarios, making it a versatile tool for global applications. The service is built on Microsoft’s proprietary deep neural networks, ensuring that it delivers fast, reliable, and high-quality results. Like for the other services, developers can use REST APIs, SDKs, and client libraries to interact with the service, enabling easy integration into existing workflows and applications.

Furthermore, Azure AI Speech supports customizable speech models, allowing businesses to fine-tune the service according to their specific needs, such as adapting to industry-specific terminology or enhancing recognition accuracy for certain accents. Azure AI Speech provides three primary functionalities: speech-to-text, text-to-speech, and speech translation, as described shortly.

### Speech-to-text

This feature allows applications to convert speech into text. It supports real-time transcription and batch processing of audio files, making it suitable for various applications, including meeting transcriptions, voice command processing, and accessibility services. The speech-to-text service is highly accurate, benefiting from continuous improvements in Microsoft's AI models. It supports multiple languages and can be customized using custom models to improve accuracy in specific contexts, such as industry jargon or regional accents.

### Text-to-speech

This capability synthesizes natural-sounding speech from text, making it ideal for generating spoken content dynamically. The text-to-speech service can be used in various scenarios, including creating voice responses in chatbots, generating audio ebooks, and providing vocal guidance in applications. It supports numerous languages and voices, including neural voices that provide more natural intonations and expressions. Custom voice fonts can also be created to match a brand's unique voice, ensuring consistency across various channels.

### Speech translation

Azure AI Speech also offers real-time translation of speech into another language, enabling cross-lingual communication in applications. This service can be integrated into multilingual chat applications, global conferencing tools, and customer support systems to facilitate communication between speakers of different languages. It supports a wide range of language pairs and delivers translations with high accuracy and low latency, making it suitable for real-time communication scenarios.

### Additional features

In addition to these core features, Azure AI Speech offers features such as profanity filtering, punctuation addition, and the ability to identify multiple speakers in a conversation (speaker diarization). The service also integrates with other Azure services, such as Azure AI Language, for enhanced functionality like sentiment analysis or key phrase extraction from transcribed text. These additional features will not be covered in this chapter, so you can refer to the provided documentation links.

## Creating the required Azure resources

Before diving into three sample applications, one per feature, you need to create an Azure resource for the AI Speech service. The steps are exactly the same as for the previous services and can be summarized as follows:

1. Log in to the Azure Portal and click **AI Services** in the dashboard.
2. Locate the Speech service and click **Create** on its card to begin the setup.
3. Choose your subscription and the resource group created at the beginning of this ebook.
4. Select your closest region and provide a name for the service, for example**,** speech-succinctly for consistency with the current examples.
5. Choose the **Free** pricing tier.
6. Click **Review + Create** and then **Create** to deploy the service.

Also remember to copy the name of the selected region and the API key. These will both be required to authenticate against the service in C# code.

## Sample application: speech-to-text

The goal of the first sample application is to demonstrate how to convert speech into text. This will be accomplished by creating a WPF application that loads an existing audio file containing spoken sentences. A .wav audio file is included with the companion solution and contains the following sentence: “Hey! How are you doing today? I hope you are doing great!” Obviously, feel free to select a different audio file.

Open a Terminal instance in Visual Studio Code and create a new project with the following commands:

```
> md \AIServices\AzureSpeechToText
> cd \AIServices\AzureSpeechToText
> dotnet new wpf
> dotnet add package Microsoft.CognitiveServices.Speech
```

The Microsoft.CognitiveServices.Speech Nuget package allows for interacting with the Azure AI Speech service from .NET and is common to all the examples described in this chapter. Open the folder containing the new project and get ready to write code.

### Defining the user interface

The user interface for the first sample project is very simple. It contains a button that allows for loading an audio file, a label that displays the process status, and a text box that shows the result of the conversion. Code Listing 14 contains the XAML code that needs to be added to the MainPage.xaml file.

*Code Listing 14*

```
<Window x:Class="AzureSpeechToText.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Azure Speech to Text" Height="200" Width="400">
    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Button Name="BtnOpenFile" Content="Open Audio File"
                Width="120" Height="30" Click="BtnOpenFile_Click"/>
        <TextBox Name="TxtTranscription" Width="360" Height="100"
                 Margin="0,10,0,0" TextWrapping="Wrap" Grid.Row="1"/>
        <Label Name="LblStatus" Content="Status: Idle" Margin="0,10,0,0"
               Grid.Row="2" />
    </Grid>
</Window>
```

## Adding speech-to-text capabilities in C#

Code Listing 15 contains the code that allows for loading and analyzing the audio file. Comments will follow shortly.

*Code Listing 15*

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.Win32;
using System.Windows;

namespace AzureSpeechToText
{
    public partial class MainWindow : Window
    {
        private string subscriptionKey =
            "your-api-key";
        private string region =
            "your-region";

        public MainWindow()
        {
            InitializeComponent();
        }

        private async void BtnOpenFile_Click(
            object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog =
                new OpenFileDialog
            {
                Filter =
                "Audio Files (*.wav)|*.wav|All files (*.*)|*.*"
            };
            if (openFileDialog.ShowDialog() == true)
            {
                LblStatus.Content = "Status: Processing";
                string audioFilePath =
                    openFileDialog.FileName;
                txtTranscription.Text =
                    await TranscribeAudioAsync(audioFilePath);
                LblStatus.Content = "Status: Complete";
            }
        }

        private async Task<string>
            TranscribeAudioAsync(string audioFilePath)
        {
```

```
            var config = SpeechConfig.FromSubscription(
                subscriptionKey, region);
            using var audioInput =
                AudioConfig.FromWavFileInput(audioFilePath);
            using var recognizer =
                new SpeechRecognizer(config, audioInput);

