



Wrox Programmer to Programmer™



Professional

C# and .NET

2021 Edition

Christian Nagel

Table of Contents

[COVER](#)

[TITLE PAGE](#)

[INTRODUCTION](#)

[THE WORLD OF .NET](#)

[THE WORLD OF C#](#)

[WHAT'S NEW IN C#](#)

[WHAT'S NEW IN ASP.NET CORE](#)

[WHAT'S NEW WITH WINDOWS](#)

[WHAT YOU NEED TO WRITE AND RUN C# CODE](#)

[WHAT THIS BOOK COVERS](#)

[CONVENTIONS](#)

[SOURCE CODE](#)

[ERRATA](#)

[PART I: The C# Language](#)

[1 .NET Applications and Tools](#)

[FROM .NET FRAMEWORK TO .NET CORE TO .NET](#)

[.NET TERMS](#)

[.NET SUPPORT LENGTH](#)

[APPLICATION TYPES AND TECHNOLOGIES](#)

[DEVELOPER TOOLS](#)

[USING THE .NET CLI](#)

[SUMMARY](#)

[2 Core C#](#)

[FUNDAMENTALS OF C#](#)

[NULLABLE TYPES](#)

[USING PREDEFINED TYPES](#)
[CONTROLLING PROGRAM FLOW](#)
[ORGANIZATION WITH NAMESPACES](#)
[WORKING WITH STRINGS](#)
[COMMENTS](#)
[C# PREPROCESSOR DIRECTIVES](#)
[C# PROGRAMMING GUIDELINES](#)
[SUMMARY](#)

[3 Classes, Records, Structs, and Tuples](#)

[CREATING AND USING TYPES](#)
[PASS BY VALUE OR BY REFERENCE](#)
[CLASSES](#)
[RECORDS](#)
[STRUCTS](#)
[ENUM TYPES](#)
[REF, IN, AND OUT](#)
[TUPLES](#)
[VALUETUPLE](#)
[DECONSTRUCTION](#)
[PATTERN MATCHING](#)
[PARTIAL TYPES](#)
[SUMMARY](#)

[4 Object-Oriented Programming in C#](#)

[OBJECT ORIENTATION](#)
[INHERITANCE WITH CLASSES](#)
[MODIFIERS](#)
[INHERITANCE WITH RECORDS](#)
[USING INTERFACES](#)

GENERIC

SUMMARY

5 Operators and Casts

OPERATORS

USING BINARY OPERATORS

TYPE SAFETY

OPERATOR OVERLOADING

COMPARING OBJECTS FOR EQUALITY

IMPLEMENTING CUSTOM INDEXERS

USER-DEFINED CONVERSIONS

SUMMARY

6 Arrays

MULTIPLE OBJECTS OF THE SAME TYPE

SIMPLE ARRAYS

MULTIDIMENSIONAL ARRAYS

JAGGED ARRAYS

ARRAY CLASS

ARRAYS AS PARAMETERS

ENUMERATORS

USING SPAN WITH ARRAYS

INDICES AND RANGES

ARRAY POOLS

BITARRAY

SUMMARY

7 Delegates, Lambdas, and Events

REFERENCING METHODS

DELEGATES

LAMBDA EXPRESSIONS

[EVENTS](#)

[SUMMARY](#)

[8 Collections](#)

[OVERVIEW](#)

[COLLECTION INTERFACES AND TYPES](#)

[LISTS](#)

[STACKS](#)

[LINKED LISTS](#)

[SORTED LIST](#)

[DICTIONARIES](#)

[SETS](#)

[PERFORMANCE](#)

[IMMUTABLE COLLECTIONS](#)

[SUMMARY](#)

[9 Language Integrated Query](#)

[LINQ OVERVIEW](#)

[STANDARD QUERY OPERATORS](#)

[PARALLEL LINQ](#)

[EXPRESSION TREES](#)

[LINQ PROVIDERS](#)

[SUMMARY](#)

[10 Errors and Exceptions](#)

[HANDLING ERRORS](#)

[PREDEFINED EXCEPTION CLASSES](#)

[CATCHING EXCEPTIONS](#)

[USER-DEFINED EXCEPTION CLASSES](#)

