Agenda: What is .NET and its Evolution

- What is .NET
- Types of Applications we can develop in .NET
- Primary .NET Implementations
- About MS.NET Framework
- Important Components of .NET Framework
- Architecture of .NET Framework
- About Mono and Xamarin
- About Universal Windows Platform (UWP)
- What is .NET Core
- Benefits of .NET core
- What is new in .NET Core
- What is .NET Standard
- .NET Core vs .NET Framework
- About .NET 5 and later versions.

What is .NET

- .NET is a free, cross-platform, open-source development platform for building many kinds of apps.
- It's a **.NET Foundation** project and is maintained by Microsoft and the community on GitHub.
- .NET is supported by Microsoft on **Windows, macOS, Linux, IoS and Android.**
- Red Hat supports .NET on Red Hat Enterprise Linux (RHEL).
- Samsung supports .NET on Tizen platforms.

Types of applications we can develop using .NET

a) **Windows apps**

- Windows Desktop apps
  - Windows Forms
  - Windows WPF
  - Universal Windows Platform (UWP)
  - MAUI
- Windows services

b) **Cross-platform client apps**

- Desktop apps
- Games

- Mobile apps

c) **Cloud apps**
- Cloud native apps
- Console apps
- Serverless functions in the cloud
- Web apps, web APIs, and microservices

d) **Other app types**
- Machine learning and AI
- Internet of Things (IoT)

Programming Languages actively supported by Microsoft

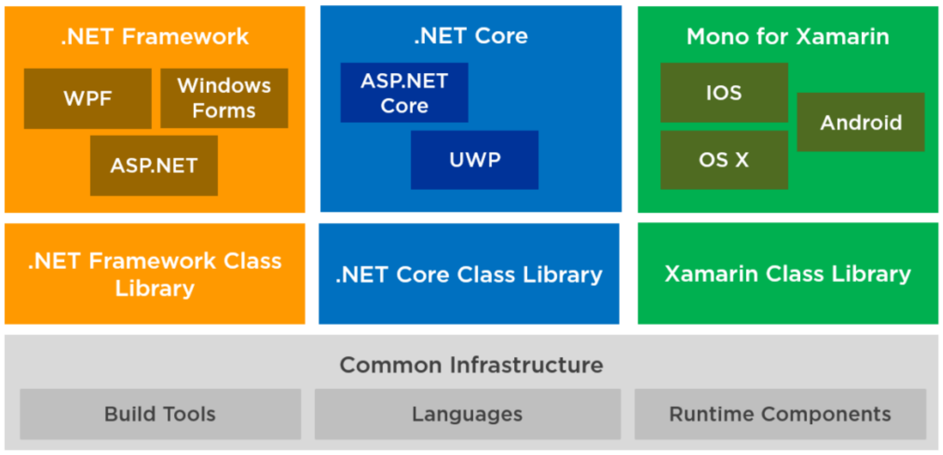
1. **C#** is simple, powerful, type-safe, and object-oriented, while retaining the expressiveness and elegance of C-style languages.
2. **VB.NET** is an easy language to learn that you use to build a variety of apps that run on .NET. Among the .NET languages, the syntax of VB is the closest to ordinary human language, often making it easier for people new to software development.
3. **F#** is a cross-platform, functional-first programming language that also supports traditional object-oriented and imperative programming. It's data-oriented, where code involves transforming data with functions.

Development Tools

1. Visual Studio.NET.
2. VS Code.

Following are **primary .NET implementations** that Microsoft actively develops and maintains:

1. **.NET Framework** is the original implementation of .NET. It supports running websites, services, desktop apps, and more on Windows.
2. **.NET** is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS. .NET is open source on GitHub. .NET was previously called **.NET Core.**
3. **Xamarin/Mono** is a .NET implementation for running apps on all the major mobile operating systems, including iOS and Android.



Each implementation of .NET includes the following components:

- **One or more runtimes**. Examples: CLR for .NET Framework, CoreCLR and CoreRT for .NET Core.
- A class library that implements the **.NET Standard** and may **implement additional APIs**. Examples: .NET Framework Base Class Library, .NET Core Base Class Library.
- Optionally, one or more **application frameworks**. Examples: ASP.NET, Windows Forms, and Windows Presentation Foundation (WPF) are included in the .NET Framework.
- Optionally, **development tools**. Some development tools are shared among multiple implementations.

