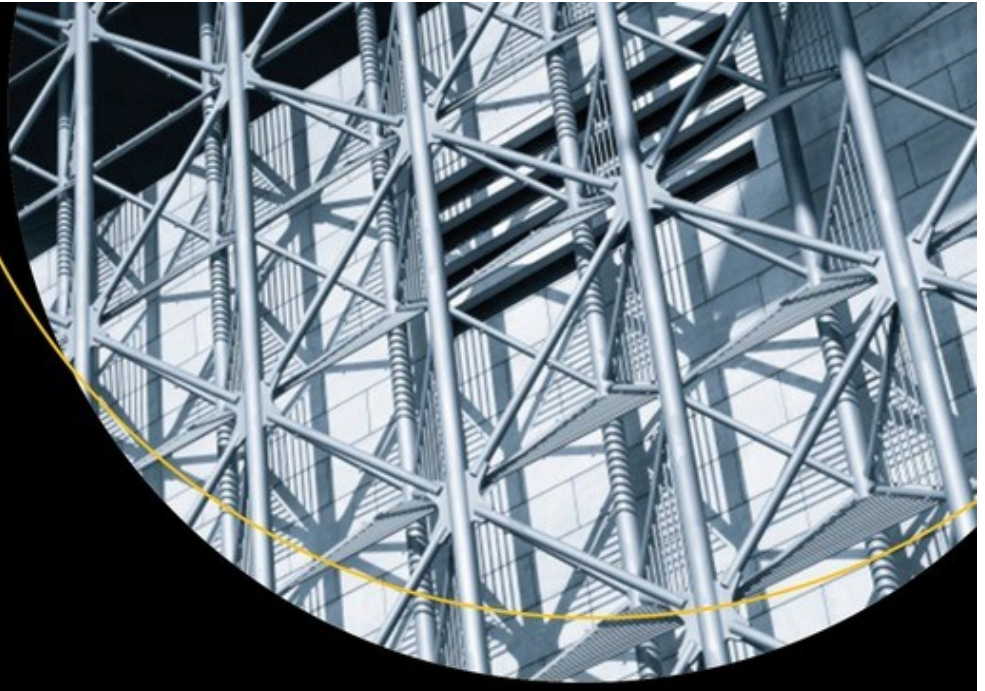

Introducing .NET 6

Getting Started with Blazor, MAUI, Windows App SDK, Desktop Development, and Containers

Nico Vermeir

Apress®

Nico Vermeir

Introducing .NET 6
Getting Started with Blazor, MAUI, Windows App SDK, Desktop Development, and Containers

Apress®

Nico Vermeir
Merchtem, Belgium

ISBN 978-1-4842-7318-0 e-ISBN 978-1-4842-7319-7
https://doi.org/10.1007/978-1-4842-7319-7

This Apress imprint is published by the registered company APress Media, LLC part of Springer Nature.
The registered company address is: 1 New York Plaza, New York, NY 10004, U.S.A.

Introduction

Welcome to .NET 6! A very exciting new release of Microsoft's managed application runtime and SDK. .NET 6 is a release that has been long in the making; it is the next step in the one .NET dream. In this book, we will discover what .NET 6 has to offer; we will learn about exciting updates on existing frameworks like WinForms and WPF and discover new things like Minimal APIs.

We will start with a quick tour around .NET 6 in the first chapter, just to get a feel of how big this .NET release really is. In the next chapter, we will go a bit more technical and see what the different runtimes are and how a cross-platform framework like .NET still manages to run native applications on Windows, mobile, and more. In Chapter 3, we will go into the command line tooling; here we will discover that Visual Studio is not performing any magic tricks, it's just calling the CLI underneath. In Chapters 4–7, we will learn about the different application frameworks .NET hosts, from native Windows desktop to web applications with ASP.NET Core to cross-platform mobile applications. From there, we cross over into the cloud and see Azure's support for .NET 6. The final three chapters are a bit more advanced; we go into application architecture and what .NET 6 and C# 10 features help write better architectured code, and we take a look at the compiler platform, or Roslyn. And finally we end on a chapter with some more advanced topics like threading and async/await.

