.NET Framework Overview

Understanding Core Concepts

Prepared by:
Zyad Khaled

December 25, 2025

Contents

1 Introduction to .NET Versions

.NET is a free, open-source development platform created by Microsoft for building many different types of applications. Over the years, .NET has evolved through several versions, each bringing new features and improvements.

1.1 .NET Framework

The original .NET Framework was released in 2002 and runs only on Windows. Key versions include:

- **.NET Framework 1.0/1.1** (2002-2003): The first release with basic features
- **.NET Framework 2.0** (2005): Added generics and improved performance
- **.NET Framework 3.5** (2007): Introduced LINQ (Language Integrated Query)
- **.NET Framework 4.0-4.8** (2010-2019): Added async/await, performance improvements, and modern language features

The .NET Framework is Windows-only and is now in maintenance mode, meaning it receives security updates but no new features.

1.2 .NET Core

.NET Core was introduced in 2016 as a cross-platform, open-source reimplementation of .NET. Major versions include:

- **.NET Core 1.0** (2016): First cross-platform release
- **.NET Core 2.0** (2017): Expanded API surface
- **.NET Core 3.0/3.1** (2019): Added Windows desktop support

1.3 Modern .NET (5+)

Starting with .NET 5, Microsoft unified the platform:

- **.NET 5** (2020): Unified platform (skipped version 4 to avoid confusion)
- **.NET 6** (2021): Long-term support (LTS) release
- **.NET 7** (2022): Standard-term support
- **.NET 8** (2023): LTS release with improved performance

For stability and long-term support, it's recommended to use the latest LTS version.

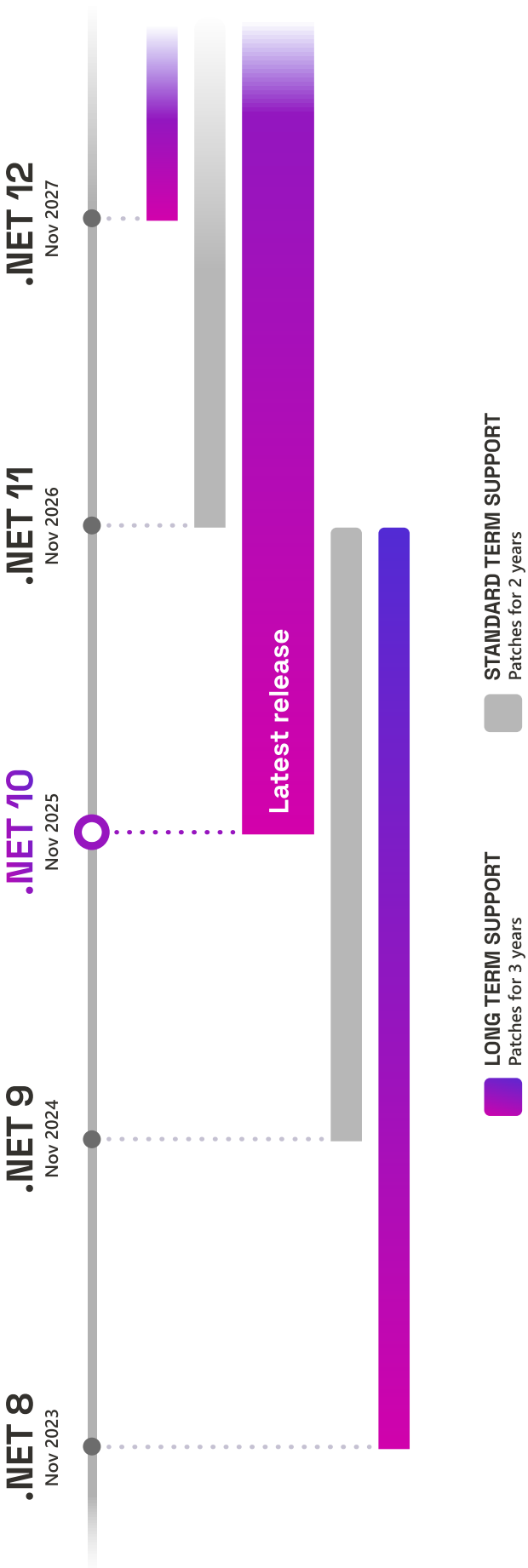

Figure 1: .NET Release Schedule and Support Timeline

2 Understanding Namespaces

A namespace is a way to organize code and prevent naming conflicts. Think of it like folders on your computer that help organize files.