About MS.NET Framework

The first .NET implementation that has existed since 2002.

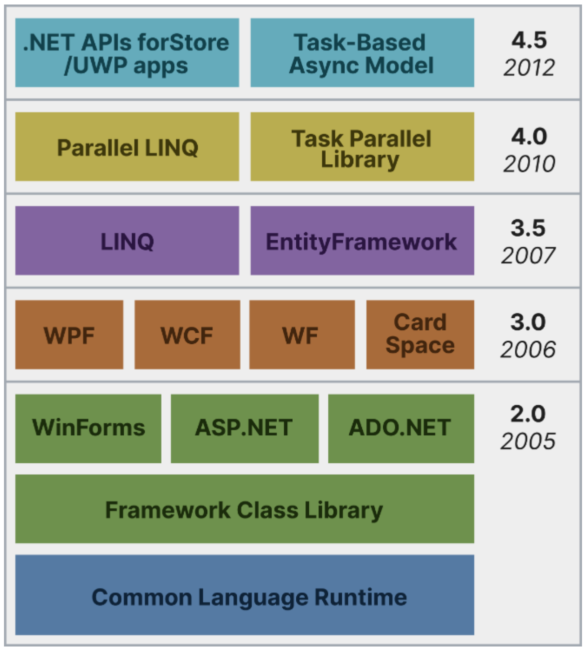
It's best optimized for developing **Windows** desktop applications using WinForms and WPF.

It supports the largest set of APIs, you're much less likely to have to worry about whether you can easily accomplish some complex task.

Version History of .NET Framework

Version number	CLR version	Release date	Support ended	Development tool	Replaces
1.0	1.0	2002-02-13	2009-07-14	Visual Studio .NET 2002	N/A
1.1	1.1	2003-04-24	2015-06-14	Visual Studio .NET 2003	1.0
2.0	2.0	2005-11-07	2011-07-12	Visual Studio 2005	N/A
3.0	2.0	2006-11-06	2011-07-12	Expression Blend	2.0
3.5	2.0	2007-11-19	N/A	Visual Studio 2008	2.0, 3.0
4.0	4	2010-04-12	2016-01-12	Visual Studio 2010	N/A

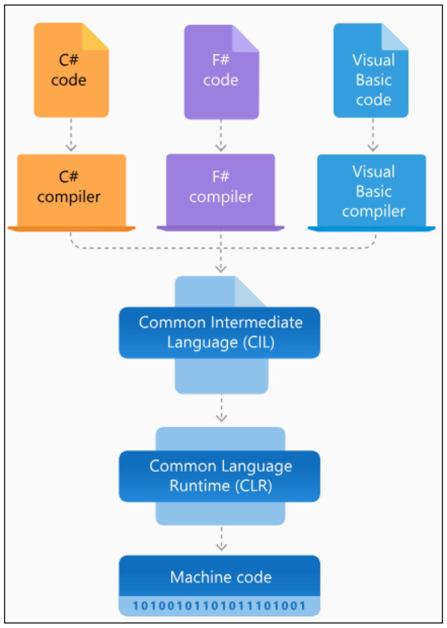
4.5	4	2012-08-15	2016-01-12	Visual Studio 2012	4.0
4.5.1	4	2013-10-17	2016-01-12	Visual Studio 2013	4.0, 4.5
4.5.2	4	2014-05-05	N/A	N/A	4.0–4.5.1
4.6	4	2015-07-20	N/A	Visual Studio 2015	4.0–4.5.2
4.6.1	4	2015-11-30	N/A	Visual Studio 2015 Update 1	4.0–4.6
4.6.2	4	2016-08-02	N/A		4.0–4.6.1
4.7	4	2017-04-05	N/A	Visual Studio 2017	4.0–4.6.2
4.7.1	4	2017-10-17	N/A	Visual Studio 2017	4.0–4.7
4.7.2	4	2018-04-30	N/A	Visual Studio 2017	4.0–4.7.1
4.8	4	2019-04-18	N/A	Visual Studio 2019	4.0–4.8

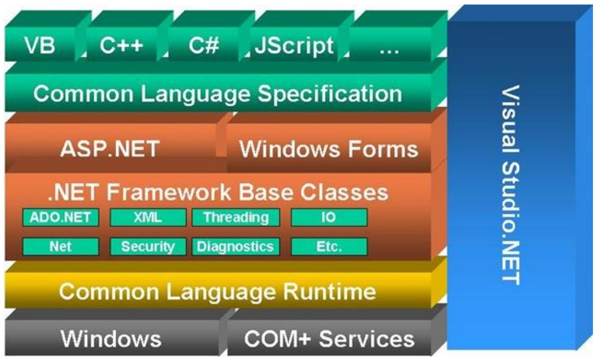


Architecture of .NET Framework

The two major components of .NET Framework are the Common Language Runtime and the .NET Framework Class Library.