[CALLER INFORMATION](#)

[SUMMARY](#)

11 Tasks and Asynchronous Programming

WHY ASYNCHRONOUS PROGRAMMING IS IMPORTANT

TASK-BASED ASYNC PATTERN

TASKS

ERROR HANDLING

CANCELLATION OF ASYNC METHODS

ASYNC STREAMS

ASYNC WITH WINDOWS APPS

SUMMARY

12 Reflection, Metadata, and Source Generators

INSPECTING CODE AT RUNTIME AND DYNAMIC PROGRAMMING

CUSTOM ATTRIBUTES

USING REFLECTION

USING DYNAMIC LANGUAGE EXTENSIONS FOR REFLECTION

EXPANDOOBJECT

SOURCE GENERATORS

SUMMARY

13 Managed and Unmanaged Memory

MEMORY

MEMORY MANAGEMENT UNDER THE HOOD

STRONG AND WEAK REFERENCES

WORKING WITH UNMANAGED RESOURCES

UNSAFE CODE

SPAN<T>

PLATFORM INVOKE

SUMMARY

PART II: Libraries

14 Libraries, Assemblies, Packages, and NuGet

THE HELL OF LIBRARIES

ASSEMBLIES

CREATING AND USING LIBRARIES

CREATING NUGET PACKAGES

MODULE INITIALIZERS

SUMMARY

15 Dependency Injection and Configuration

WHAT IS DEPENDENCY INJECTION?

USING THE .NET DI CONTAINER

USING THE HOST CLASS

LIFETIME OF SERVICES

INITIALIZATION OF SERVICES USING
OPTIONS

USING CONFIGURATION FILES

CONFIGURATION WITH .NET APPLICATIONS

AZURE APP CONFIGURATION

SUMMARY

16 Diagnostics and Metrics

DIAGNOSTICS OVERVIEW

LOGGING

METRICS

ANALYTICS WITH VISUAL STUDIO APP
CENTER

APPLICATION INSIGHTS

SUMMARY

17 Parallel Programming

OVERVIEW

[PARALLEL CLASS](#)
[TASKS](#)
[CANCELLATION FRAMEWORK](#)
[CHANNELS](#)
[TIMERS](#)
[THREADING ISSUES](#)
[INTERLOCKED](#)
[MONITOR](#)
[SPINLOCK](#)
[WAITHANDLE](#)
[MUTEX](#)
[SEMAPHORE](#)
[EVENTS](#)
[BARRIER](#)
[READERWRITERLOCKSLIM](#)
[LOCKS WITH AWAIT](#)
[SUMMARY](#)

[18 Files and Streams](#)

[OVERVIEW](#)
[MANAGING THE FILE SYSTEM](#)
[ITERATING FILES](#)
[WORKING WITH STREAMS](#)
[USING READERS AND WRITERS](#)
[COMPRESSING FILES](#)
[WATCHING FILE CHANGES](#)
[JSON SERIALIZATION](#)
[USING FILES AND STREAMS WITH THE](#)
[WINDOWS RUNTIME](#)

SUMMARY

19 Networking

OVERVIEW

WORKING WITH UTILITY CLASSES

USING SOCKETS

USING TCP CLASSES

USING UDP

USING WEB SERVERS

THE HTTPCLIENT CLASS

HTTPCLIENT FACTORY

SUMMARY

20 Security

ELEMENTS OF SECURITY

VERIFYING USER INFORMATION

ENCRYPTING DATA

ENSURING WEB SECURITY

SUMMARY

21 Entity Framework Core

INTRODUCING EF CORE

CREATING A MODEL

SCAFFOLDING A MODEL FROM THE
DATABASE

MIGRATIONS

WORKING WITH QUERIES

LOADING RELATED DATA

WORKING WITH RELATIONSHIPS

SAVING DATA

CONFLICT HANDLING

[USING TRANSACTIONS](#)
[USING AZURE COSMOS DB](#)
[SUMMARY](#)

[22 Localization](#)

[GLOBAL MARKETS](#)
[NAMESPACE SYSTEM.GLOBALIZATION](#)
[RESOURCES](#)
[LOCALIZATION WITH ASP.NET CORE](#)
[LOCALIZATION WITH WINUI](#)
[SUMMARY](#)