2.1 What is a Namespace?

A namespace groups related classes, interfaces, and other types together. This organization helps:

- Avoid naming conflicts between different parts of code
- Make code more readable and maintainable
- Logically organize your application structure

2.2 Basic Namespace Syntax

Here's how to declare and use namespaces:

```
// Declaring a namespace
namespace MyApplication
{
    public class Calculator
    {
        public int Add(int a, int b)
        {
            return a + b;
        }
    }
}

// Using a namespace
using MyApplication;

class Program
{
    static void Main()
    {
        Calculator calc = new Calculator();
        int result = calc.Add(5, 3);
    }
}
```

2.3 Common Built-in Namespaces

.NET provides many built-in namespaces:

- `System`: Core functionality (Console, String, Array)
- `System.Collections.Generic`: Generic collections (List, Dictionary)

- `System.IO`: File and stream operations
- `System.Linq`: LINQ query operations
- `System.Net`: Network communication

2.4 Nested Namespaces

Namespaces can be nested for better organization:

```
namespace Company.Department.Project
{
    public class Employee
    {
        public string Name { get; set; }
    }
}

// Using the nested namespace
using Company.Department.Project;
```

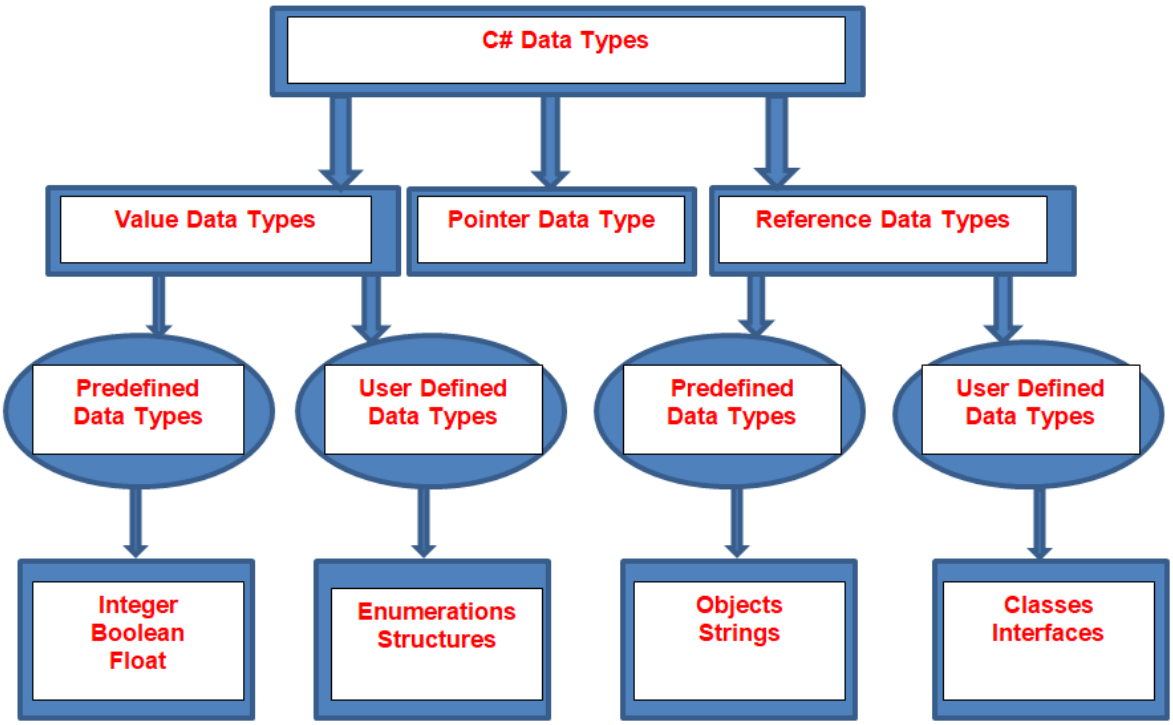

Figure 2: Namespace Organization in .NET Projects

3 .NET Core Explained

.NET Core is the modern, cross-platform version of .NET that allows you to build applications that run on Windows, macOS, and Linux.