            var result = await recognizer.RecognizeOnceAsync();
            return result.Text;
        }
    }
}
```

Following is a description of the types and members from the Azure SDK for AI Speech that are relevant to the example:

- The **`SpeechConfig`** class is crucial for setting up the connection to Azure AI Speech services. The **`FromSubscription`** method initializes the **`SpeechConfig`** with your Azure subscription key and region, which is necessary to authenticate the service. Additionally, the **`SpeechRecognitionLanguage`** property is used to specify the language in which speech will be recognized, such as **`en-US`** for English.
- The **`AudioConfig`** class manages audio inputs and outputs. In this case, the **`FromDefaultMicrophoneInput`** method is used to capture audio from the default microphone of the system. **`AudioConfig`** can also be initialized from other sources, such as **`FromWavFileInput`**, for reading audio from a WAV file, or **`FromStreamInput`**, for capturing audio from a stream.
- The **`SpeechRecognizer`** class connects your application to Azure AI Speech for real-time speech recognition. It requires both **`SpeechConfig`** and **`AudioConfig`** objects for initialization. The **`RecognizeOnceAsync`** method is used in the example to perform single-shot recognition, which means it listens for a single statement and returns the recognized text. The **`SpeechRecognizer`** class also supports continuous recognition via the **`StartContinuousRecognitionAsync`** method, which allows the application to keep listening and transcribing speech until explicitly stopped by **`StopContinuousRecognitionAsync`**. This can be useful for transcribing long conversations or speeches.
- The **`SpeechRecognitionResult`** class represents the outcome of a speech recognition operation. The **`Reason`** property is of type **`ResultReason`** and indicates whether the recognition was successful (**`RecognizedSpeech`**) or encountered issues like **`NoMatch`** or **`Canceled`**. The recognized text is typically accessed through the **`Text`** property. Table 3 summarizes possible values for the **`Reason`** property.
- The **`VoiceInfo`** class represents individual voices available for speech synthesis. Its **`ShortName`** property identifies each voice by a concise identifier (such as **`en-US-AriaNeural`**), which is used to set the **`SpeechSynthesisVoiceName`** in the **`SpeechConfig`** instance.

*Table 3: Values from the ResultReason enumeration (source: Microsoft)*

| Value | Description |
|---|---|
| **NoMatch** | Indicates speech could not be recognized. |
| **Canceled** | Indicates the recognition was canceled. |
| **RecognizingSpeech** | Indicates the speech result contains hypothesis text. |
| **RecognizedSpeech** | Indicates the speech result contains final text that has been recognized. |
| **RecognizingIntent** | Indicates the intent result contains hypothesis text and intent. |
| **RecognizedIntent** | Indicates the intent result contains final text and intent. |
| **TranslatingSpeech** | Indicates the translation result contains hypothesis text and its translations. |
| **TranslatedSpeech** | Indicates the translation result contains final text and corresponding translations. |
| **SynthesizingAudio** | Indicates the synthesized audio result contains a non-zero amount of audio data. |
| **SynthesizingAudioCompleted** | Indicates the synthesized audio is now complete for this phrase. |
| **RecognizingKeyword** | Indicates the speech result contains (unverified) keyword text. |
| **RecognizedKeyword** | Indicates that keyword recognition completed recognizing the given keyword. |
| **SynthesizingAudioStarted** | Indicates the speech synthesis is now started. |

| Value | Description |
| --- | --- |
| `TranslatingParticipantSpeech` | Indicates the transcription result contains hypothesis text and its translations for other participants in the conversation. |
| `TranslatedParticipantSpeech` | Indicates the transcription result contains final text and corresponding translations for other participants in the conversation. |
| `TranslatedInstantMessage` | Indicates the transcription result contains the instant message and corresponding translations. |
| `TranslatedParticipantInstantMessage` | Indicates the transcription result contains the instant message for other participants in the conversation and corresponding translations. |
| `EnrollingVoiceProfile` | Indicates the voice profile is being enrolled and more audio is needed to complete a voice profile. |
| `EnrolledVoiceProfile` | Indicates the voice profile has been enrolled. |
| `RecognizedSpeakers` | Indicates successful identification of some speakers. |
| `RecognizedSpeaker` | Indicates successful verification of a speaker. |
| `ResetVoiceProfile` | Indicates a voice profile has been reset. |
| `DeletedVoiceProfile` | Indicates a voice profile has been deleted. |
| `VoicesListRetrieved` | Indicates the voices list has been retrieved successfully. |

***Note: The official documentation contains the full list of synthesized voices and supported languages.***

## Running the application

You can now run the application by pressing F5 for debugging. This is useful in case something is not working as expected. When the application starts, select an audio file that contains spoken sentences, and then wait for the operation to be completed. Figure 36 shows the result based on the audio file attached to the companion solution.

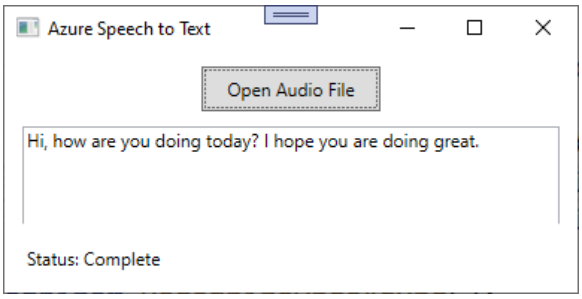



*Figure 36: Speech has been converted to text*

As you can see, with extremely limited effort, you were able to transcribe spoken sentences into text. Now you can take a step further by doing the opposite work: converting written text into speech.

# Sample application: text-to-speech

In this example, you will create a WPF application that allows for entering text and converts this into speech using the Azure AI Speech text-to-speech functionality. The application will also allow users to select a voice and then play the synthesized speech. You will understand how to generate natural-sounding speech from text, which can be useful in scenarios such as generating audio content, creating voice interfaces, or assisting visually impaired users.

Having said this, in Visual Studio Code, open an instance of the Terminal and create a new WPF project with the following commands:

```
> md \AIServices\AzureTextToSpeech
> cd \AIServices\AzureTextToSpeech
> dotnet new wpf
> dotnet add package Microsoft.CognitiveServices.Speech
```

Now you are ready to define the user interface.

## Defining the user interface

The user interface is very simple. It contains a text box where the user can enter input text, a combo box from which the user can select one of the available voices per language, and a button that starts the process. Code Listing 16 demonstrates this.

*Code Listing 16*

```
<Window x:Class="AzureTextToSpeech.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Azure Text to Speech" Height="250" Width="400">
    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBox Name="TxtInput" Width="360" Height="100"
                 TextWrapping="Wrap" AcceptsReturn="True"/>
        <ComboBox Name="CmbVoices" Width="360" Height="30"
                  Margin="0,10,0,0" Grid.Row="1"/>
        <Button Name="BtnSpeak" Content="Convert to Speech"
                Width="150" Height="30" Margin="0,10,0,0"
                Grid.Row="2" Click="BtnSpeak_Click"/>
    </Grid>
</Window>
```

The next step is adding the logic that enables the user interface.

## Adding text-to-speech capabilities in C#

In the MainPage.xaml.cs, add the code shown in Code Listing 17. As usual, comments will follow shortly. Also, you will notice some objects that were already discussed in the previous example.

*Code Listing 17*

```
using Microsoft.CognitiveServices.Speech;
using System.Windows;

namespace AzureTextToSpeech
{
    public partial class MainWindow : Window
    {
        private string subscriptionKey =
            "your-api-key";
        private string region =
            "your-region";

        public MainWindow()
        {
            InitializeComponent();
            LoadVoicesAsync();
```

