In this chapter, we’ll walk through what .NET Standard is, why it exists, and how you should think about targeting it when writing your libraries. This isn’t a high-level overview but a hands-on explanation that you might expect in a programming book, complete with code examples and practical guidance.

### **What Is .NET Standard?**

At its core, .NET Standard is a formal specification of APIs that are available across multiple .NET implementations. The idea is simple: by defining a common set of APIs, you can write libraries that work not just on one platform (like .NET Framework) but on several (including .NET Core, Mono, Xamarin, and others). This uniformity was essential at a time when the .NET ecosystem was more fragmented.

For instance, if you write a library targeting .NET Standard 2.0, you can use it in any application that supports that version—even if it’s built on .NET Framework or .NET Core. In your project file, you might see something like this:

### **The Evolution and Versioning of .NET Standard**

.NET Standard is versioned, and each new version adds more APIs. Think of it as a series of concentric circles—each higher version includes all the APIs of the previous versions plus additional ones. There are two key principles behind its versioning:

* **Additive:** Every new version is an extension of the previous one. There are no breaking changes; you just get more tools in your toolbox.
* **Immutable:** Once a version is released, it doesn’t change. This immutability means that libraries built against a specific version are guaranteed to have access to the same APIs regardless of when they’re run.

However, this approach has a trade-off. Targeting a higher version means you get access to more APIs, but you also limit your library’s compatibility to more modern platforms. Conversely, targeting a lower version broadens compatibility at the expense of available features.

### **Which Version Should You Target?**

For most libraries, targeting .NET Standard 2.0 is the sweet spot. Here’s why:

* **Broad Compatibility:** .NET Standard 2.0 is supported by all modern .NET platforms, including the .NET Framework. This makes it ideal if you need your library to run in legacy environments.
* **Rich API Surface:** It offers a significantly larger set of APIs compared to earlier versions, reducing the need to depend on multiple NuGet packages.

If you’re in a scenario where you don’t need to support the .NET Framework—say, you’re building for .NET Core or .NET 5+—you might consider .NET Standard 2.1 or even moving directly to the newer .NET versions (like .NET 6, .NET 7, or the upcoming .NET 9). The decision often comes down to a balance between compatibility and the feature set you need.

For reusable libraries intended for a wide audience, a common strategy is to multi-target both .NET Standard 2.0 and a later .NET version. This way, users on modern platforms can take advantage of new features while still supporting older platforms.

### **.NET Standard in the Context of .NET 5+**

Starting with .NET 5, Microsoft shifted the paradigm. Instead of using .NET Standard as the primary means of sharing code, .NET 5+ unifies the API surface into a single product. This approach offers several benefits:

* **Immediate Feature Availability:** New APIs and language features are available as soon as they’re implemented, without waiting for a new version of .NET Standard.
* **Simplified Target Frameworks:** Instead of juggling between “netstandard” and “netcoreapp” or “netframework,” you now target a single TFM (e.g., net9.0) or add platform-specific suffixes (like net9.0-windows).

This approach simplifies the development process by ensuring that the same set of APIs is available regardless of the workload.

### **Compatibility Mode and Practical Considerations**

When .NET Standard 2.0 was introduced, it also brought a compatibility mode for the .NET Framework. This mode allows a .NET Standard library to reference .NET Framework libraries as if they were compiled against .NET Standard. While this is extremely useful, it isn’t without its caveats—certain platform-specific libraries, such as those using Windows Presentation Foundation (WPF), might still pose challenges.

In practice, if you ever find yourself needing to bridge the gap between .NET Framework and more modern implementations, targeting .NET Standard 2.0 is usually your best bet. It minimizes the number of dependencies while providing a broad API surface.

### **Challenges with .NET Standard and How .NET 5+ Improves Things**

Despite its many benefits, .NET Standard isn’t without its issues. Here are some of the common challenges:

* **Slowness to Add New APIs:** The review process for adding APIs to .NET Standard can slow down feature adoption. You might find yourself waiting years for a new API to become available.
* **Complex Versioning:** Mapping the .NET Standard versions to various .NET implementations can be confusing, especially when dealing with legacy platforms.
* **Runtime Exceptions:** Sometimes, you might compile your code without errors only to encounter runtime exceptions if a platform doesn’t implement a specific API.

.NET 5+ addresses these problems by unifying the API surface. New features are available immediately across all supported platforms, and the simplified target framework monikers (TFMs) eliminate the need for complex version mapping. Additionally, built-in analyzers help detect potential compatibility issues at compile time, reducing the likelihood of runtime errors.

### **Final Thoughts**

.NET Standard played a pivotal role in bringing uniformity to the .NET ecosystem. While its necessity has diminished with the advent of .NET 5 and later versions, understanding its principles is still valuable—especially if you’re working on libraries that need to support a wide range of .NET implementations. In modern development, the choice often comes down to targeting .NET Standard 2.0 for broad compatibility or embracing the unified API approach of .NET 5+ for the latest features and performance improvements.

In our next chapter, we’ll dive into multi-targeting and explore how to set up your projects to support multiple frameworks simultaneously. Happy coding!