Table of Contents

Introduction

We're a Team That Includes You

Technology authors write for a demanding group of people (for the best of possible reasons). You know that building software solutions using any platform or language is extremely complicated and is specific to your department, company, client base, and subject matter. Perhaps you work in the electronic publishing industry, develop systems for the state or local government, or work at NASA or a branch of the military. Collectively, we have worked in a variety of industries, including developing children's educational software (Oregon Trail/Amazon Trail), various enterprise systems, and projects within the medical and financial industries. The chances are almost 100 percent that the code you write at your place of employment has little to do with the code we have authored over the years.

Therefore, in this book, we have deliberately chosen to avoid creating demonstrations that tie the example code to a specific industry or vein of programming. Given this, we explain C#, OOP, the .NET Runtime, and the .NET Core base class libraries using industry-agnostic examples. Rather than having every example fill a grid with data, calculate payroll, or some other domain-specific task, we stick to subject matter we can all relate to: automobiles (with some geometric structures and employee payroll systems thrown in for good measure). And that's where you come in.

Our job is to explain the C# programming language and the core aspects of the .NET 5 platform the best we possibly can. As well, we will do everything we can to equip you with the tools and strategies you need to continue your studies at this book's conclusion.

Your job is to take this information and apply it to your specific programming assignments. We obviously understand that your projects most likely don't revolve around automobiles with friendly pet names (Zippy the BMW or a Yugo named Clunker, among others), but that's what applied knowledge is all about!

Rest assured, once you understand the topics and concepts presented within this text, you will be in a perfect position to build .NET 5 solutions that map to your own unique programming environment.

An Overview of This Book

Pro C# 9 with .NET 5 is logically divided into eight distinct parts, each of which contains a number of related chapters. Here is a part-by-part and chapter-by-chapter breakdown of the text.

Part I: Introducing C# and .NET 5

The purpose of Part I is to acclimate you to the nature of the .NET 5 platform and various development tools used during the construction of .NET 5 applications.

Chapter 1: Introducing C# and .NET (Core) 5

This first chapter functions as the backbone for the remainder of the text. The primary goal of this chapter is to acquaint you with a number of .NET Core-centric building blocks, such as the Common Language Runtime (CLR), Common Type System (CTS), Common Language Specification (CLS), and Base Class Libraries (BCL). Here, you will take an initial look at the C# programming language, namespaces, and the .NET 5 assembly format.

Chapter 2: Building C# Applications

The goal of this chapter is to introduce you to the process of compiling C# source code files. After installing the .NET 5 SDK and runtimes, you will learn about the completely free (and fully functional) Visual Studio Community edition as well as the extremely popular (and also free) Visual Studio Code. You learn how to create, run, and debug .NET 5 C# applications using both Visual Studio and Visual Studio Code.

Part II: Core C# Programming

The topics presented in this part of the book are quite important because you will use them regardless of which type of .NET 5 software you intend to develop (e.g., web applications, desktop GUI applications, code libraries, services, etc.). Here, you will learn about the fundamental data types of .NET 5, work with text manipulation, and learn the role of various C# parameter modifiers (including optional and named arguments).

Chapter 3: Core C# Programming Constructs, Part 1

This chapter begins your formal investigation of the C# programming language. Here, you will learn about the role of the `Main()` method, top-level statements (new in C# 9.0), and numerous details regarding the intrinsic data types of the .NET 5 platform and variable declaration. You will work with and manipulate textual data using `System.String` and `System.Text.StringBuilder`. You will also examine iteration and decision constructs, pattern matching, narrowing and widening operations, and the `unchecked` keyword.

Chapter 4: Core C# Programming Constructs, Part 2

This chapter completes your examination of the core aspects of C#, beginning with creating and manipulating arrays of data. Next, you examine how to construct overloaded type methods and define parameters using the `out`, `ref`, and `params` keywords. You will also learn about the enum type, structures, and nullable data types, and you will understand the distinction between value types and reference types. Finally, you will learn about tuples, a new feature in C# 7 and updated in C# 8.