```
        }

        private async void LoadVoicesAsync()
        {
            var config = SpeechConfig.
                FromSubscription(subscriptionKey, region);
            var synthesizer =
                new SpeechSynthesizer(config);

            var result =
                await synthesizer.GetVoicesAsync();
            CmbVoices.ItemsSource =
                result.Voices.Select(v => v.ShortName).ToList();
            CmbVoices.SelectedIndex = 0;
        }

        private async void BtnSpeak_Click(
            object sender, RoutedEventArgs e)
        {
            if (string.IsNullOrWhiteSpace(TxtInput.Text)) return;

            var config = SpeechConfig.
                FromSubscription(subscriptionKey, region);
            config.SpeechSynthesisVoiceName =
                CmbVoices.SelectedItem.ToString();

            using var synthesizer = new SpeechSynthesizer(config);
            await synthesizer.SpeakTextAsync(TxtInput.Text);
        }
    }
}
```

Following is a description of the relevant types and members in the code:

- The **`SpeechConfig`** class has the same purpose as the previous example. In this case, it is worth mentioning the **`SpeechSynthesisVoiceName`** property, which allows for specifying the voice to use for synthesis. For instance, you might set this to **`en-US-JennyNeural`** for a specific American English voice.

- The **`SpeechSynthesizer`** class handles the conversion of text into spoken audio. It requires a **`SpeechConfig`** object for initialization. The **`SpeakTextAsync`** method is used to take a string of text and synthesize it into speech, which is then played through the configured audio output device. For more control, the **`SpeakSsmlAsync`** method can be used to synthesize speech from SSML (Speech Synthesis Markup Language) input, which allows for finer control over speech characteristics. Additionally, **`StartSpeakingAsync`** begins synthesis and returns immediately, allowing the app to perform other tasks while synthesis continues in the background.
- **`SpeechSynthesisResult`**: This class contains the result of the text-to-speech operation. The **`Reason`** property indicates whether the synthesis was successful (**`SynthesizingAudioCompleted`**) or encountered problems. Possible values are listed in Table 1. The **`AudioData`** property contains the synthesized audio data in raw bytes, which can be saved or further processed.

## Running the application

At this point, you can finally run the application and see the result of the work. When running, enter some text, as shown in Figure 36. Select one of the available voices, depending on the language for your text, and then click **Convert to Speech**. Voice identifiers always start with a language code, so for example, the **`en-US-EmmaNeural`** voice used in Figure 37 relates to the **`en-US`** culture.

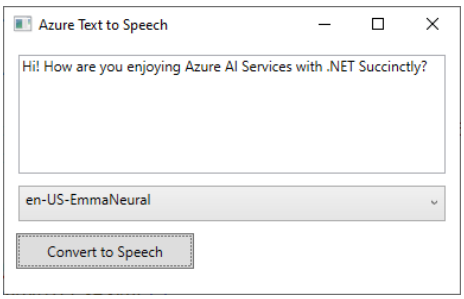



*Figure 37: Converting text to speech*

If everything is okay with the code, you will hear a voice speaking your text with a natural tone.

# Sample application: speech translation

The last example for this chapter demonstrates how to create a WPF application that takes spoken input from the user (via the default microphone) and translates it into a different language using the Azure AI Speech translation service. This example is useful to reproduce the scenario of real-time communication in multilingual settings, such as global conferences or customer support.

With the usual approach, create a new WPF project called **AzureSpeechTranslation** with the following commands to be typed inside a Terminal instance in VS Code:

```
> md \AIServices\AzureSpeechTranslation
```

```
> cd \AIServices\AzureSpeechTranslation
> dotnet new wpf
> dotnet add package Microsoft.CognitiveServices.Speech
```

Open the folder containing the project. Now, you will write a simple user interface and the logic that translates the user input.

## Defining the user interface

The user interface for this example is also very simple. It includes a `ComboBox` control to allow the target language selection, a button that will enable the microphone and send the speech to the Azure AI Speech service, and a text box that displays the translation result. Code Listing 18 demonstrates how to implement this simple user interface in the MainPage.xaml file.

*Code Listing 18*

```
<Window x:Class="AzureSpeechTranslation.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Azure Speech Translation" Height="200" Width="400">
    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition />
        </Grid.RowDefinitions>
        <ComboBox Name="CmbLanguages" Width="360" Height="30"
                  VerticalAlignment="Top" Margin="0,10,0,0"/>
        <Button Name="BtnTranslate" Content="Translate Speech"
                Width="150" Height="30" Grid.Row="1"
                Margin="0,10,0,0" Click="BtnTranslate_Click"/>
        <TextBox Name="TxtTranslation" Width="360" Height="80"
                 Margin="0,10,0,0" Grid.Row="2"
                 TextWrapping="Wrap" AcceptsReturn="True"/>
    </Grid>
</Window>
```

Now you can add the C# logic that enables the controls.

## Adding speech translation capabilities in C#

In the MainPage.xaml.cs file, add the code shown in Code Listing 19. Some objects related to the AI Speech service will now be familiar, but explanations will follow shortly.

*Code Listing 19*

```
using Microsoft.CognitiveServices.Speech;
```

```
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Translation;
using System.Windows;

namespace AzureSpeechTranslation
{
    public partial class MainWindow : Window
    {
        private string subscriptionKey =
            "your-api-key";
        private string region =
            "your-region";

        public MainWindow()
        {
            InitializeComponent();
            LoadLanguages();
        }

        private void LoadLanguages()
        {
            CmbLanguages.ItemsSource =
                new[] { "fr-FR", "de-DE", "es-ES", "zh-CN" };
            CmbLanguages.SelectedIndex = 0;
        }

        private async void BtnTranslate_Click(
            object sender, RoutedEventArgs e)
        {
            try
            {
                var config = SpeechTranslationConfig.
                    FromSubscription(subscriptionKey, region);
                config.SpeechRecognitionLanguage =
                    "en-US";
                config.AddTargetLanguage(CmbLanguages.
                    SelectedItem.ToString());

                // Configure microphone input.
                using var audioInput =
                    AudioConfig.FromDefaultMicrophoneInput();
                using var recognizer =
                    new TranslationRecognizer(config, audioInput);

                var result = await recognizer.RecognizeOnceAsync();

                if (result.Reason == ResultReason.TranslatedSpeech)
                {
                    TxtTranslation.Text =
```