The book is written in a demo-based manner. Feel free to pull up your computer and follow along while reading; all the steps are explained so that we can discover the topics together.

Happy learning!

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page.

Acknowledgments

Thank you to the great people at Apress for the support during the writing of this book.

Thank you Damien Foggon for the technical review; you've helped me grow as an author and made this a better book.

A big thank you to the worldwide .NET community. Each and every one of you keeps pushing me every day to grow as a developer, as a community member, and as a person.

Table of Contents

About the Author

Nico Vermeir

is a Microsoft MVP in the field of Windows development. He works as an application architect at Inetum-Realdolmen Belgium and spends a lot of time keeping up with the rapidly changing world of technology. He loves talking about and using the newest and experimental technologies in the .NET stack. Nico cofounded MADN, a user group focusing on building modern applications in .NET. He regularly presents on the topic of .NET at user groups and conferences.

In his free time, you can find him enjoying rides on his motorcycle, jamming on his guitar, or having a beer with friends.

N. Vermeir, *Introducing .NET 6*
https://doi.org/10.1007/978-1-4842-7319-7_1

1. A Tour of .NET 6

Nico Vermeir[1] ✉
(1) Merchtem, Belgium

Welcome to .NET 6! An exciting new release of Microsoft's popular framework. .NET 6 is the next big step in delivering the "one .NET" vision. The vision that would unify all of .NET to have a single runtime for mobile, Web, IoT, games, and many more targets.

In this first chapter, we will look at the versioning of .NET, together with its support timeframes and release schedule. We will go over the supported operating systems, what it means to have a unified platform, and how to get started with .NET 6 using the command line interface.

.NET 6

The .NET framework has been around since the year 2000. Over the years, it has grown into a very mature, popular framework that could target many platforms. However, sharing code between those different platforms was not an easy task because of how the .NET framework was built. With .NET Core, Microsoft wanted to start from a clean slate using .NET Standard as a way to share code between the different platforms. They took the API surface of the base class library and started implementing everything anew, using modern techniques and APIs to improve performance of the framework. .NET Standard was created as an interface. It exposed parts of the BCL API; .NET Core is an implementation of that .NET Standard interface. Because .NET Standard was an abstraction, we could create .NET Standard class libraries that could be referenced from every type of platform, as long as they used the correct version of .NET Core. This quickly became confusing as .NET Core 3 was using .NET Standard 2.1, but .NET Standard 1.6 was .NET Core 1.0. The next step in unifying all of .NET

was taken with .NET 5. .NET 5, which was actually .NET Core 4, was the release where .NET Core became the successor of the classic .NET Framework. The final release of the classic .NET Framework is 4.8; from that moment on, .NET Core is the main branch of the framework. To avoid confusion, later on .NET Core was renamed to simply .NET and the versioning of the classic .NET framework was taken over, hence .NET 5. .NET Standard disappeared as well; as of .NET 5, we just have .NET 5 class libraries and those are compatible with every platform that is on .NET 5. .NET 6 is one of the final steps in unifying the platform as it unifies Mono and .NET, fulfilling the "one .NET" dream.

So far we have spoken about .NET, .NET Core, and .NET Framework. This might get a bit confusing, so here is how I will talk about the different types of .NET in this book:

- .NET: This is .NET 5, .NET 6, and future releases. It is the unified release.
- .NET Core: This is the previous release that wasn't .NET Framework.
- .NET Framework: The classic .NET framework that ended on version 4.8.

Version Support

First and foremost, .NET 6 is a Long Term Support release (LTS), meaning it will receive updates for the coming 3 years (up until, and including, 2024). Non-LTS versions are supported for 1 year, usually up to 3 months after the next LTS version release.

So how do we recognize LTS versions from non-LTS versions? Simple. Every odd numbered release (.NET 5, .NET 7, etc.) will be a non-LTS release and every even numbered release (.NET 6, .NET 8, etc.) will be an LTS release (Figure 1-1). The current release cadence Microsoft has set for .NET is a new release every year around November. This release cadence was introduced with .NET 5.