1. The **Common Language Runtime (CLR)** is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.
2. The **Class Library** provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.





About Mono

- Mono, the open source and free development platform based on the .NET Framework, allows developers to build cross-platform applications with improved developer productivity.
- Was launched on July 19, 2001, after Microsoft first announced their .NET Framework in June 2000
- The project was started by enthusiasts, led by **Miguel de Icaza**, who believed that the benefits of .NET should be enjoyed on other platforms besides Windows, and that the best way to accomplish this was through an open-source effort.
- The supervision of Mono has moved from one company to another, as **de Icaza** moved:
  Ximian → Novell → Xamarin → Microsoft.

It can run ASP.NET, ADO.NET, Silverlight and Windows.Forms applications without recompilation.

The easiest way to describe what Mono currently supports is:

Everything in .NET 4.x except **WPF**, **WWF**, and with **limited WCF** and **limited ASP.NET async stack**

Mono also powers games built using the **Unity** engine.

Supported Platforms: Mono has support for both 32 and 64 bit systems on a number of architectures as well as a number of operating systems.

- Linux
- macOS, iOS, tvOS, watchOS
- Sun Solaris
- IBM AIX and i

- BSD - OpenBSD, FreeBSD, NetBSD
- Microsoft Windows
- Sony PlayStation 4
- XboxOne

In summary, if you have a .NET Framework application for Windows, and you now want your application to support Windows, macOS, Linux, BSD, and a bunch of other platforms but you want to spend a minimal amount of effort on converting your .NET application to run on many platforms, Mono may still be your best bet.

Microsoft now seems to be focusing most of their Mono efforts on the Xamarin platform and support for iOS and Android.

Xamarin

Xamarin offers two commercial products: **Xamarin.iOS** and **Xamarin.Android**. They're both built on top of *Mono*, an open-source version of the .NET Framework.

a) On iOS, Xamarin's *Ahead-of-Time* (*AOT*) Compiler compiles Xamarin.iOS applications directly to native ARM assembly code.

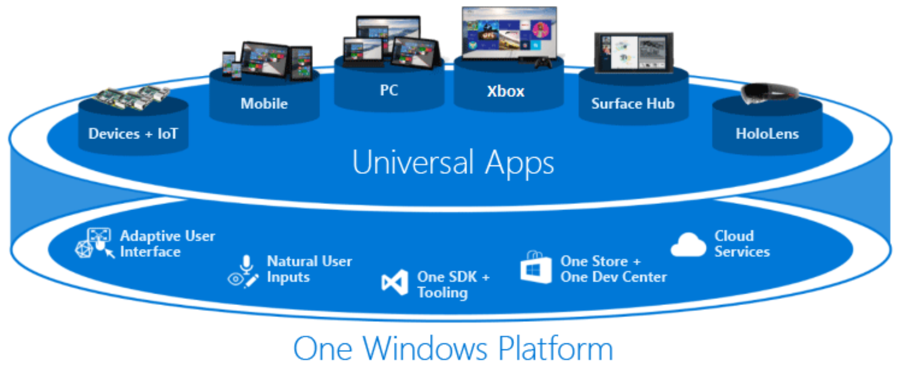
b) On Android, Xamarin's compiler compiles down to *Intermediate Language* (*IL*), which is then *Just-in-Time* (*JIT*) compiled to native assembly when the application launches.

Application Output

When Xamarin applications are compiled, the result is an Application Package, either an **.app** file in iOS, or .**apk** file in Android. These files are indistinguishable from application packages built with the platform's default IDEs and are deployable in the exact same way.

Universal Windows Platform (UWP)

- UWP is an implementation of .NET that is used for building modern, **touch-enabled** Windows applications and software for the Internet of Things (IoT).
- It's designed to unify the different types of devices that you may want to target, including PCs, tablets, phablets, phones, and even the Xbox.
- UWP applications are developed and published into **Microsoft Store** for distribution.
- Able to use device specific capabilities and adapt the UI to different device screen sizes, resolutions, and DPI.