```
                        result.Translations.FirstOrDefault().Value;
                }
                else
                {
                    TxtTranslation.Text =
                        "Translation failed: " + result.Reason.ToString();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error accessing the microphone:
                                {ex.Message}");
            }
        }
    }
}
```

Following is a summary of the relevant objects used in the code:

- The **`AudioConfig`** class allows for interacting with audio devices. In particular, the **`FromDefaultMicrophoneInput`** method captures audio from the default audio input device.
- The **`TranslationRecognizer`** class creates a connection between the application and the Azure AI Speech translation service, and it requires the service configuration and the audio input configuration objects as arguments.
- The **`TranslationRecognizer.RecognizeOnceAsync`** method starts listening to the incoming audio and stops listening when there is no more signal. At this point, it sends the audio to the Azure AI Speech service. Alternative methods are **`StartContinuousRecognitionAsync`** and **`StartKeywordRecognitionAsync`**; the first method allows for continuous voice recognition, whereas the second one recognizes keywords rather than sentences. Recognition with these methods must be then explicitly stopped with the corresponding **`StopContinuousRecognitionAsync`** and **`StopKeywordRecognitionAsync`** methods.
- The result is an object of type **`TranslationRecognitionResult`**. This exposes the **`Result`** property, of type **`ResultReason`**, whose value will be **`TranslatedSpeech`** if the translation was successful. Other possible values are summarized in Table 1. It also exposes the **`Translations`** property, of type **`IReadonlyDictionary<string, string>`**, whose first item contains the translation result that is displayed in the user interface.

You can provide different messages and user interface behaviors, depending on the value returned by the **`Reason`** property. Now that you have an idea of the types and members, you can run the application.

## Running the application

When you run the application (F5 or the **`dotnet run`** command from the Terminal), you will be able to select a language and click the button to start voice recognition. Figure 38 shows an example of speech translation using Azure AI Speech, where English speech has been translated to French text.

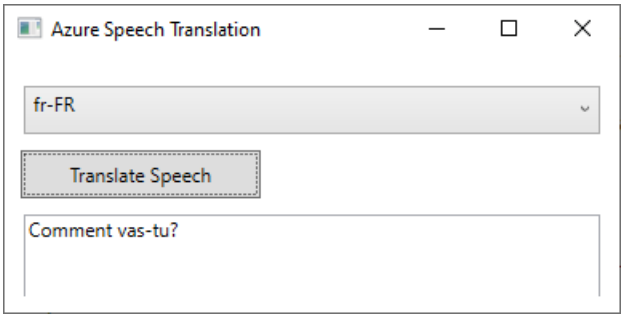



*Figure 38: Demonstrating speech translation*

As you can imagine, the opportunities offered by this service are enormous. You can also combine the three speech features together with Azure AI Translator to create powerful, language-oriented applications.

# Errors and exceptions

If the requested service fails, the Azure AI Speech service can throw the following exceptions:

- **`AudioFormatException`**: Occurs when the provided audio file is in an unsupported format or the data is corrupted.
- **`SpeechRecognitionException`**: Happens when the speech-to-text engine cannot properly process the audio, often due to low quality or unrecognizable speech patterns.

Do not forget to implement a **`try..catch`** block as a best practice for exception handling.

# Chapter summary

The Microsoft Azure AI Speech service offers robust and flexible speech capabilities that can be seamlessly integrated into .NET applications. You have seen how to use the speech-to-text functionality to convert speech into text; you have seen how to use text-to-speech to convert written text quickly into spoken, natural language; finally, you have seen how fast it is at translating speech into written text from one language to another.

This powerful service is extremely valuable for developers looking to build voice-enabled applications, enhance accessibility, or automate transcription processes. The example provided not only illustrates the ease of integration but also highlights the possible applications of Azure AI Speech in real-world scenarios.

# Chapter 9 Azure AI Computer Vision Services

The AI Computer Vision services represent one of the most relevant technologies in the AI offerings from Microsoft, because they target advanced image and video analysis; therefore, they can be of extremely common application. This chapter describes how to perform image and OCR analysis with AI Computer Vision services, with examples that developers can immediately apply in their scenarios.

## Introducing Azure AI Computer Vision services

Microsoft Azure AI Computer Vision, also referred to as Computer Vision, is a part of Azure's cognitive services that allows developers to integrate advanced computer vision capabilities into their applications. Leveraging state-of-the-art machine learning models, it enables the analysis and understanding of visual data, such as images, videos, and spatial information.

This service provides a range of features including image recognition, facial analysis, optical character recognition (OCR), and spatial analysis. With Azure AI Computer Vision, businesses can automate tasks that require visual data processing, such as extracting text from images, identifying objects, or analyzing movement in physical spaces.

Azure AI Computer Vision is cloud-based, which means that it allows businesses to scale and manage vision-related workloads without the need for on-premises infrastructure. The service supports a wide range of industries, from retail and manufacturing to healthcare, providing solutions that help automate and enhance business operations using AI-powered visual analytics.

Azure AI Computer Vision is a comprehensive service that provides several functionalities aimed at enabling businesses to extract insights from visual data. Some of the primary features include:

- **Image analysis:** The image analysis service provides a set of functionalities to analyze images and extract valuable insights. Using advanced algorithms, it detects objects, people, and scenes; categorizes images into thousands of predefined tags; and provides a summary of the image content in the form of a description. It also supports image moderation to detect explicit or unwanted content. This service can handle various image formats, and it provides analysis results in JSON format, making it easy to integrate with different types of applications.
- **Facial recognition:** The facial recognition service focuses on identifying and analyzing human faces in images or video feeds. It allows developers to detect and compare faces, determine facial attributes such as age and gender, and even recognize emotions. It is commonly used in security and identity verification systems. Additionally, it supports creating face databases to recognize individuals and track their interactions over time. Privacy and security are key concerns, and Azure AI Computer Vision offers encryption and compliance features to meet legal requirements.

- **Spatial analysis:** This service enables developers to derive insights from real-time video feeds. It uses AI models to detect and track movement in predefined spaces, identifying patterns and generating actionable insights. Common use cases include analyzing customer behavior in retail environments, monitoring safety protocols in workplaces, and ensuring social distancing in public areas. Spatial analysis leverages Azure's scalable architecture to handle large amounts of video data and can be integrated with IoT devices for enhanced automation.
- **Optical character recognition (OCR):** Azure's OCR capability transforms images of text into machine-readable formats, supporting multiple languages and a wide range of image formats. OCR is commonly used in scenarios such as document scanning, data extraction from forms, and processing handwritten notes. The service can identify text in various fonts and styles and convert it into structured data that can be processed further or stored for future use.

As you can imagine, intelligent applications built on top of Azure AI Computer Vision can dramatically help companies automate and improve their processes. The aforementioned services are divided into two main groups: image analysis, spatial analysis, and OCR recognition are part of the Computer Vision group. Face recognition is exposed by the Face API group. Both go under the umbrella of the AI Computer Vision services. In the next sections, you will start using AI Computer Vision with specific code examples built on top of WPF with a convenient user interface.

## Exclusions and limitations

The Azure AI Computer Vision services are extremely powerful, and for this reason they are strictly regulated. For example, the Face recognition service can detect biometric information and could help identify people and their physical characteristics. As you can imagine, this can lead to enormous risks if the collected data is obtained, stored, and retrieved without the appropriate permissions or outside scenarios specified by national and international laws.

Based on this, Microsoft decided to open Face API only to customers and organizations that submit a registration form and comply with specific legal requirements. All the necessary information can be found in the Limited Access to Face API page. As a consequence, this chapter will only describe how to set up an application and how to write the necessary code to work with Face API, but no image will be analyzed, and no screenshot will be provided.

***Note: For some Azure regions, Microsoft could allow age estimation and gender detection without the need to submit a registration form.***

Spatial analysis involves analyzing real-time videos via IP cameras and requires the presence of multiple people to provide an appropriate result. Therefore, only a description of the service and of the related .NET objects will be provided.

# Configuring the Azure AI Computer Vision resources

Before diving into code examples, you need to set up the Azure AI Computer Vision service in the Azure Portal. To accomplish this, follow these steps based on what you learned in the previous chapters:

1. Go to the Azure Portal and sign in with your Azure account.
2. Click **Azure AI services**. Figure 39 shows where you can find both the Computer Vision and Face API services.
3. Click **Create** on the **Computer vision** card.
4. On the Create Computer Vision page, choose your Azure subscription and specify the resource group you created previously.
5. Choose the region closest to your location and enter a unique name for your AI Computer Vision resource, for example, ai-vision-succinctly.
6. In the Pricing Tier box, select the **Free** plan.
7. Make sure you acknowledge that you have read the notice about responsible usage of AI. Carefully read the full note and then select the related checkbox.
8. Click **Review + Create** > **Create**.

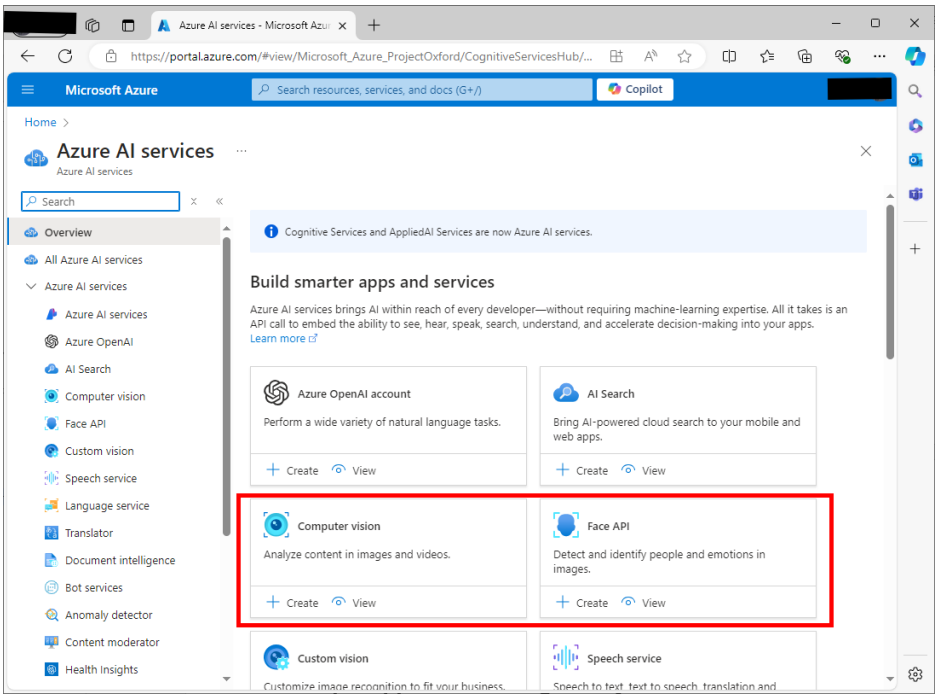



*Figure 39: Locating the Computer Vision and Face API groups*

Once the resource is created, go to the resource's overview page and locate your API key and endpoint. These will be required to interact with the Azure AI Computer Vision service in C# code.

## Sample application: image analysis

The goal of the first sample application is to allow users to upload an image from the local machine and analyze it using the Azure AI Computer Vision's image analysis service. The application will display detected objects, tags, and a description of the image.

Open Visual Studio Code and start a new instance of the Terminal. When ready, type the following sequence of commands:

```
> md \AIServices\ImageAnalysisApp
> cd \AIServices\ImageAnalysisApp
> dotnet new wpf
> dotnet add package Microsoft.Azure.CognitiveServices.Vision.ComputerVision
```

The Microsoft.Azure.CognitiveServices.Vision.ComputerVision NuGet package is required to work against image analysis and OCR recognition features. When you're ready, open the project in VS Code. At this point, you can define a simple user interface.

### Defining the user interface

In the MainPage.xaml file, add the code shown in Code Listing 20.

*Code Listing 20*

```
<Window x:Class="ImageAnalysisApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:ImageAnalysisApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Button Content="Upload Image" HorizontalAlignment="Left"
                VerticalAlignment="Top" Width="120" Margin="10,10,0,0"
                Height="30" Click="UploadImage_Click"/>
        <Image Grid.Row="1" x:Name="UploadedImage"
               HorizontalAlignment="Left" VerticalAlignment="Top"
               Margin="10,10,0,0" Width="300" Height="300"/>
        <TextBlock Grid.Row="2" x:Name="AnalysisResult"
               HorizontalAlignment="Left" VerticalAlignment="Top"
               Margin="10,10,0,0" TextWrapping="Wrap"
```

```
                FontSize="16"/>
    </Grid>
</Window>
```

As you can see, the user interface is very simple: it contains a button that allows for uploading an image file of choice, an **`Image`** control to display the image, and a **`TextBlock`** that displays the analysis result. Now it is time to leverage the image analysis API in C#.

## Image analysis in C#

The purpose of the code is to make the application analyze an image and display a description, tags, and detected objects. For instance, uploading an image of a car might result in the description "A red car on the road," tags such as "car" and "road," and object detection for the car itself. Code Listing 21 demonstrates how to accomplish this, with code that you need to write into the MainPage.xaml.cs file.

*Code Listing 21*

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;

namespace ImageAnalysisApp
{
    public partial class MainWindow : Window
    {
        private readonly string subscriptionKey
            = "your-api-key";
        private readonly string endpoint
            = "your-endpoint";
        private readonly ComputerVisionClient client;

        public MainWindow()
        {
            InitializeComponent();
            client = new ComputerVisionClient(
                new ApiKeyServiceClientCredentials(subscriptionKey))
            {
                Endpoint = endpoint
            };
        }

        private async void UploadImage_Click(object sender,
```

```
                RoutedEventArgs e)
        {
            var openFileDialog = new Microsoft.Win32.OpenFileDialog();
            openFileDialog.Filter =
                "Image files (*.jpg, *.jpeg, *.png)|*.jpg;*.jpeg;*.png";

            if (openFileDialog.ShowDialog() == true)
            {
                string filePath = openFileDialog.FileName;
                UploadedImage.Source = new BitmapImage(new Uri(filePath));
                using (var imageStream = File.OpenRead(filePath))
                {
                    var features = new List<VisualFeatureTypes?>
                    {
                        VisualFeatureTypes.Tags,
                        VisualFeatureTypes.Description,
                        VisualFeatureTypes.Objects
                    };
                    var analysis = await client.
                        AnalyzeImageInStreamAsync(imageStream, features);

                    string result = $"Description: " +
                        $"{analysis.Description.Captions[0].Text}\n";
                    result += "Tags: " + string.Join(", ",
                        analysis.Tags.Select(tag => tag.Name)) + "\n";
                    result += "Objects:\n";

                    foreach (var obj in analysis.Objects)
                    {
                        result += $"- {obj.ObjectProperty} at " +
                            $"{obj.Rectangle.X},{obj.Rectangle.Y}\n";
                    }

                    AnalysisResult.Text = result;
                }
            }
        }
    }
}
```

Following is a list of the relevant types and members used in the code:

- The **`ComputerVisionClient`** class provides access to the Azure Computer Vision API. It facilitates the analysis of images by interacting with the cloud service. In the sample, the **`AnalyzeImageInStreamAsync`** method is used to submit an image for analysis, but the class also supports other methods for different scenarios. These include **`AnalyzeImageAsync`** for image URLs, and **`ReadAsync`** for text extraction. The **`ComputerVisionClient`** class requires an instance of **`ApiKeyServiceClientCredentials`** to authenticate with the Azure service.