Part III: Object-Oriented Programming with C#

In this part, you will come to understand the core constructs of the C# language, including the details of object-oriented programming. This part will also examine how to process runtime exceptions and will dive into the details of working with strongly typed interfaces. Finally, you will learn about object lifetime and garbage collection.

Chapter 5: Understanding Encapsulation

This chapter begins your examination of object-oriented programming (OOP) using the C# programming language. After you are introduced to the pillars of OOP (encapsulation, inheritance, and polymorphism), the remainder of this chapter will show you how to build robust class types using constructors, properties, static members, constants, and read-only fields. You will also learn about partial type definitions, object initialization syntax, and automatic properties, and the chapter will wrap up with an examination of record types, new in C# 9.0.

Chapter 6: Understanding Inheritance and Polymorphism

Here, you will examine the remaining pillars of OOP (inheritance and polymorphism), which allow you to build families of related class types. As you do this, you will examine the role of virtual methods, abstract methods (and abstract base classes), and the nature of the polymorphic interface. Then you will explore pattern matching with the `is` keyword, and finally, this chapter will explain the role of the ultimate base class of the .NET Core platform, `System.Object`.

Chapter 7: Understanding Structured Exception Handling

The point of this chapter is to discuss how to handle runtime anomalies in your code base through the use of structured exception handling. Not only will you learn about the C# keywords that allow you to handle such problems (`try`, `catch`, `throw`, `when`, and `finally`), but you will also come to understand the distinction between application-level and system-level exceptions. In addition, this chapter will show you how to set Visual Studio on break on all exceptions to debug the exceptions that escape your notice.

Chapter 8: Working with Interfaces

The material in this chapter builds upon your understanding of object-based development by covering the topic of interface-based programming. Here, you will learn how to define classes and structures that support multiple behaviors, how to discover these behaviors at runtime, and how to selectively hide particular behaviors using explicit interface implementation. In addition to creating a number of custom interfaces, you will also learn how to implement standard interfaces found within the .NET Core platform. You will use these to build objects that can be sorted, copied, enumerated, and compared.

Chapter 9: Understanding Object Lifetime

The final chapter of this part examines how the CLR manages memory using the .NET Core garbage collector. Here, you will come to understand the role of application roots, object generations, and the `System.GC` type. Once you understand the basics, you will examine the topics of disposable objects (using the `IDisposable` interface) and the finalization process (using the `System.Object.Finalize()` method). This chapter will also investigate the `Lazy<T>` class, which allows you to define data that will not be allocated until requested by a caller. As you will see, this feature can be helpful when you want to ensure you do not clutter the heap with objects that are not actually required by your programs.

Part IV: Advanced C# Programming

This part of the book will deepen your understanding of the C# language by walking you through a number of more advanced (but important) concepts. Here, you will complete your examination of the .NET Core type system by investigating collections and generics. You will also examine a number of more advanced features of C# (e.g., extension methods, operator overloading, anonymous types, and pointer manipulation). You will then examine delegates and lambda expressions, take a first look at Language Integrated Query, and finish the section with two chapters that focus on processes and multithreaded/async programming.

Chapter 10: Collections and Generics

This chapter explores the topic of *generics*. As you will see, generic programming gives you a way to create types and type members, which contain various *placeholders* that can be specified by the caller. In a nutshell, generics greatly enhance application performance and type safety. Not only will you explore various generic types within the `System.Collections.Generic` namespace, but you will also learn how to build your own generic methods and types (with and without constraints).

Chapter 11: Advanced C# Language Features

This chapter deepens your understanding of the C# programming language by introducing you to a number of advanced programming techniques. Here, you will learn how to overload operators and create custom conversion routines (both implicit and explicit) for your types. You will also learn how to build and interact with type indexers, as well as work with extension methods, anonymous types, partial methods, and C# pointers using an unsafe code context.