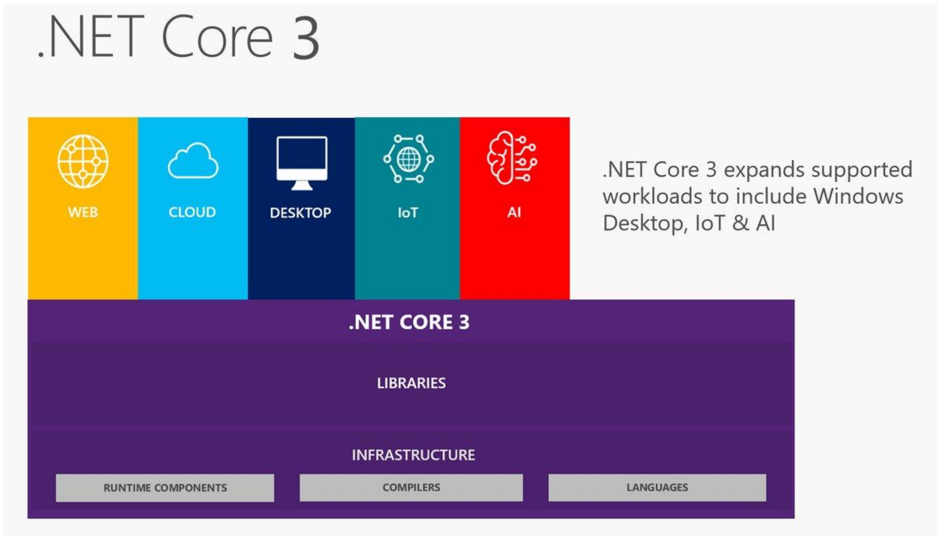
UI elements respond to the size and DPI of the screen the app is running on by adjusting their layout and scale.

What is .NET Core

- .NET Core is a **general-purpose development** platform maintained by Microsoft and the .NET community on GitHub.
- It is free and **open source, cross-platform**, supporting **Windows, macOS and Linux.** It was contributed to **.NET Foundation** by Microsoft in 2014 and is now most activate .NET Foundation project.
- On Linux, Microsoft primarily supports .NET Core running on **Red Hat Package Manger** (RPM) and **Debian** distribution families (Ubuntu / Linux Mint).
- **Consistent across architectures:** Runs your code with the same behavior on multiple CPU architectures, including x86, x64 and ARM.
- **It provides flexible deployment** and can be included in your app or installed side-by-side.
- Can be used with **Docker containers**.
- .NET Core provides compatibility with .NET Framework and Mono APIs by implementing the .NET Standard specification.

Version	Release date	Released with
.NET Core 1.0	2016-06-27	Visual Studio 2015 Update 3
.NET Core 1.1	2016-11-16	Visual Studio 2017 Version 15.0
.NET Core 2.0	2017-08-14	Visual Studio 2017 Version 15.3
.NET Core 2.1	2018-05-30	Visual Studio 2017 Version 15.7
.NET Core 2.2	2018-12-04	Visual Studio 2019 Version 16.0

.NET Core 3.0	2019-09-23	Visual Studio 2019 Version 16.3
.NET Core 3.1	2019-12-03	Visual Studio 2019 Version 16.4



Use .NET Core when:

1. There are Cross platform needs.
2. Microservices are being used.
3. Docker containers are being used.
4. Applications needs high performance and scalability.
5. If you want CLI control.
6. For Windows Forms or WPF applications.
7. For Client Side: Blazor

.NET Core didn't support:

1. ASP.NET WebForms.
2. WCF Services.
3. You need access to Windows specific API's like Windows Registry, WMI etc

Platform Support: .NET Core 3.X is supported on the following operating systems:

- Windows Client: 7, 8.1, 10 (1607+)
- Windows Server: 2012 R2 SP1+
- macOS: 10.12+
- RHEL: 6+

- Fedora: 26+
- Ubuntu: 14.04+
- Debian: 8+
- SLES: 12+
- openSUSE: 42.3+
- Alpine: 3.7+

About Blazor:

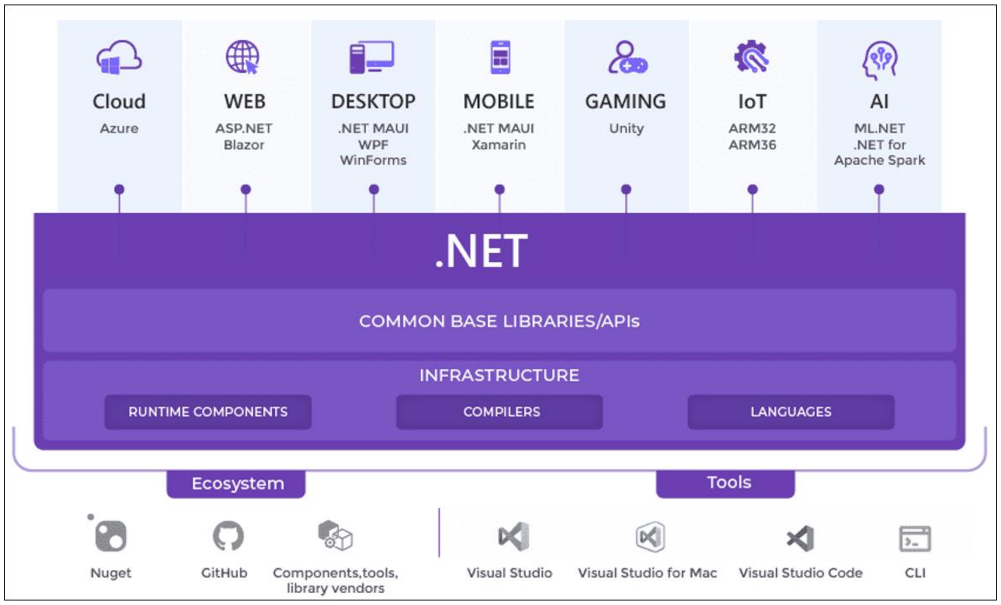
- Blazor was introduced with the release of .NET Core 3.0.
- Blazor is a web framework that allows developers to build **interactive web UIs** using C# instead of JavaScript or TypeScript.
- Blazor supports both client-side and server-side development models.
    - With client-side Blazor, the application runs on the client's web browser using WebAssembly, providing a rich user experience.
    - With server-side Blazor, the application runs on the server and communicates with the client using SignalR, which can be advantageous for scenarios where network bandwidth or client device limitations are a concern.

About .NET 5 and later versions

.NET 5 is the next major release of .NET Core following 3.1. We named this new release .NET 5 instead of .NET Core 4 for two reasons:

- Microsoft skipped version numbers 4.x to avoid confusion with .NET Framework 4.x.
- .NET 5 supports more types of apps and more platforms than .NET Core or .NET Framework.
- **.NET 5 and above** is **just one .NET** going forward, and you will be able to use it to target **Windows, Linux, macOS, iOS, Android, tvOS, watchOS and WebAssembly** and more.

Version	**Release date**	**Released with**
.NET 5	2020-11-10	Visual Studio 2019 Version 16.8
.NET 6 (LTS)	2021-11-08	Visual Studio 2022 Version 17.0
.NET 7 (STS)	2022-11-08	Visual Studio 2022 Version 17.4
.NET 8 (LTS)	**2023-11-08**	Visual Studio 2022 Version 17.8



.NET 5 includes the following improvements and new features compared to .NET Core 3.1:

- WCF now is CoreWCF (Note: better alternative option is gRPC)
- Is paired with C# 9 Language updates.
  - Records
  - Init only properties
  - Relational pattern matching
  - Top-level statements
  - Source generators
- Single file apps.
- App trimming.

There are no plans to port the following technologies from .NET Framework to .NET 5 and above.

- Web Forms
- Windows Workflow

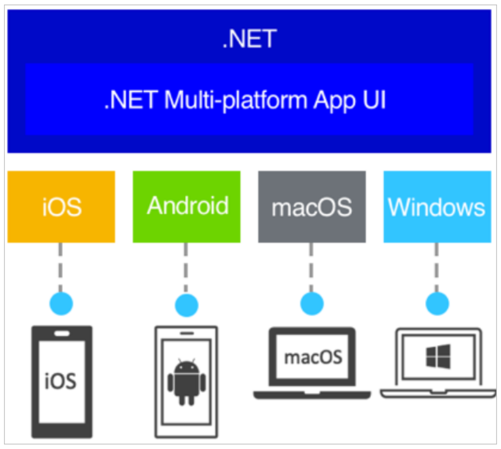
What's New in .NET 6

- Numerous performance improvements.
- Is paired with C# 10 Language updates.
  - global using directive
  - Implicit global using directives
  - File-scoped namespaces

    - Nullable reference types
- ASP.NET Core
    - Minimal APIs
- Supported with VS 2022
    - Hot reload (dotnet watch)
    - New git tooling
    - Intelligent code editing
    - Robust diagnostic and testing tools
    - Better team collaboration.
    - .NET **Multi-platform App UI** (.NET MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and XAML.