- The **`VisualFeatureTypes`** enumeration defines the types of visual features that can be extracted from an image. In the sample, **`Objects`** and **`Tags`** are requested, but other values are available for different use cases. For instance, **`Categories`** returns a hierarchical list of objects, while **`Description`** generates human-readable descriptions of the image content. The enumeration also includes values like **`Faces`**, **`ImageType`**, **`Color`**, and **`Adult`**, which provide information on the people in the image, the image's type, dominant colors, and the presence of adult content, respectively.
- The **`DetectedObject`** class represents objects detected in an image. It contains properties such as **`ObjectProperty`**, which holds the name of the detected object, and **`Confidence`**, which indicates the model's confidence level in the detection. The **`DetectedObject`** class also includes **`Rectangle`**, a property of type **`BoundingRect`**, which provides the coordinates of the bounding box around the detected object. This enables drawing or highlighting the detected objects in a user interface. The sample demonstrates the use of these properties to retrieve and display the name and bounding box of each detected object.
- The **`ImageTag`** class represents the tags generated from the image analysis. Each tag is associated with a specific object or concept identified in the image. The **`ImageTag`** class contains properties like **`Name`**, which represents the tag's name, and **`Confidence`**, which indicates the model's confidence level in the accuracy of the tag. In the sample, tags are extracted and listed along with their confidence values. Tags can be used for metadata generation, content categorization, or search indexing in various applications.

Now that you have more details about the relevant objects, it's time to run the application.

## Running the application

To run the application, you can either press F5 for debugging mode or type **`dotnet run`** in the Terminal. When the application starts, click **Upload Image** and select an image you want to analyze. The companion solution includes an image, whose analysis results are as shown in Figure 40.

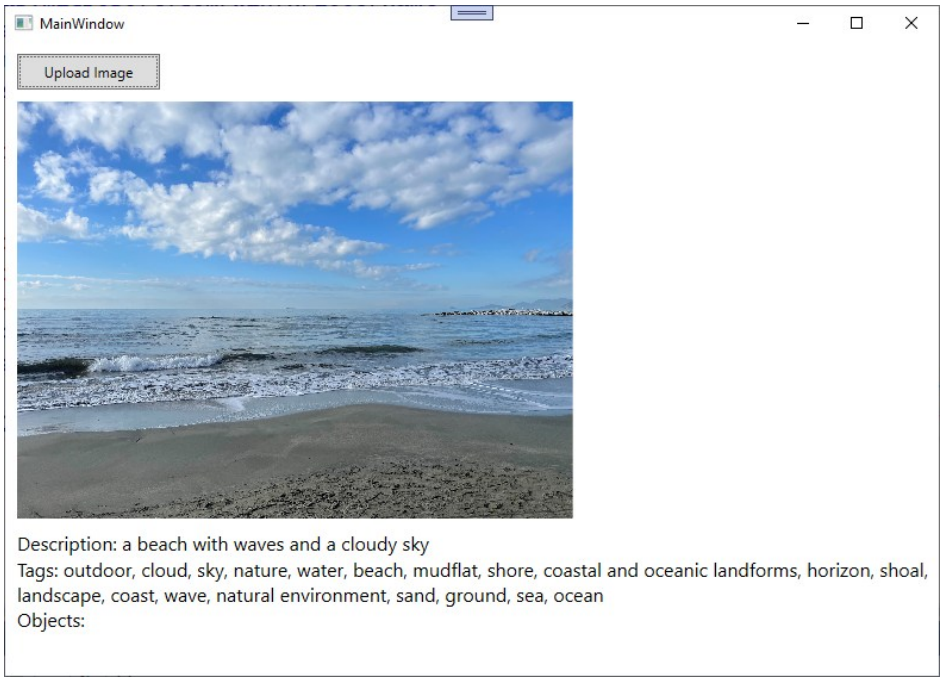



*Figure 40: Image analysis result*

As you can see, the Computer Vision service has returned a description in natural language, also providing tags. No objects are detected in the picture, so these are not described. You can easily imagine the potential of this feature, for example in helping visually impaired people.

## Sample application: face detection

***Note: Based on the limitations described at the beginning of this chapter, only the source code and explanation will be provided.***

The Face API can not only detect faces inside an image or video, but also their attributes, including biometric characteristics. The purpose of the sample code is to detect faces in a locally stored image and analyze facial attributes such as age and emotions.

Follow the steps you already know to create a new WPF application called FaceAnalysisApp in Visual Studio Code. You also need to install the Microsoft.Azure.CognitiveServices.Vision.Face NuGet package, currently in preview, with the following command line:

```
> dotnet add package Microsoft.Azure.CognitiveServices.Vision.Face
```

At this point, you can configure the Azure resources.

## Configuring the Azure resources

Face recognition relies on a dedicated Azure service called Face API, so you cannot reuse the resources created previously. In the Azure Portal, click **AI Services**. Locate the Face API service (see Figure 38) and click **Create**. Follow the same steps described for image analysis to set up the new service, which can be called face-succinctly-service. Review and create the Face API service and, when deployed, follow the steps you know to retrieve the endpoint and API key.

## Defining the user interface

For the user interface, you can simply reuse the XAML shown in Code Listing 4, which allows you to upload an image and display the analysis results.

## Face detection in C#

The sample application will detect faces in an uploaded image and display attributes such as estimated age and dominant emotions. For example, uploading an image of a person smiling may result in detecting a face with the emotion "Happiness." To accomplish this, write the code shown in Code Listing 22 in the code-behind file for the main page.

*Code Listing 22*

```
using System;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Media.Imaging;
using Microsoft.Azure.CognitiveServices.Vision.Face;
using Microsoft.Azure.CognitiveServices.Vision.Face.Models;

namespace FaceAnalysisApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private readonly string subscriptionKey = "your-key-here";
        private readonly string endpoint = "your-endpoint-here";
        private readonly FaceClient faceClient;

        public MainWindow()
        {
            InitializeComponent();
            faceClient = new FaceClient(
                new ApiKeyServiceClientCredentials(subscriptionKey))
```

```
            {
                Endpoint = endpoint
            };
        }

        private async void UploadImage_Click(object sender,
            RoutedEventArgs e)
        {
            var openFileDialog = new Microsoft.Win32.OpenFileDialog();
            openFileDialog.Filter =
                "Image files (*.jpg, *.jpeg, *.png)|*.jpg;*.jpeg;*.png";

            if (openFileDialog.ShowDialog() == true)
            {
                string filePath = openFileDialog.FileName;
                UploadedImage.Source = new BitmapImage(new Uri(filePath));
                using (var imageStream = File.OpenRead(filePath))
                {
                    var faceAttributes = new FaceAttributeType[]
                    {
                        FaceAttributeType.Age,
                        FaceAttributeType.Emotion
                    };
                    var faces = await faceClient.
                        Face.DetectWithStreamAsync(imageStream,
                        returnFaceAttributes: faceAttributes);

                    string result = "Detected faces:\n";

                    foreach (var face in faces)
                    {
                        result += $"- Face detected with age:
                            {face.FaceAttributes.Age}, " +
                            $"Emotion: {face.FaceAttributes.Emotion.
                            ToRankedList().
                            FirstOrDefault().Key}\n";
                    }

                    AnalysisResult.Text = result;
                }
            }
        }
    }
}
```

Following is a list of relevant types and members used to perform face recognition:

- **`FaceClient`**: The main client class to interact with Azure's facial recognition service.
- **`FaceAttributeType`**: Specifies the facial attributes to detect, such as age and emotions.