[23 Tests](#)

[OVERVIEW](#)
[UNIT TESTING](#)
[USING A MOCKING LIBRARY](#)
[ASP.NET CORE INTEGRATION TESTING](#)
[SUMMARY](#)

[PART III: Web Applications and Services](#)

[24 ASP.NET Core](#)

[UNDERSTANDING WEB TECHNOLOGIES](#)
[CREATING AN ASP.NET CORE WEB PROJECT](#)
[ADDING CLIENT-SIDE CONTENT](#)
[CREATING CUSTOM MIDDLEWARE](#)
[ENDPOINT ROUTING](#)
[REQUEST AND RESPONSE](#)
[SESSION STATE](#)
[HEALTH CHECKS](#)
[DEPLOYMENT](#)
[SUMMARY](#)

25 Services

UNDERSTANDING TODAY'S SERVICES

REST SERVICES WITH ASP.NET CORE

CREATING A .NET CLIENT

USING EF CORE WITH SERVICES

AUTHENTICATION AND AUTHORIZATION
WITH AZURE AD B2C

IMPLEMENTING AND USING SERVICES WITH
GRPC

USING AZURE FUNCTIONS

MORE AZURE SERVICES

SUMMARY

26 Razor Pages and MVC

SETTING UP SERVICES FOR RAZOR PAGES
AND MVC

RAZOR PAGES

ASP.NET CORE MVC

SUMMARY

27 Blazor

BLAZOR SERVER AND BLAZOR
WEBASSEMBLY

CREATING A BLAZOR SERVER WEB
APPLICATION

BLAZOR WEBASSEMBLY

RAZOR COMPONENTS

SUMMARY

28 SignalR

OVERVIEW

CREATING A SIMPLE CHAT USING SIGNALR

[GROUPING CONNECTIONS](#)
[STREAMING WITH SIGNALR](#)
[SUMMARY](#)

[PART IV: Apps](#)

[29 Windows Apps](#)

[INTRODUCING WINDOWS APPS](#)
[INTRODUCING XAML](#)
[WORKING WITH CONTROLS](#)
[WORKING WITH DATA BINDING](#)
[IMPLEMENTING NAVIGATION](#)
[IMPLEMENTING LAYOUT PANELS](#)
[SUMMARY](#)

[30 Patterns with XAML Apps](#)

[WHY MVVM?](#)
[DEFINING THE MVVM PATTERN](#)
[SAMPLE SOLUTION](#)
[MODELS](#)
[SERVICES](#)
[VIEW MODELS](#)
[VIEWS](#)
[MESSAGING USING EVENTS](#)
[SUMMARY](#)

[31 Styling Windows Apps](#)

[STYLING](#)
[SHAPES](#)
[GEOMETRY](#)
[TRANSFORMATION](#)
[BRUSHES](#)

[STYLES AND RESOURCES](#)

[TEMPLATES](#)

[ANIMATIONS](#)

[VISUAL STATE MANAGER](#)

[SUMMARY](#)

[INDEX](#)

[COPYRIGHT](#)

[DEDICATION](#)

[ABOUT THE AUTHOR](#)

[ABOUT THE TECHNICAL EDITOR](#)

[ACKNOWLEDGMENTS](#)

[END USER LICENSE AGREEMENT](#)

List of Illustrations

Chapter 1

[FIGURE 1-1](#)

[FIGURE 1-2](#)

Chapter 5

[FIGURE 5-1](#)

Chapter 6

[FIGURE 6-1](#)

[FIGURE 6-2](#)

[FIGURE 6-3](#)

[FIGURE 6-4](#)

[FIGURE 6-5](#)

[FIGURE 6-6](#)

[FIGURE 6-7](#)

Chapter 8

[FIGURE 8-1](#)

[FIGURE 8-2](#)

[FIGURE 8-3](#)

[FIGURE 8-4](#)

Chapter 9

[FIGURE 9-1](#)

[FIGURE 9-2](#)

Chapter 13

[FIGURE 13-1](#)

[FIGURE 13-2](#)

[FIGURE 13-3](#)

[FIGURE 13-4](#)

[FIGURE 13-5](#)