3.1 Key Features of .NET Core

Cross-Platform: Unlike the .NET Framework, .NET Core runs on multiple operating systems, making your applications more portable.

Open Source: The entire source code is available on GitHub, allowing community contributions and transparency.

High Performance: .NET Core is optimized for speed and can handle high-load scenarios efficiently.

Modular: You can include only the packages you need, reducing application size.

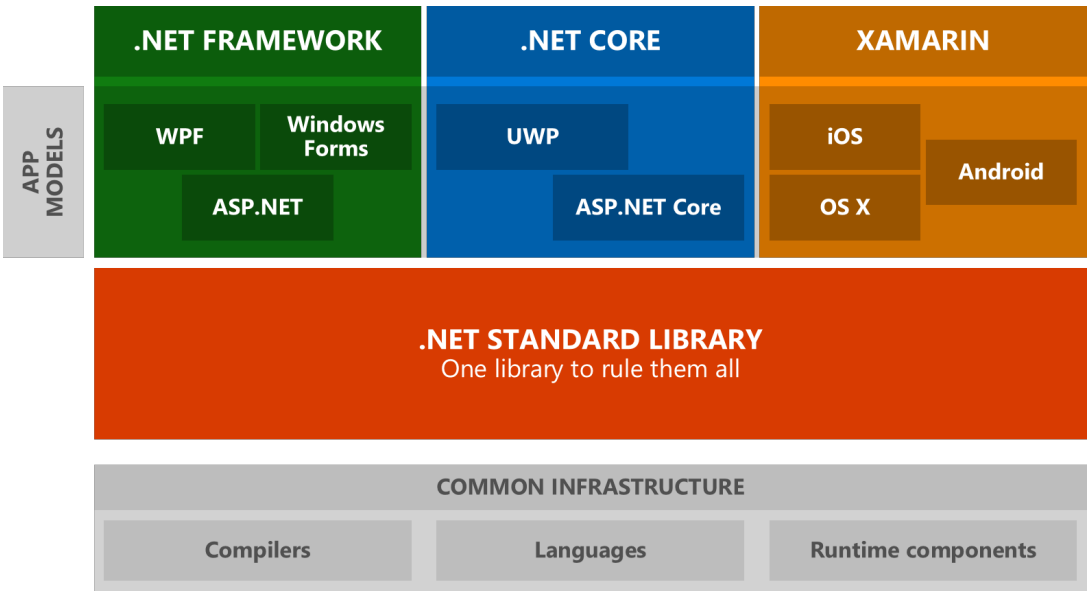

Figure 3: .NET Core Cross-Platform Architecture

3.2 Types of Applications

With .NET Core, you can build:

- **Web Applications**: Using ASP.NET Core for websites and APIs
- **Console Applications**: Command-line tools and utilities
- **Microservices**: Small, independent services
- **Cloud Applications**: Optimized for cloud platforms like Azure
- **IoT Applications**: For Internet of Things devices

3.3 .NET Core vs .NET Framework

.NET Core	.NET Framework
Cross-platform (Windows, Mac, Linux)	Windows only
Open source	Partially open source
Modular, lightweight	Larger installation
Better performance	Good performance
Active development	Maintenance mode
Recommended for new projects	Used for legacy projects

3.4 Getting Started with .NET Core

To start developing with .NET Core:

1. Download and install the .NET SDK from Microsoft's website
2. Use an IDE like Visual Studio, Visual Studio Code, or JetBrains Rider
3. Create your first application using the command line:

```
dotnet new console -n MyFirstApp
cd MyFirstApp
dotnet run
```

4 Understanding Solutions

A Solution in .NET is a container that helps organize one or more related projects. Think of it as a workspace that keeps everything together.