- **`DetectedFace`**: Represents a detected face, containing information about the face's attributes.
- **`DetectWithStreamAsync`**: A method that executes the actual analysis over a stream. As an alternative, you can use **`DetectWithUrlAsync`** if the image can be reached via URL.
- **`FaceAttributes`**: A collection of detected face attributes. Each attribute is represented by a property; for example, **`Age`** represents the estimated age, and **`Emotion`** represents the emotional attributes detected on the face. Properties like **`Age`** and **`Emotion`** contain collections of values, ordered by rank.
- **`ToRankedList`**: A method that returns an **`IEnumerable<KeyValuePair<string, double>>`**. Values in this collection are ordered by rank, from the highest match to the lowest.

As mentioned previously, you will be able to run face detection only if you submit a registration form to Microsoft and your application is approved.

# Sample application: OCR

The goal of the third sample application is to show the optical character recognition (OCR) features provided by the Azure AI Computer Vision service. This will be demonstrated by retrieving text detected on a JPEG image. For the sake of simplicity, an image file with text is included with the companion source code and can be used for the next example. The good news is that you do not need to configure a new Azure resource, because OCR recognition is part of the image analysis API that you already configured and used for the first sample application. Having said this, open the Terminal in Visual Studio Code and create a new WPF app called **OCRApp** with the following commands:

```
> md \AIServices\OCRApp
> cd \AIServices\OCRApp
> dotnet new wpf
> dotnet add package Microsoft.Azure.CognitiveServices.Vision.ComputerVision
```

When you're done, open the project folder in Visual Studio Code and open the MainPage.xaml file.

## Defining the user interface

For the user interface, you can safely reuse the same XAML code shown in Code Listing 4. In fact, you need the same visual elements: a button to load the image file, an **`Image`** control to display the image, and a **`TextBlock`** that shows the analysis result. At this point, you can implement the C# code that performs the image analysis.

## OCR in C#

The application will extract text from the uploaded image and display it in the text area. For example, uploading an image of a receipt will result in the display of its printed text. To accomplish this, open the MainPage.xaml.cs file and add the code shown in Code Listing 23 (an explanation is coming shortly). Notice that you use the same API key and service endpoint of the first sample project.

*Code Listing 23*

```
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using System.IO;
using System.Windows;
using System.Windows.Media.Imaging;

namespace OCRApp
{
    public partial class MainWindow : Window
    {
        private readonly string
            subscriptionKey = "your-api-key";
        private readonly string
            endpoint = "your-endpoint";
        private readonly ComputerVisionClient client;

        public MainWindow()
        {
            InitializeComponent();
            client = new ComputerVisionClient(
                new ApiKeyServiceClientCredentials(subscriptionKey))
            {
                Endpoint = endpoint
            };
        }

        private async void UploadImage_Click(object sender,
                            RoutedEventArgs e)
        {
            var openFileDialog = new Microsoft.Win32.OpenFileDialog();
            openFileDialog.Filter =
                "Image files (*.jpg, *.jpeg, *.png)|*.jpg;*.jpeg;*.png";

            if (openFileDialog.ShowDialog() == true)
            {
                string filePath = openFileDialog.FileName;
                UploadedImage.Source =
                    new BitmapImage(new Uri(filePath));
                using (var imageStream = File.OpenRead(filePath))
                {
```

```
                    var ocrResult = await client.
                        RecognizePrintedTextInStreamAsync(true,
                        imageStream);

                    string result = "Extracted text:\n";

                    foreach (var region in ocrResult.Regions)
                    {
                        foreach (var line in region.Lines)
                        {
                            result += string.Join(" ",
                                line.Words.Select(word => word.Text)) +
                                "\n";
                        }
                    }

                    AnalysisResult.Text = result;
                }
            }
        }
    }
}
```

Following is a description of the relevant types and members used for OCR:

- The **`ComputerVisionClient`** class is the main entry point for accessing the Azure Cognitive Services Computer Vision API. In the example, the **`RecognizePrintedTextInStreamAsync`** method is used to perform OCR on an image stream. This method sends the image to the Azure service, where it recognizes printed text. The client requires authentication via the **`ApiKeyServiceClientCredentials`** class, which is initialized with the subscription key provided by Azure. Additionally, the **`Endpoint`** property must be set to the service's endpoint URL to establish the connection.
- The **`RecognizePrintedTextInStreamAsync`** method takes a Boolean parameter to specify whether to detect the text orientation and a stream containing the image data. The result of this method is an **`OcrResult`** object, which contains the recognized text broken down into regions, lines, and words. Alternative methods such as **`RecognizePrintedTextAsync`** can be used to perform OCR on images from a URL rather than a stream.
- The **`OcrResult`** class encapsulates the result of the OCR operation. It contains the **`Regions`** property, which is a collection of **`OcrRegion`** objects. Each **`OcrRegion`** represents a detected block of text within the image, typically corresponding to paragraphs or larger text areas. The **`OcrResult`** also includes the **`Language`** property, indicating the detected language of the text in the image. The **`Orientation`** property specifies the text orientation, such as up, down, or sideways.

- The **`OcrRegion`** class represents a region of detected text in the image. It contains the **`Lines`** property, which is a collection of **`OcrLine`** objects. Each **`OcrRegion`** corresponds to a larger block of text, like a paragraph, and within that region, text is further broken down into lines. The **`BoundingBox`** property provides the coordinates of the region, allowing for spatial analysis of where the text is located within the image.
- The **`OcrLine`** class represents a line of text within an **`OcrRegion`**. It contains the **`Words`** property, which is a collection of **`OcrWord`** objects. The **`OcrLine`** object also includes a **`BoundingBox`** property, which gives the coordinates of the line within the image. The sample code iterates through the **`Lines`** property of each **`OcrRegion`** to extract and display the recognized text.
- The **`OcrWord`** class represents an individual word recognized in a line of text. The **`Text`** property contains the actual word recognized by the OCR service. Like **`OcrRegion`** and **`OcrLine`**, the **`OcrWord`** object has a **`BoundingBox`** property that specifies the word's position within the image. The sample code demonstrates how to concatenate the recognized words in a line to form the complete text output.