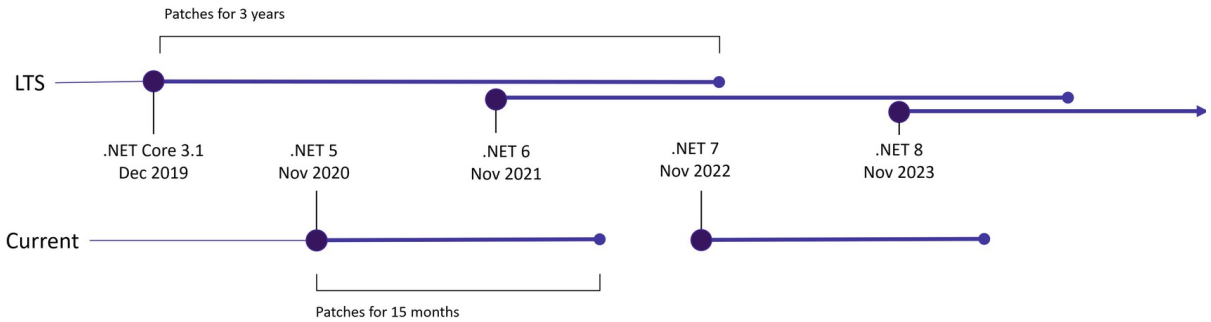

Figure 1-1 Release timeline of .NET (Source: Microsoft)

Why is this important? If you're starting a new software project, it's important to know that the underlying framework will not cause any security risks. No software is bug-free, so bugs and security risks will show up over the lifetime of any software; .NET is no exception. Writing your software using a version of .NET that will receive patches and updates for the coming years ensures that vulnerabilities and bugs in the framework get patched instead of potentially make your application crash, or vulnerable for attacks.

Does this mean that we can forget about the odd-numbered releases, since they are only supported for about a year? Not necessarily, it all depends on the context around the software you're developing. If you're building software that will still be in active development by the time of the next .NET LTS release, it can easily be included in the backlog to upgrade to the next version once it lands. If you're building software that will be delivered in the current non-LTS timeframe and there's no maintenance planned on the software, make sure your customer knows about the support. So, as usual it depends. Luckily, upgrading to a next release usually isn't very difficult. If you are in a consultant role, set the correct expectations to your customer.

> **Tip** Do not jump on the latest version of .NET just because it's the latest version, be sure to check the support status, inform your customer when applicable, and make a well-informed decision.

Supported Versions

There are multiple versions of .NET under active support at any given time. Table 1-1 gives an overview of the support status of the more recent .NET versions.

Table 1-1 An overview of .NET versions and their support status

Version	Original release date	Support level	End of support
.NET 6	November 2021	LTS	February 2025
.NET 5	November 2020	Non-LTS	February 2022
.NET Core 3.1	December 2019	LTS	December 2022

All details concerning support for .NET can be found in the official

.NET Support Policy found at https://dotnet.microsoft.com/platform/support/policy/dotnet-core.

A Unified Platform

From the very start, .NET Core was meant to be cross-platform and cross-idiom. Its purpose was to bring a bunch of separate, .NET-based technologies together under one umbrella. Before .NET Core, we could do different styles of apps, but not all of those were part of .NET, for example, Mono, the open-source .NET implementation for Linux- and Unix-based systems and Xamarin, the native mobile .NET solution built on Mono.

.NET Core 3 shifted the unification of .NET into high gear by adding Windows Presentation Foundation (WPF) and Windows Forms (WinForms) support into the framework. .NET 5 expanded on this work by adding Mono; the work on Mono brought .NET into the WebAssembly world. Blazor WebAssembly was the first result of this unification. With Blazor WebAssembly, we got native .NET running in the browser, using Mono. More information on Blazor can be found in Chapter 5 of this book. .NET 6 delivers the fully realized unified vision by including Xamarin as a part of .NET instead of a separate framework.