[FIGURE 13-6](#)

Chapter 14

[FIGURE 14-1](#)

[FIGURE 14-2](#)

[FIGURE 14-3](#)

Chapter 16

[FIGURE 16-1](#)

[FIGURE 16-2](#)

[FIGURE 16-3](#)

[FIGURE 16-4](#)

[FIGURE 16-5](#)

Chapter 17

[FIGURE 17-1](#)

Chapter 18

[FIGURE 18-1](#)

Chapter 19

[FIGURE 19-1](#)

[FIGURE 19-2](#)

Chapter 20

[FIGURE 20-1](#)

Chapter 22

[FIGURE 22-1](#)

[FIGURE 22-2](#)

[FIGURE 22-3](#)

[FIGURE 22-4](#)

[FIGURE 22-5](#)

[FIGURE 22-6](#)

[FIGURE 22-7](#)

Chapter 23

[FIGURE 23-1](#)

Chapter 24

[FIGURE 24-1](#)

[FIGURE 24-2](#)

Chapter 25

[FIGURE 25-1](#)

[FIGURE 25-2](#)

[FIGURE 25-3](#)

[FIGURE 25-4](#)

[FIGURE 25-5](#)

[FIGURE 25-6](#)

Chapter 26

[FIGURE 26-1](#)

[FIGURE 26-2](#)

[FIGURE 26-3](#)

[FIGURE 26-4](#)

[FIGURE 26-5](#)

[FIGURE 26-6](#)

[FIGURE 26-7](#)

[FIGURE 26-8](#)

[FIGURE 26-9](#)

[FIGURE 26-10](#)

Chapter 27

[FIGURE 27-1](#)

[FIGURE 27-2](#)

[FIGURE 27-3](#)

[FIGURE 27-4](#)

[FIGURE 27-5](#)

Chapter 28

[FIGURE 28-1](#)

[FIGURE 28-2](#)

[FIGURE 28-3](#)

[FIGURE 28-4](#)

[FIGURE 28-5](#)

Chapter 29

[FIGURE 29-1](#)

[FIGURE 29-2](#)

[FIGURE 29-3](#)

[FIGURE 29-4](#)

[FIGURE 29-5](#)

[FIGURE 29-6](#)

[FIGURE 29-7](#)

[FIGURE 29-8](#)

[FIGURE 29-9](#)

[FIGURE 29-10](#)

[FIGURE 29-11](#)

[FIGURE 29-12](#)

[FIGURE 29-13](#)

[FIGURE 29-14](#)

[FIGURE 29-15](#)

[FIGURE 29-16](#)

[FIGURE 29-17](#)

[FIGURE 29-18](#)

[FIGURE 29-19](#)

[FIGURE 29-20](#)

Chapter 30

[FIGURE 30-1](#)

[FIGURE 30-2](#)

[FIGURE 30-3](#)

[FIGURE 30-4](#)

Chapter 31

[FIGURE 31-1](#)

[FIGURE 31-2](#)

[FIGURE 31-3](#)

[FIGURE 31-4](#)

[FIGURE 31-5](#)

[FIGURE 31-6](#)

[FIGURE 31-7](#)

[FIGURE 31-8](#)

[FIGURE 31-9](#)

[FIGURE 31-10](#)

[FIGURE 31-11](#)

[FIGURE 31-12](#)

[FIGURE 31-13](#)

[FIGURE 31-14](#)

PROFESSIONAL C# and .NET

2021 Edition

Christian Nagel



INTRODUCTION

EVEN THOUGH .NET was announced in the year 2000, it is not becoming a grandfather technology. Instead, .NET keeps increasing developer traction since it has become open source and is available not only on Windows but also on Linux platforms. .NET can also run within the browser on the client—without the need to install a plugin—by using the WebAssembly standard.

As new enhancements for C# and .NET are coming, a focus lies not only on performance gains but also on ease of use. .NET more and more is a choice for new developers.

C# is also attractive for long-term developers. Every year, Stack Overflow asks developers about the most loved, dreaded, and wanted programming languages and frameworks. For several years, C# has been within the top 10 of the most loved programming languages. ASP.NET Core now holds the top position as the most loved web framework. .NET Core is number one in the most loved other frameworks/libraries/tools category. See <https://insights.stackoverflow.com/survey/2020> for details.

