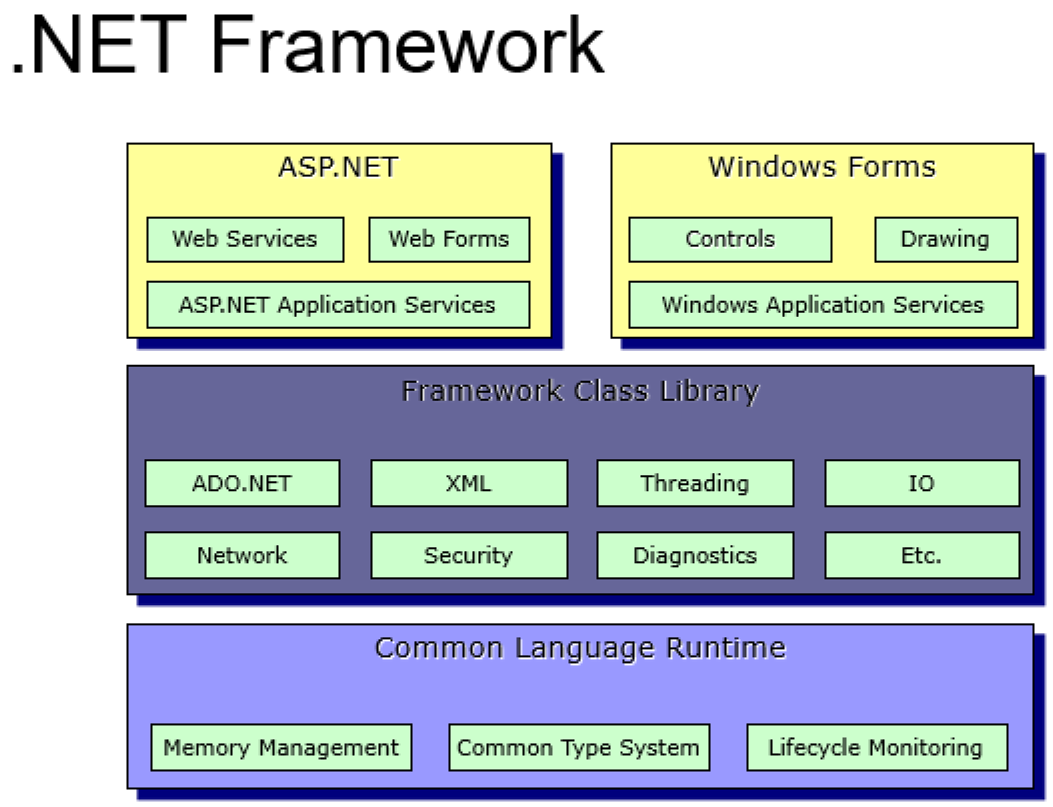
The .NET Framework is a software development platform that was introduced by Microsoft.

For the windows platform

In February 2002, Microsoft launched the first version of the .NET Framework 1.0

Used to develop form based, console-based, mobile and web-based application or services.

.NET Framework supports more than 60 programming languages



An assembly in .NET is a self-contained unit that contains compiled code, metadata, and resources.

It is the building block of a .NET application and can be a DLL (Dynamic Link Library) or an EXE (executable) file.

An assembly provides versioning, security, and deployment benefits, allowing for easy sharing and reuse of code.

It encapsulates multiple classes, types, and resources, providing a logical and organized unit of functionality.

Assemblies are loaded and executed by the .NET runtime, providing the foundation for running .NET applications.

In .NET, an assembly is a self-contained unit that contains compiled code, metadata, and resources. The main components of an assembly are:

MSIL (Microsoft Intermediate Language): This is the compiled bytecode representation of the code within the assembly. It is a platform-independent representation that can be executed by the Common Language Runtime (CLR).

Metadata: Metadata contains information about the types, members, and references within the assembly. It includes details such as type names, method signatures, versioning information, and security permissions. Metadata facilitates type resolution and runtime reflection.

Resources: An assembly can contain various resources such as images, icons, localization files, configuration files, and other non-executable data. These resources can be embedded within the assembly or referenced externally.

Manifest: The assembly manifest is a metadata file that contains information about the assembly itself. It includes details such as the assembly's name, version, culture, strong name, referenced assemblies, and entry points.

Type Definitions: An assembly can define multiple types, including classes, interfaces, structures, enums, and delegates. These type definitions encapsulate the behavior and data associated with the classes and other components within the assembly.