Chapter 12: Delegates, Events, and Lambda Expressions

The purpose of this chapter is to demystify the delegate type. Simply put, a .NET Core delegate is an object that points to other methods in your application. Using this type, you can build systems that allow multiple objects to engage in a two-way conversation. After you have examined the use of .NET Core delegates, you will then be introduced to the C# `event` keyword, which you can use to simplify the manipulation of raw delegate programming. You will wrap up this chapter by investigating the role of the C# lambda operator (`=>`) and exploring the connection between delegates, anonymous methods, and lambda expressions.

Chapter 13: LINQ to Objects

This chapter begins your examination of Language Integrated Query (LINQ). LINQ allows you to build strongly typed query expressions that can be applied to a number of LINQ targets to manipulate data in the broadest sense of the word. Here, you will learn about LINQ to Objects, which allows you to apply LINQ expressions to containers of data (e.g., arrays, collections, and custom types). This information will serve you well as you encounter a number of additional LINQ APIs throughout the remainder of this book.

Chapter 14: Processes, AppDomains, and Load Contexts

Now that you have a solid understanding of assemblies, this chapter dives deeper into the composition of a loaded .NET Core executable. The goal of this chapter is to illustrate the relationship between processes, application domains, and contextual boundaries. These topics provide the proper foundation for Chapter 15, where you will examine the construction of multithreaded applications.

Chapter 15: Multithreaded, Parallel, and Async Programming

This chapter examines how to build multithreaded applications and illustrates a number of techniques you can use to author thread-safe code. The chapter opens by revisiting the .NET delegate type to ensure explaining a delegate's intrinsic support for asynchronous method invocations. Next, you will investigate the types within the `System.Threading` namespace. The next section covers the Task Parallel Library (TPL). Using the TPL, .NET developers can build applications that distribute their workload across all available CPUs in a wickedly simple manner. At this point, you will also learn about the role of Parallel LINQ, which provides a way to create LINQ queries that scale across multiple machine cores. The remainder of the chapter covers creating nonblocking calls using the `async/await` keywords, introduced in C# 5; local functions and generalized `async` return types, both new in C# 7; and asynchronous streams, introduced in C# 8.

Part V: Programming with .NET Core Assemblies

Part V dives into the details of the .NET Core assembly format. Not only will you learn how to deploy and configure .NET Core code libraries, but you will also come to understand the internal composition of a .NET Core binary image. This section explains the role of .NET Core attributes and the role of resolving type information at runtime and the role of the Dynamic Language Runtime (DLR) and the C# `dynamic` keyword. The final chapter covers the syntax of Common Intermediate Language (CIL) and the role of dynamic assemblies.

Chapter 16: Building and Configuring Class Libraries

At a high level, *assembly* is the term used to describe a binary file created with a .NET Core compiler. However, the true story of .NET Core assemblies is far richer than that. Here, you will learn how to build and deploy assemblies and learn the difference between class libraries and console applications and the difference between .NET Core and .NET Standard class libraries. The final section covers the new options available in .NET 5, such as single file executables and ready-to-run publishing.

Chapter 17: Type Reflection, Late Binding, and Attribute-Based Programming

This chapter continues your examination of .NET Core assemblies by checking out the process of runtime type discovery using the `System.Reflection` namespace. Using the types of this namespace, you can build applications that can read an assembly's metadata on the fly. You will also learn how to load and create types at runtime dynamically using late binding. The final topic of this chapter will explore the role of .NET Core attributes (both standard and custom). To illustrate the usefulness of each of these topics, the chapter shows you how to construct an extendable application complete with snap-ins.

Chapter 18: Dynamic Types and the Dynamic Language Runtime

.NET 4.0 introduced a new aspect of the .NET runtime environment called the *Dynamic Language Runtime*. Using the DLR and the C# `dynamic` keyword, you can define data that is not truly resolved until runtime. Using these features simplifies some complex .NET Core programming tasks dramatically. In this chapter, you will learn some practical uses of dynamic data, including how to leverage the .NET Core reflection APIs in a streamlined manner, as well as how to communicate with legacy COM libraries with a minimum of fuss and bother.