When you use C# and ASP.NET Core, you can create web applications and services (including microservices) that run on Windows, Linux, and Mac. You can use the Windows Runtime to create native Windows apps using C#, XAML, and .NET. You can create libraries that you share between ASP.NET Core, Windows apps, and .NET MAUI. You can also create traditional Windows Forms and WPF applications.

Most of the samples of this book are built to run on a Windows or Linux system. Exceptions are the Windows app samples that run only on the Windows platform. You can

use Visual Studio, Visual Studio Code, or Visual Studio for the Mac as the developer environment; only the Windows app samples require Visual Studio.

THE WORLD OF .NET

.NET has a long history; the first version was released in the year 2002. The new .NET generation with a complete rewrite of .NET (.NET Core 1.0 in the year 2016) is very young. Recently, many features from the old .NET version have been brought to .NET Core to ease the migration experience.

When creating new applications, there is no reason not to move to the new .NET versions. Whether old applications should stay with the old version of .NET or be migrated to the new one depends on the features used, how difficult the migration is, and what advantages you gain after the application is migrated. The best options here need to be considered with an application-by-application analysis.

The new .NET provides easy ways to create Windows and web applications and services. You can create microservices running in Docker containers in a Kubernetes cluster; create web applications; use the new OpenTelemetry standard to analyze distributed traces in a vendor-independent manner; create web applications returning HTML, JavaScript, and CSS; and create web applications returning HTML, JavaScript, and .NET binaries that run in the client's browser in a safe and standard way using WebAssembly. You can create Windows applications in traditional ways using WPF and Windows Forms and make use of modern XAML features and controls that support the fluent design with WinUI and mobile applications with .NET MAUI.

.NET uses modern patterns. Dependency injection is built into core services, such as ASP.NET Core and EF Core, which not only makes unit testing easier but also allows developers to easily enhance and change features from these technologies.

.NET runs on multiple platforms. Besides Windows and macOS, many Linux environments are supported, such as Alpine, CentOS, Debian, Fedora, openSUSE, Red Hat, SLES, and Ubuntu.

.NET is open source (<https://github.com/dotnet>) and freely available. You can find meeting notes for the C# compiler (<https://github.com/dotnet/csharplang>), the source code for the C# compiler (<https://github.com/dotnet/Roslyn>), the .NET runtime and libraries (<https://github.com/dotnet/runtime>), and ASP.NET Core (<https://github.com/dotnet/aspnetcore>) with Razor Pages, Blazor, and SignalR.

Here's a summary of some of the features of the new .NET:

- .NET is open source.
- .NET uses modern patterns.
- .NET supports development on multiple platforms.
- ASP.NET Core can run on Windows and Linux.

THE WORLD OF C#

When C# was released in the year 2002, it was a language developed for the .NET Framework. C# was designed with ideas from C++, Java, and Pascal. Anders Hejlsberg had come to Microsoft from Borland and brought experience from the language development of Delphi. At Microsoft, Hejlsberg worked on Microsoft's version of Java, named J++, before creating C#.

NOTE *Today, Anders Hejlsberg has moved to TypeScript (although he still influences C#), and Mads Torgersen is the project lead for C#. C# improvements are discussed openly at <https://github.com/dotnet/csharplang>, and you can read C# language proposals and event meeting notes. You can also submit your own proposals for C#.*

C# started not only as an object-oriented general-purpose programming language but was a component-based programming language that supported properties, events, attributes (annotations), and building assemblies (binaries including metadata).

Over time, C# was enhanced with generics, Language Integrated Query (LINQ), lambda expressions, dynamic features, and easier asynchronous programming. C# is not an easy programming language because of the many features it offers, but it's continuously evolving with features that are practical to use. With this, C# is more than an object-oriented or component-based language; it also includes ideas of functional programming—things that are of practical use for a general-purpose language developing all kinds of applications.

Nowadays, a new version of C# is released every year. C# 8 added nullable reference types, and C# 9 added records and more. C# 10 is releasing with .NET 6 in 2021 and C# 11 will be released with .NET 7 in 2022. Because of the frequency of changes nowadays, check the GitHub repository for the book (read more in the section “Source Code”) for continuous updates.