The **`RecognizePrintedTextInStreamAsync`** method uses the **`TextRecognitionMode`** enumeration to distinguish between printed text and handwritten text. The enumeration has two values: **`Printed`**, which is used when recognizing printed text, and **`Handwritten`**, which is used when recognizing handwritten text, allowing for the identification of cursive or printed handwriting. Now that you have knowledge of the SDK types for OCR, it is time to start the application.

## Running the application

You can now run the application, pressing F5 to start it in debugging mode or typing **`dotnet run`** in the Terminal. When you're ready, click **Upload Image**. Select an image file that contains text, such as the sample image included with the companion solution. Figure 41 shows the result of the OCR processing on the companion image file.

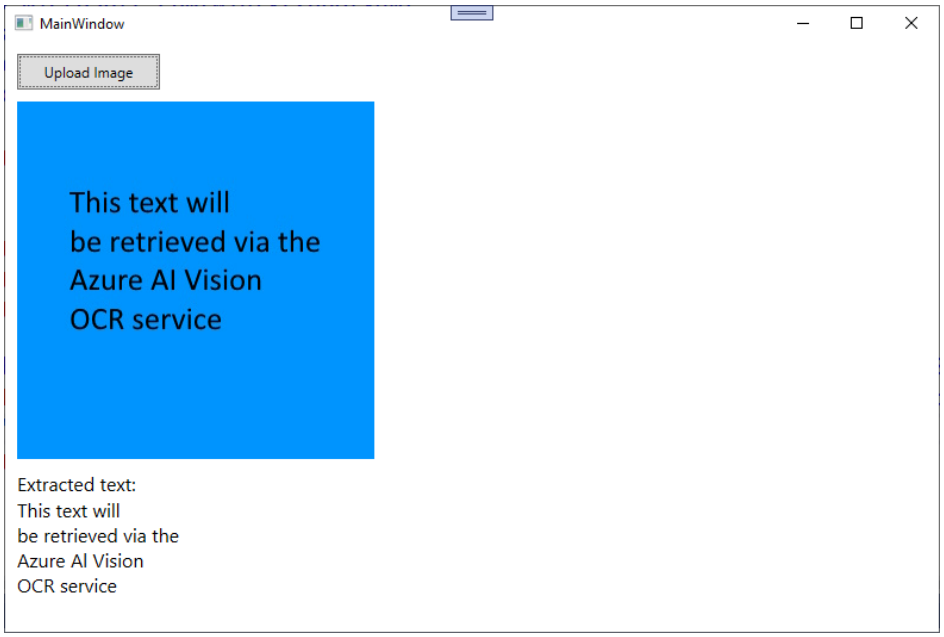



*Figure 41: Result of the AI-powered OCR*

This is another extremely powerful feature: imagine the combination of OCR with translation services, and how this could help customers worldwide. Also, remember that OCR works with any image that contains text, not just plain text saved as image files.

## Hints about Spatial Analysis

Azure Spatial Analysis, part of the Azure AI Computer Vision suite, is designed to analyze physical spaces by tracking objects (like people) in a given environment using live video feeds from connected cameras. The service provides capabilities such as monitoring room occupancy, social distancing, and tracking the movement of people in real-time. Spatial analysis can be highly valuable for businesses, security systems, and smart building management.

Azure Spatial Analysis relies on the Azure Cognitive Services Computer Vision SDK, which handles the interaction with the service. The main types and methods involved in setting up a spatial analysis application are summarized in Table 4.

*Table 4: Relevant types used for spatial analysis*

| .NET Type | Description |
|---|---|
| **`ComputerVisionClient`** | This class is the main entry point for interacting with the Computer Vision service, including spatial analysis features. It provides methods for sending video frames and receiving analyzed data. |

| .NET Type | Description |
| --- | --- |
| `AnalyzeImageInStreamAsync` | Used for analyzing images or video frames streamed to the API. While this is more commonly used for image analysis, spatial analysis often involves sending frames from a video stream for continuous evaluation. |
| `SpatialAnalysisConfiguration` | Represents the configuration for a spatial analysis operation. It includes information such as the camera setup, spatial zones, and rules for triggering events (e.g., when a specific number of people enter a designated area). |
| `SpatialOperationResult` | The result object returned after performing a spatial analysis operation. It contains the analyzed data from the video feed, including information like detected people, their positions, and any specific rule violations. |
| `PeopleCount` | The number of people detected within the analyzed area. |
| `OccupancyStatus` | Information about the occupancy in specific zones. |
| `PersonDetails` | Detailed information about each detected person, such as their location, movement patterns, and proximity to others. |
| `Zone` | Defines a specific area within the spatial environment that you want to monitor. A zone can represent rooms, aisles, or specific boundaries where you want the analysis to occur. |
| `Person` | Represents an individual detected in the video feed. This object includes data like the person's location, movement, and interaction with other people or objects within the defined zones. |
| `ProximityRule` | A configuration object that defines rules for proximity analysis, such as checking whether two or more people are standing too close to each other. This is particularly useful for enforcing social distancing rules in a physical space. |

Further information and code examples about accessing IP (Internet Protocol) cameras can be found in the official documentation page.

## Errors and exceptions

If any of the requested services fails, the Azure AI Vision service can throw the following exceptions:

- **`ImageFormatException`**: Thrown when the provided image for analysis is in an unsupported format (e.g., TIFF when only JPEG or PNG is allowed).
- **`OCRLanguageNotSupportedException`**: Thrown when the OCR service does not support the language in the image.

Do not forget to implement a **`try..catch`** block as a best practice for exception handling.

## Chapter summary

Microsoft Azure AI Computer Vision provides powerful tools for integrating AI-driven image and video analysis into applications. From recognizing objects and faces to extracting text from images and analyzing spatial movement, these services can enhance a wide range of business applications.

The provided code examples demonstrate real-world use cases such as image tagging, face detection, and OCR, offering a practical starting point for developers seeking to leverage Azure AI Computer Vision in their applications. By integrating these services, businesses can improve automation, security, and data extraction processes, ultimately leading to more intelligent and efficient operations.

# Conclusion

Microsoft Azure AI services present a robust and comprehensive suite of tools designed to enable businesses to harness the power of artificial intelligence with scalability and flexibility. From natural language processing and computer vision to advanced machine learning models, Azure's AI services provide significant opportunities for organizations to automate tasks, gain deeper insights from data, and enhance customer experiences. The platform's seamless integration with other Microsoft services, along with its commitment to security and regulatory compliance, positions it as a key player in the AI landscape. Building intelligent apps on top of .NET with Visual Studio Code makes development easier and available to all the major systems on the market.

This ebook highlighted Azure's AI services features through prebuilt models and customizable solutions. It's suitable for both AI beginners and seasoned developers. However, while Azure AI services offer substantial benefits, it is crucial for organizations to align their AI strategies with their business goals, considering factors such as cost, infrastructure, and the potential ethical implications of deploying AI solutions. Overall, Azure's AI capabilities are well-suited for companies seeking to innovate and stay competitive in a rapidly evolving digital world.