Collectively, these components form an assembly in .NET and provide the necessary information and resources for the execution and consumption of code within the .NET framework.

In .NET, there are two main types of assemblies:

Private Assemblies: Private assemblies are local to a specific application and are typically stored in the application's directory. They are used for encapsulating and organizing code and resources specific to that application.

Shared Assemblies: Shared assemblies, also known as strong-named assemblies, are designed to be shared among multiple applications. They have a unique strong name and are stored in the Global

Assembly Cache (GAC), a centralized repository. Shared assemblies promote code reuse and enable versioning and deployment control.

.NET Framework 4.8.1

.NET Framework 4.8

.NET Framework 4.7.2

.NET Framework 4.7.1

.NET Framework 4.7

.NET Frame;work 4.6.2

.NET Framework 4.6.1

.NET Framework 4.6

.NET Framework 4.5.2

.NET Framework 4.5.1

.NET Framework 4.5

.NET Framework 4

.NET Framework 3.5

.NET Framework 3.0

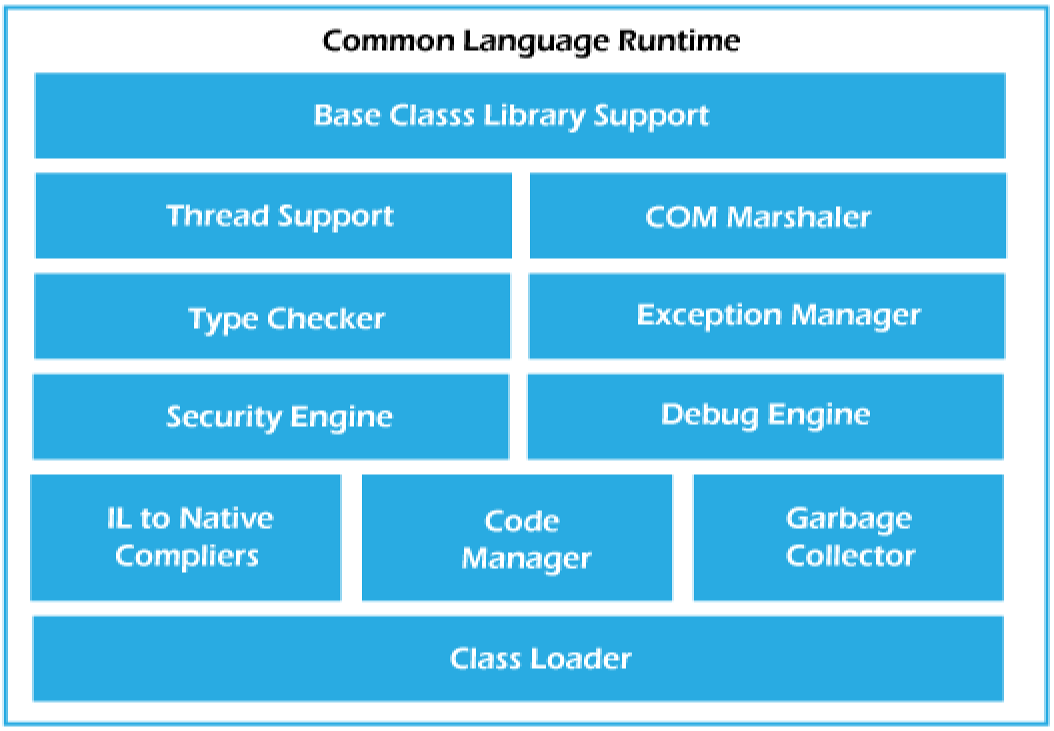
.NET Framework 2.0

.NET Framework 1.1

.NET Framework 1.0

Reference

https://learn.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies



Components of the Common Language runtime/Architecture of CLR

CLR provides a runtime environment that manages memory, executes code, handles exceptions, enforces security and type safety, manages resources, performs JIT compilation, and handles assembly loading and versioning for .NET applications. Its primary goal is to provide a robust and efficient execution environment for running managed code within the .NET framework.

Class loader: In the CLR (Common Language Runtime), the class loader is responsible for loading and initializing .NET classes and types.

It locates and loads the required assembly files containing the classes needed by a .NET application.

It performs type verification, ensuring that the loaded types adhere to the rules of type safety.

The class loader also handles assembly resolution, managing dependencies and resolving references to other assemblies.

It plays a crucial role in the overall execution and management of .NET applications within the CLR environment.

Code Manager: In the CLR (Common Language Runtime), the code manager is responsible for managing the execution of .NET code.