WHAT'S NEW IN C#

Every year, a new version of C# is released, with many new features available in each version. The latest versions include features such as nullable reference types to reduce exceptions of type `NullableReferenceException` and instead let the compiler help more; features to increase productivity such as indices and ranges; switch expressions that make the switch statement look old; features for using declarations; and enhancements with pattern matching. Top-level statements allow reducing the number of source code lines with small applications and records—classes where the compiler creates boilerplate code for equality comparison, deconstruction, and `with` expressions. Code generators allow creating code automatically while the compiler runs. All these new features are covered in this book.

WHAT'S NEW IN ASP.NET CORE

ASP.NET Core now contains new technology for creating web applications: Blazor Server and Blazor WebAssembly. With Blazor, you have a full-stack option to write C# code both for the client and for the server. With Blazor Server, the Razor components you create containing HTML and C# code run on the server. With Blazor WebAssembly, Razor components written with C# and HTML run on the client using the HTML 5 standard WebAssembly that allows you to run binary code in the browser, which is supported by all modern web browsers.

For creating services, you can now use gRPC with ASP.NET Core for binary communication between services. This is a great option for service-to-service communication to reduce the bandwidth needed, as well as CPU and memory usage if a lot of data transfer is needed.

WHAT'S NEW WITH WINDOWS

For developing applications for Windows, a new technology combines the features of the Universal Windows Platform and desktop applications: WinUI 3. WinUI is the native UI platform for Windows 10 applications. With WinUI 3, you can use modern XAML code that includes compiled binding to create desktop applications. New controls with Microsoft's fluent design system are available. These controls are not delivered with the Windows Runtime as was previously the case with the Universal Windows Platform (UWP). These controls are developed independently of the Windows 10 version that allows you to use the newest controls with Windows 10 versions 1809 and above. As the roadmap available with WinUI shows, these new controls will be usable from WPF applications as well.

WHAT YOU NEED TO WRITE AND RUN C# CODE

.NET runs on Windows, Linux, and Mac operating systems. You can create and build your programs on any of these operating systems using Visual Studio Code (<https://code.visualstudio.com>). You can build and run most of the samples on Windows or Linux and use the .NET development tools of your choice. Only the WinUI applications require you to use the Windows platform, and here, Visual Studio is the best option to use. The minimum version required to build and run the WinUI application is version 16.10.

The command line plays an important part when using the .NET CLI and the Azure CLI; you can use the new Windows Terminal. With the newest Windows 10 versions, this

terminal is delivered as part of Windows. With older versions, you can download it from the Microsoft Store.

Most .NET developers use the Windows platform as their development machine. When using the Windows Subsystem for Linux (WSL 2), you can build and run your .NET applications in a Linux environment, and you can install different Linux distributions from your Windows environment and access the same files. Visual Studio even allows debugging your .NET applications while they run in a Linux environment on WSL 2.

With some samples of the book, Microsoft Azure is shown as an optional hosting environment to run your web applications, use Azure Functions, and use Entity Framework Core to access SQL Server and Azure Cosmos DB. For this, you can use a free trial offering from Microsoft Azure; visit <https://azure.microsoft.com/free> to register.

WHAT THIS BOOK COVERS

This book covers these four major parts:

- The C# language
- Using base class libraries from .NET
- Developing web applications and services
- Developing Windows applications

Let's get into the different parts and all the chapters in more detail.

Part I, “The C# Language”

The first part of this book covers all the aspects of the C# programming language. You learn the syntax options and see how the C# syntax integrates with classes and interfaces from .NET. This part gives good grounding in the C# language. This section doesn't presume knowledge of any particular programming language, but it's assumed you are an experienced programmer. You start looking at C#'s basic syntax and data types before getting into advanced C# features.

- [Chapter 1](#), “.NET Applications and Tools,” covers what you need to know to create .NET applications. You learn about the .NET CLI and create a Hello World application using C# 9 top-level statements.
- [Chapter 2](#), “Core C#,” dives into core C# features and gives you details on top-level statements and information on declaration of variables and data types. The chapter covers target-typed new expressions, explains nullable reference types, and defines a program flow that includes the new `switch` expressions.