What is MAUI?

- .NET Multi-platform Application UI (.NET MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and XAML.
- Using .NET MAUI, you can develop apps that can run on Android, iOS, macOS, and Windows from a **single shared code-base**. You can add platform-specific source code and resources if necessary.
- NET MAUI is intended to unify and replace technologies like WPF, UWP or Xamarin.



What's New in .NET 7

.NET 7 is focuses on being unified, modern, simple, and *fast*. .NET 7 will be **supported for 18 months** as a standard-term support (STS) release (previously known as a *current* release).

- Is paired with C# 11 Language updates.
    - Raw string literals
    - Required Modifier
    - Anonymous Structures and Unions

- Improvements in dotnet new command.
- You can create containerized versions of your applications using dotnet publish

.NET 8

- Is paired with C# 12 Language updates.
    - global using directive
    - Implicit global using directives
    - File-scoped namespaces
    - Nullable reference types
- Improved support for Cloud Native Applications.
- Source generators enchancements.

https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8

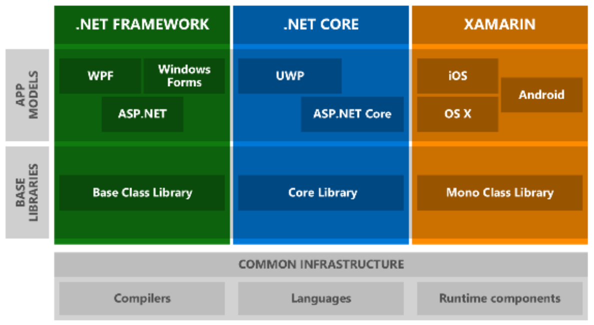
What is .NET Standard

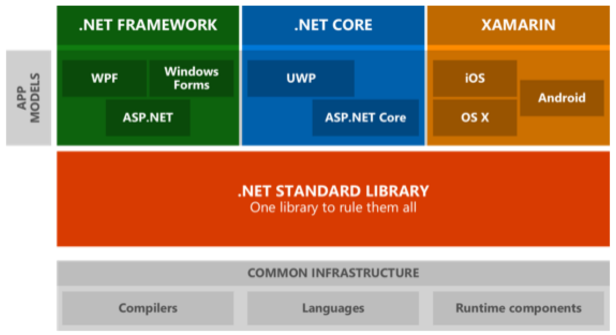
.NET Standard is a formal specification of .NET APIs that are intended to be available on **all .NET implementations**. This unifies the .NET platforms and prevents future fragmentation.

.NET Standard exposes a set of library *contracts*. .NET implementations must support each contract fully or not at all.

.NET Standard 2.0 will be implemented by .NET Framework, .NET Core, and Xamarin.

However, .NET 5 adopts a different approach to establishing uniformity, and this new approach eliminates the need for .NET Standard in many scenarios.





For developers, this means they **only have to master one base class library**. Libraries targeting .NET Standard will be able to run on all .NET platforms. And platform providers don't have to guess which APIs they need to offer in order to consume the libraries available on NuGet.

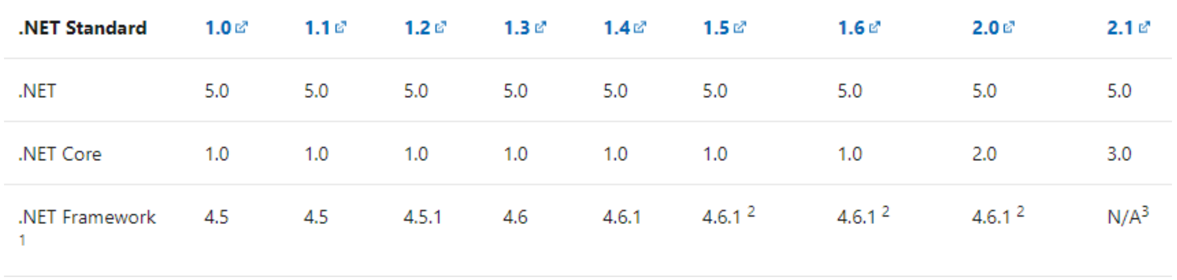
The following platforms support .NET Standard libraries:

- .NET Core

- .NET Framework
- Mono
- Xamarin.iOS, Xamarin.Mac, Xamarin.Android
- Universal Windows Platform (UWP)
- Windows
- Windows Phone
- Windows Phone Silverlight

API availability in .NET Standard is very predictable: **higher version equals more APIs**.

.NET Standard versions: The following table lists versions of .NET Standard and the platforms supported:



.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 [2]	4.6.1 [2]	4.6.1 [2]	N/A[3]

You can use this table to understand what the highest version of .NET Standard is that you can target, based on which .NET platforms you intend to run on. For instance, if you want to run on .NET Framework 4.5 and .NET Core 1.0, you can at most target .NET Standard 1.1.

When choosing a .NET Standard version, you should consider this trade-off:

1. The higher the version, the more APIs are available to you.
2. The lower the version, the more platforms implement it.

In general, it's **recommended to target the *lowest* version of .NET Standard** possible. So, after you find the highest .NET Standard version you can target, follow these steps:

1. Target the next **lower** version of .NET Standard and build your project.
2. If your project builds successfully, repeat step 1. Otherwise, retarget to the next higher version and that's the version you should use.

- The primary distribution vehicle for the .NET Standard reference assemblies is **NuGet packages.**
- .NET Standard versions are backward compatible. That means that **netstandard1.0** libraries run on **netstandard1.1** platforms and higher. However, there is no forward compatibility - lower .NET Standard platforms cannot reference higher ones. This means that **netstandard1.0** libraries cannot reference libraries targeting `netstandard1.1` or higher.

Target Framework Moniker (TFM)

TFM: Target Frameworks	**Symbols**
.NET Framework	net11, net20, net35, net40, net45, net451, net452, net46, net461, net462, net47, net471, net472, net48
.NET Standard	netstandard1_0, netstandard1_1, netstandard1_2, netstandard1_3, netstandard1_4, netstandard1_5, netstandard1_6, netstandard2_0, netstandard2_0, netstandard2_1
.NET Core	netcoreapp1.0, netcoreapp1.1, netcoreapp2.0, netcoreapp2.1, netcoreapp2.2, netcoreapp3.0, netcoreapp3.1
.NET	net5.0, net6.0, net7.0, net8.0
.NET OS Specific	**`net5.0-windows, net6.0-windows, net7.0-windows, net8.0-windows`**

Note: **For .NET 5 code, `net5.0` replaces both `netcoreapp` and `netstandard` TFMs.**

Target .NET Standard in the following scenarios:

- Use netstandard2.0 to share code between .NET Framework and all other implementations of .NET.
- Use **netstandard2.1** to share code between Mono, Xamarin, and .NET Core and .NET 5+

.NET 5+ are single products with a uniform set of capabilities and APIs that can be used for Windows desktop apps and cross-platform console apps, cloud services, and websites.

- **`net5.0/.net6.0 / .net7.0 / .net8.0`**
  This TFM is for code that runs everywhere. With a few exceptions, it includes only technologies that work cross-platform.
- **`net5.0-windows / net6.0-windows / net7.0-windows / net8.0-windows`**
  This is an example of an OS-specific TFM that add OS-specific functionality to everything that `net5.0` refers to.

When to target net5.0 and above vs. netstandard:

- For existing code that targets netstandard, there's no need to change the TFM to net5.0 or above
- .NET 5 and above implement .NET Standard 2.1 and earlier.
- The only reason to retarget from .NET Standard to .NET 5+ would be to **gain access to more runtime features, language features, or APIs**. For example, in order to use C# 9, you need to target .NET 5 or a later version.
- If you don't need to support .NET Framework, you could go with .NET Standard 2.1 or .NET 5/6/7/8. Microsoft recommend you skip .NET Standard 2.1 and go straight to .NET 5/6/7/8.

How to specify target framework:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0, net48</TargetFramework>
  </PropertyGroup>
</Project>
```