Chapter 19: Understanding CIL and the Role of Dynamic Assemblies

The goal of the final chapter in this section is twofold. The first part examines the syntax and semantics of CIL in much greater detail than in previous chapters. The remainder of this chapter will cover the role of the `System.Reflection.Emit` namespace. You can use these types to build software that can generate .NET Core assemblies in memory at runtime. Formally speaking, assemblies defined and executed in memory are termed *dynamic assemblies*.

Part VI: File Handling, Object Serialization, and Data Access

By this point in the text, you have a solid handle on the C# language and the details of the .NET Core assembly format. Part VI leverages your newfound knowledge by exploring a number of commonly used services found within the base class libraries, including file I/O, object serialization, and database access using ADO.NET.

Chapter 20: File I/O and Object Serialization

The `System.IO` namespace allows you to interact with a machine's file and directory structure. Over the course of this chapter, you will learn how to create (and destroy) a directory system programmatically. You will also learn how to move data into and out of various streams (e.g., file based, string based, and memory based). The latter part of this chapter will examine the XML and JSON object serialization services of the .NET Core platform. Simply put, serialization allows you to persist the public state of an object (or a set of related objects) into a stream for later use. Deserialization (as you might expect) is the process of plucking an object from the stream into memory for consumption by your application.

Chapter 21: Data Access with ADO.NET

This chapter covers database access using ADO.NET, the database API for .NET Core applications. Specifically, this chapter will introduce you to the role of .NET data providers and how to communicate with a relational database using ADO.NET, which is represented by connection objects, command objects, transaction objects, and data reader objects. This chapter also begins the creation of the AutoLot database, which will be enhanced in Chapters 22 and 23.

Part VII: Entity Framework Core

By this point in the text, you have a solid handle on the C# language and the details of the .NET Core assembly format. Part VI leverages your newfound knowledge by exploring a number of commonly used services found within the base class libraries, including file I/O, database access using ADO.NET, and database access using Entity Framework Core.

Chapter 22: Introducing Entity Framework Core

This chapter introduces Entity Framework (EF) Core. EF Core is an object-relational mapping (ORM) framework built on top of ADO.NET. EF Core provides a way to author data access code using strongly typed classes that directly map to your business model. Here, you will come to understand the building blocks of EF Core, including `DbContext`, entities, the specialized collection class `DbSet<T>`, and the `DbChangeTracker`. Next, you will learn about query execution, tracking versus nontracking queries, and some of the more notable features of EF Core. The final piece of theory in this chapter is the EF Core global tool for the .NET Core command-line interface (CLI).

Chapter 23: Build a Data Access Layer with Entity Framework Core

This chapter builds the AutoLot data access layer. It begins with scaffolding the AutoLot database from Chapter 21 into a derived `DbContext` and entity classes. Then the project and database are updated to move to a code first approach. The entities are updated to their final version, and a migration is created and executed to update the database to match the entities. The final database change is to create a migration for the stored procedure from Chapter 21 and a new database view. Repositories are added for code encapsulation, and a data initialization process is created. The final part of the chapter test drives the data access layer using xUnit for automated integration tests.

Part VIII: Windows Presentation Foundation

The initial desktop GUI API supported by the .NET platform was termed Windows Forms. While this API is still fully supported, .NET 3.0 introduced programmers to an API called Windows Presentation Foundation (WPF). Unlike Windows Forms, this framework integrates a number of key services, including data binding, 2D and 3D graphics, animations, and rich documents, into a single, unified object model. This is all accomplished using a declarative markup grammar called Extensible Application Markup Language (XAML). Furthermore, the WPF control architecture provides a trivial way to restyle the look and feel of a typical control radically using little more than some well-formed XAML.

Chapter 24: Introducing Windows Presentation Foundation and XAML

In this chapter, you will begin by examining the motivation behind the creation of WPF (when there was already a desktop development framework in .NET). Then, you will learn about the syntax of XAML and, finally, take a look at the Visual Studio support for building WPF applications.