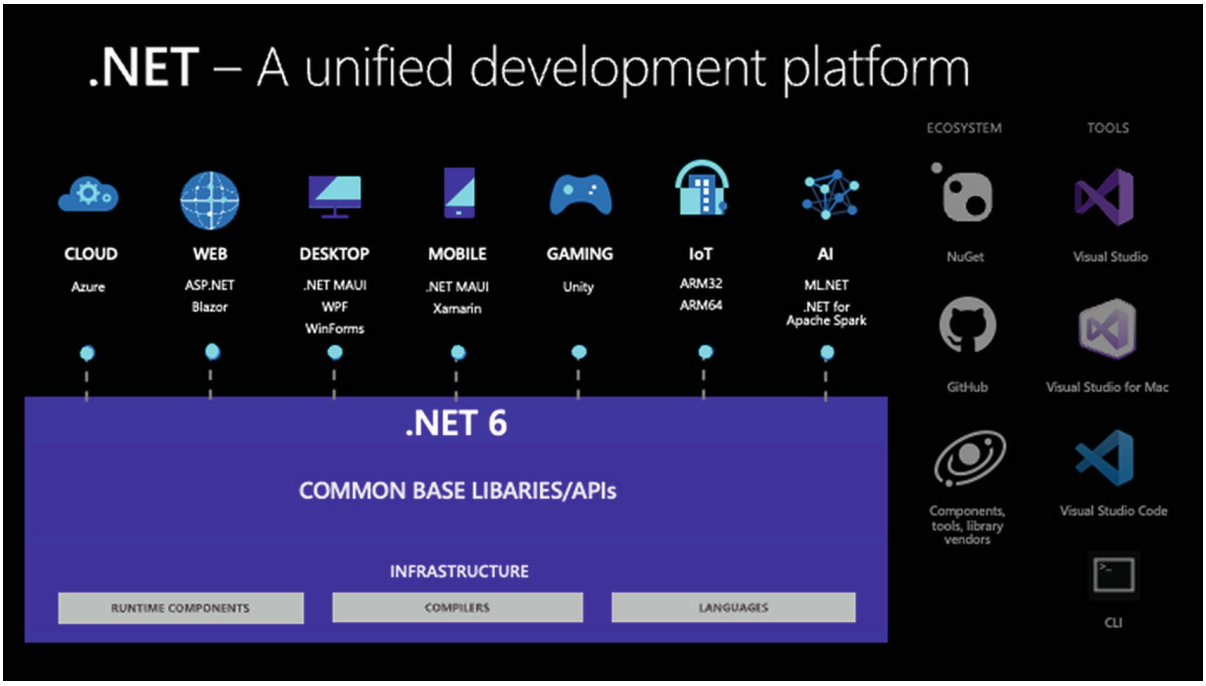

Figure 1-2 .NET – a unified platform (Source: Microsoft)

Xamarin is no longer the mobile platform that happens to look like .NET. It's now a part of the framework, using .NET class libraries and .NET SDK tools to provide a great developer experience. A quick example of this is being able to use `dotnet new ios` or `dotnet new android` followed by `dotnet build` or `dotnet run`. As a result, you'll see a mobile project being created, compiled and running on either a physical device or emulator. This is the result of work that started back in .NET 5, by bringing Mono into .NET.

We'll dive deeper into Xamarin in the MAUI chapter of this book.

Roadmap

Microsoft made the decision to openly develop .NET, something they've done since .NET Core. That means that the backlog for .NET 6, and future versions, is visible to everyone. There's even a Blazor-based web application that shows an overview of what's proposed, what's in progress, and what's been completed. The website can be found at *https://themesof.net/*, and because everything happens out in the open, the Blazor web app's source code is available at *https://github.com/terrajobst/themesof.net*.