4.1 What is a Solution?

A Solution file (with .sln extension) acts as a container for:

- Multiple projects that work together
- Build configurations
- Project dependencies
- Shared settings

4.2 Solution Structure

A typical solution might look like this:

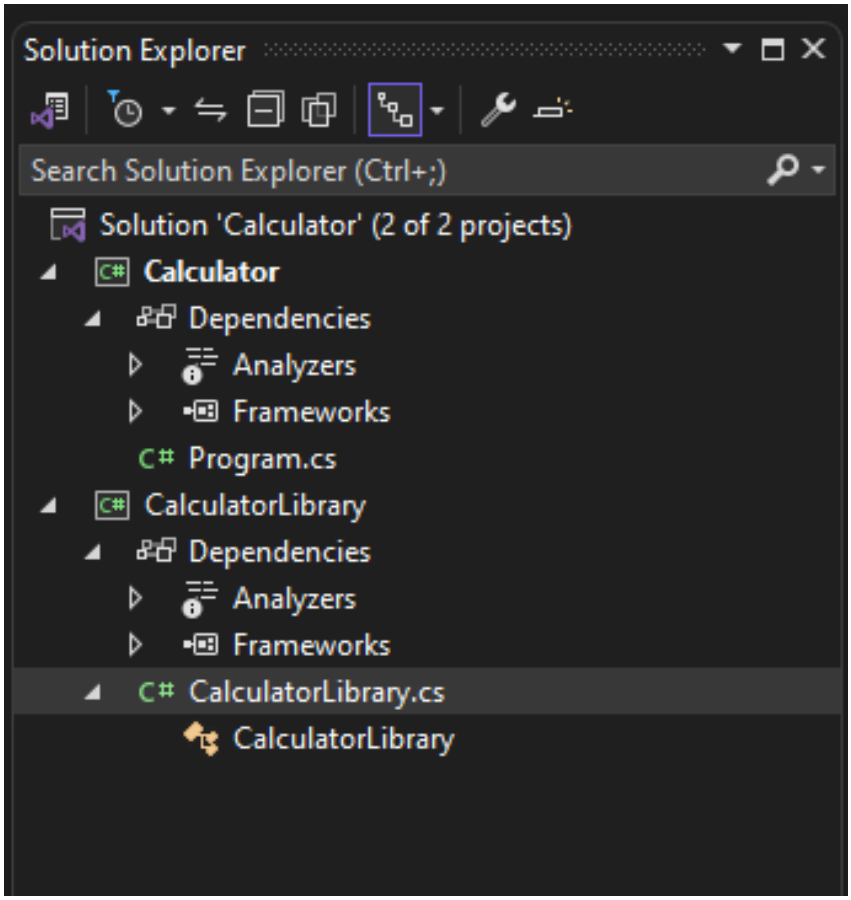

Figure 4: Solution Explorer showing multiple projects in Visual Studio

4.3 Why Use Solutions?

Organization: Keep related projects together in one place.

Dependency Management: Projects within a solution can reference each other easily.

Build Management: Build multiple projects with a single command.

Team Collaboration: Makes it easier for teams to work on different parts of the same application.

4.4 Creating a Solution

You can create a solution using Visual Studio or the command line:

```
# Create a new solution
dotnet new sln -n MySolution

# Create projects
dotnet new console -n MyApp
dotnet new classlib -n MyLibrary

# Add projects to solution
dotnet sln add MyApp/MyApp.csproj
dotnet sln add MyLibrary/MyLibrary.csproj

# Add project reference
dotnet add MyApp/MyApp.csproj reference MyLibrary/MyLibrary.csproj
```

4.5 Project Types in a Solution

Common project types include:

- **Console Application**: Command-line programs
- **Class Library**: Reusable code libraries
- **Web Application**: ASP.NET Core websites
- **Test Project**: Unit and integration tests
- **WPF/WinForms**: Desktop applications (Windows)

4.6 Best Practices

When working with solutions:

- Keep related projects together in one solution
- Use meaningful names for projects and solutions
- Separate concerns: web, business logic, data access in different projects
- Include a test project for quality assurance
- Document dependencies between projects