- [Chapter 3](#), “Classes, Records, Structs, and Tuples,” gives you information to create reference or value types, create and use tuples, and make use of the C# 9 enhancement to create and use records.
- [Chapter 4](#), “Object-Oriented Programming in C#,” goes into details of object-oriented techniques with C# and demonstrates all the C# keywords for object orientation. It also covers using inheritance with C# 9 records.
- [Chapter 5](#), “Operators and Casts,” explains the C# operators, and you also learn how to overload standard operators for custom types.
- [Chapter 6](#), “Arrays,” doesn't stop with simple arrays; you learn using multidimensional and jagged arrays, use the `Span` type to access arrays, and use the new index and range operators to access arrays.
- [Chapter 7](#), “Delegates, Lambdas, and Events,” covers .NET pointers to methods, lambda expressions with closures, and .NET events.
- [Chapter 8](#), “Collections,” dives into the different kind of collections, such as lists, queues, stacks, dictionaries, and immutable collections. The chapter also gives you the information you need to decide which collection to use in what scenario.
- [Chapter 9](#), “Language Integrated Query,” gives you the C# language integrated query features to query data from your collections. You also learn how to use multiple CPU cores with a query and what's behind expression trees that are used when you use LINQ to access your database with Entity Framework Core.
- [Chapter 10](#), “Errors and Exceptions,” covers how you should deal with errors, throw and catch exceptions, and filter exceptions when catching them.

- [Chapter 11](#), “Tasks and Asynchronous Programming,” shows the C# keywords `async` and `await` in action— not only with the task-based `async` pattern but also with `async` streams, which is a new feature since C# 8.
- [Chapter 12](#), “Reflection, Metadata, and Source Generators,” covers using and reading attributes with C#. The attributes will not just be read using reflection, but you'll also see the functionality of source generators that allow creating source code during compile time.
- [Chapter 13](#), “Managed and Unmanaged Memory,” is the last chapter of [Part I](#), which not only shows using the `IDisposable` interface with the `using` statement and the new `using` declaration but also demonstrates using the `Span` type with managed and unmanaged memory. You can read about using Platform Invoke both with Windows and with Linux environments.

Part II, “Libraries”

[Part II](#) starts with creating custom libraries and NuGet packages, but the major topics covered with [Part II](#) are for using .NET libraries that are important for all application types.

- [Chapter 14](#), “Libraries, Assemblies, Packages, and NuGet,” explains the differences between assemblies and NuGet packages. In this chapter, you learn how to create NuGet packages and are introduced to a new C# feature, module initializers, which allow you to run initial code in a library.
- [Chapter 15](#), “Dependency Injection and Configuration,” gives detail about how the `Host` class is used to configure a dependency injection container and the built-in options to retrieve configuration information

from a .NET application with different configuration providers, including Azure App Configuration and user secrets.

- [Chapter 16](#), “Diagnostics and Metrics,” continues using the `Host` class to configure logging options. You also learn about reading metric information that's offered from some NET providers, using Visual Studio App Center, and extending logging for distributed tracing with OpenTelemetry.
- [Chapter 17](#), “Parallel Programming,” covers myriad features available with .NET for parallelization and synchronization. [Chapter 11](#) shows the core functionality of the `Task` class. In [Chapter 17](#), more of the `Task` class is shown, such as forming task hierarchies and using value tasks. The chapter goes into issues of parallel programming such as race conditions and deadlocks, and for synchronization, you learn about different features available with the `lock` keyword, the `Monitor`, `SpinLock`, `Mutex`, `Semaphore` classes, and more.
- [Chapter 18](#), “Files and Streams,” not only covers reading and writing from the file system with new stream APIs that allow using the `Span` type but also covers the new .NET JSON serializer with classes in the `System.Text.Json` namespace.
- In [Chapter 19](#), “Networking,” you learn about foundational classes for network programming, such as the `Socket` class and how to create applications using TCP and UDP. You also use the `HttpClient` factory pattern to create `HttpClient` objects with automatic retries if transient errors occur.
- [Chapter 20](#), “Security,” gives you information about cryptography classes for encrypting data, explains how