Chapter 25: WPF Controls, Layouts, Events, and Data Binding

This chapter will expose you to the process of using intrinsic WPF controls and layout managers. For example, you will learn to build menu systems, splitter windows, toolbars, and status bars. This chapter will also introduce you to a number of WPF APIs (and their related controls), including the Ink API, commands, routed events, the data-binding model, and dependency properties.

Chapter 26: WPF Graphics Rendering Services

WPF is a graphically intensive API; given this fact, WPF provides three ways to render graphics: shapes, drawings and geometrics, and visuals. In this chapter, you will evaluate each option and learn about a number of important graphics primitives (e.g., brushes, pens, and transformations) along the way. This chapter will also examine ways to incorporate vector images into your WPF graphics, as well as how to perform hit-testing operations against graphical data.

Chapter 27: WPF Resources, Animations, Styles, and Templates

This chapter will introduce you to three important (and interrelated) topics that will deepen your understanding of the Windows Presentation Foundation API. The first order of business is to learn the role of logical resources. As you will see, the logical resource (also termed an *object resource*) system provides a way for you to name and refer to commonly used objects within a WPF application. Next, you will learn how to define, execute, and control an animation sequence. Despite what you might be thinking, however, WPF

animations are not limited to the confines of video games or multimedia applications. You will wrap up the chapter by learning about the role of WPF styles. Similar to a web page that uses CSS or the ASP.NET theme engine, a WPF application can define a common look and feel for a set of controls.

Chapter 28: WPF Notifications, Validations, Commands, and MVVM

This chapter begins by examining three core WPF framework capabilities: notifications, validations, and commands. In the notifications section, you will learn about observable models and collections and how they keep your application data and UI in sync. Next, you will dig deeper into commands, building custom commands to encapsulate your code. In the validations section, you will learn how to use the several validation mechanisms available in WPF applications. The chapter closes with an examination of the Model-View-ViewModel (MVVM) pattern and ends by creating an application that demonstrates the MVVM pattern in action.

Part IX: ASP.NET Core

Part VIII is devoted to an examination of constructing web applications using ASP.NET Core. ASP.ENT Core can be used to build web applications and RESTful services.

Chapter 29: Introducing ASP.NET Core

This chapter introduces ASP.NET Core and the Model-View-Controller (MVC) pattern. The next section lists (and explains) features brought forward into ASP.NET Core from classic ASP.NET MVC/WebAPI. These include controllers and actions, model binding, routing, and filters. The next section covers new features introduced in ASP.NET Core, including built-in dependency injection, cloud-ready, environmentally aware configuration system, deployment patterns, and the HTTP request pipeline. The rest of the chapter creates the two ASP.NET Core projects that will be finished in the next two chapters, shows the options to run ASP.NET Core applications, and begins the configuration process of the two ASP.NET Core projects.

Chapter 30: RESTful Services with ASP.NET Core

This chapter finishes the ASP.NET Core RESTful service application. The chapter begins with a demonstration of the different mechanisms for returning JSON results to a client and the built-in support for service applications provided by the `ApiController` attribute. Swagger/OpenAPI is added next to provide a platform for testing and documenting your service. The rest of the chapter builds the controllers for the application, as well as creating the exception filter.

Chapter 31: MVC Applications with ASP.NET Core

This is the final chapter covering ASP.NET Core and finishes the MVC-based web application. The chapter starts with a deep look in views and the Razor View Engine, including layouts and partials. Next, tag helpers (another new feature in ASP.NET Core) are explored, followed by managing client-side libraries and bundling/minification of those libraries. Next, the `CarsController` and its views are built, along with the custom tag helpers. A view component is added for the data-driven menu, and the options pattern is explored. Finally, an HTTP client service wrapper is created, and `CarsController` updated to use the ASP.NET Core service instead of the AutoLot data access layer.