The .NET team uses GitHub and GitHub Issues, Boards, and Milestones to keep track of their work. Although GitHub Issues is not very agile-friendly, especially when compared to tools like Azure DevOps or Jira, they have identified four categories of issues. Issues are categorized using labels. The four labels, as per their website, are as follows:

1. **Theme**: A top-level/overarching objective that will span the project leases. A theme will often have an associated document describing those objectives.
2. **Epic**: This is a higher level grouping of related user stories; it can span up to the entire release. For example, "Enterprises have a first class experience acquiring and deploying .NET 6.0."
3. **User story**: An explanation of the feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer. Once implemented, it will contribute value toward the overall epic. For example, "As an IT Pro, I have easy access to .NET Core installer

release information and scripts in my air gapped environment so I can use this to determine which updates need to be deployed."

4. **Issue**: These are all other work items. These could be bugs, features, or developer tasks. We leave it up to the engineering team/area owner how and if they want to use these.

Supported Operating Systems

Since .NET is a cross-platform framework, there are a multitude of operation systems supported. Support ranges from Windows to Linux, macOS, Android, iOS, and tvOS. Table 1-2 lists the different versions of Windows that support .NET 6.

Table 1-2 Versions of Windows that support .NET 6

Operating system	Version	Architecture
Windows	7 SP1, 8.1	x64, x86
Windows 10	Version 1607+	x64, x86, ARM64
Windows 11	Version 22000+	x64, x86, ARM64
Windows Server	2012+	x64, x86
Windows Server Core	2012+	x64, x86
Nano Server	Version 1809+	x64

Table 1-3 lists the supported Linux distributions with the supported versions and architecture.

Table 1-3 Linux versions that support .NET 6

Operating system	Version	Architecture
Alpine Linux	3.13+	x64, ARM64, ARM32
CentOS	7+	x64
Debian	10+	x64, x86, ARM64, ARM32
Fedora	33+	x64
openSUSE	15+	x64
Red Hat Enterprise Linux	7+	x64, ARM64
SUSE Enterprise Linux	12 SP2+	x64
Ubuntu	16.04, 18.04, 20.04+	x64, ARM64, ARM32

Table 1-4 lists the supported versions and architectures for macOS.

Table 1-4 macOS versions that support .NET 6

Operating system	Version	Architecture
macOS	10.15+	x64, ARM64

Table 1-5 lists the supported versions and architectures for Android.

Table 1-5 Android versions that support .NET 6

Operating system	Version	Architecture
Android	API 21+	x64, ARM, ARM64

Table 1-6 lists the supported versions and architectures for iOS and tvOS.

Table 1-6 iOS and tvOS versions that support .NET 6

Operating system	Version	Architecture
iOS	10.0+	x64, ARM, ARM64
tvOS	10.0+	x64, ARM, ARM64

The above tables list supported operating systems, versions, and architectures at the time of writing. The most up-to-date version of this list for .NET 6 is available at *https://github.com/dotnet/core/blob/main/release-notes/6.0/supported-os.md*.

Command Line Interface

.NET ships with a powerful Command Line Interface (CLI) tooling system since .NET Core. With the .NET command line, we can do things like creating a new project, installing tools and templates, running tests, compiling, and much more. While most of the CLI commands are rarely used manually, we can use them to script build and deploy automation. Tools like Azure DevOps or GitHub actions have full support for these commands.

The basic commands consist of:

- New
- Restore
- Build

- Publish
- Run
- Test
- Vstest
- Pack
- Migrate
- Clean
- Sln
- Help
- Store

Before we use the CLI, we have to install .NET 6 on our machine. If you have Visual Studio 2022 installed, you might already have it up and running.

We can see what version of .NET we are currently running by opening up a Powershell prompt and executing `dotnet --version`.

Figure 1-3 Current installed version of .NET

If you get another version, maybe from .NET 5, you can download the .NET 6 installer from https://dotnet.microsoft.com/download/dotnet/6.0. Make sure to download and install the SDK to get the command line tooling.

Once .NET 6 is installed, we can see what project templates we have by executing `dotnet new`. The tooling will list all available options if we don't specify a specific template as shown in Figure 1-4. The contents of this list depend of course on the different workloads and templates you have installed on your system.