It performs tasks such as just-in-time (JIT) compilation, converting Intermediate Language (IL) code into machine code at runtime.

The code manager also handles memory allocation for the compiled code and manages its execution within the runtime environment.

It ensures proper execution order and handles exception handling and debugging functionalities.

Overall, the code manager plays a vital role in executing and managing .NET code within the CLR.

JIT : JIT (Just-In-Time) compilation in the CLR (Common Language Runtime) is a process of dynamically converting Intermediate Language (IL) code into native machine code at runtime.

JIT compilation occurs when the IL code is about to be executed, optimizing it for the specific hardware and operating system.

This dynamic compilation improves performance by eliminating the need for interpretation or pre-compilation of the entire codebase.

JIT compilation also allows for runtime optimizations based on the current execution context and enables efficient memory management.

Overall, JIT compilation is a key component of the CLR, translating IL code into machine code on-the-fly for efficient execution.

Garbage Collector: The garbage collector in the CLR (Common Language Runtime) is responsible for automatic memory management in .NET applications.

It identifies and frees up memory that is no longer in use, reclaiming resources and preventing memory leaks.

The garbage collector tracks object references and determines which objects are still reachable and which can be safely deleted.

It runs in the background, periodically performing garbage collection to release unused memory and improve application performance.

The garbage collector reduces the burden of manual memory management, making .NET programming more efficient and less prone to memory-related errors.

Exception Manager. The exception manager in the CLR (Common Language Runtime) is responsible for handling and managing exceptions that occur during the execution of .NET applications.

It provides a structured mechanism for catching and handling exceptions, preventing the application from crashing.

The exception manager identifies and propagates exceptions to the appropriate exception handling code.

It supports try-catch blocks and other exception handling constructs to ensure proper error handling and recovery.

The exception manager plays a crucial role in maintaining application stability and facilitating robust error handling in the CLR environment.

Error and Exception

The type checker in the CLR (Common Language Runtime) is responsible for enforcing type safety in .NET applications.

It verifies that operations performed on variables and objects are valid and conform to the rules of the specified types.

The type checker ensures that type mismatches and unsafe operations are caught and prevented during compilation and execution.

It plays a crucial role in preventing type-related errors, improving application reliability and security.

By enforcing type safety, the type checker enhances the overall integrity and correctness of .NET programs within the CLR.

Thread support in the CLR (Common Language Runtime) refers to the ability to create and manage concurrent threads within a .NET application.

It provides a framework for creating and synchronizing threads, allowing for parallel and asynchronous execution of code.

The thread support in CLR includes features such as thread creation, thread synchronization mechanisms (e.g., locks, monitors), and thread pooling for efficient resource management.

It enables developers to write multi-threaded applications that can take advantage of the available hardware resources and improve performance.

Thread support in the CLR facilitates concurrent programming and helps build responsive and scalable applications.

The security engine in the CLR (Common Language Runtime) is responsible for enforcing security policies and providing a secure execution environment for .NET applications.

It enforces code access permissions, restricting access to sensitive resources based on the defined security policies.

The security engine also supports role-based security, allowing fine-grained control over user permissions and access rights.

It provides mechanisms for code and data verification, ensuring that code operates within a trusted and secure context.

Overall, the security engine enhances the security of .NET applications, protecting against unauthorized access and malicious code execution.

The debug engine in the CLR (Common Language Runtime) is responsible for supporting debugging functionalities during the development and testing of .NET applications.

It provides features such as breakpoints, stepping through code, inspecting variables, and capturing runtime information for debugging purposes.

The debug engine enables developers to track and diagnose issues within their code, facilitating the identification and resolution of bugs.

It works in conjunction with debugging tools, such as Visual Studio, to provide a comprehensive debugging experience for .NET applications.

Overall, the debug engine plays a crucial role in the software development process by aiding in the detection and resolution of programming errors.

COM (Component Object Model) is a binary interface standard in Windows that allows software components to communicate and interact with each other.

It provides a mechanism for creating reusable software components that can be used across different languages and platforms.

COM components can be used in a variety of applications, ranging from desktop software to web services.

COM supports features such as object instantiation, method invocation, and interface-based communication.

It offers a robust and scalable architecture for building modular and extensible software systems.

The COM marshaller in the CLR (Common Language Runtime) is responsible for facilitating communication between .NET code and COM (Component Object Model) objects.

It handles the conversion and marshalling of data between the .NET types and COM interfaces, allowing seamless interoperability.

The COM marshaller ensures that method calls, parameter passing, and data transfer between .NET and COM components are correctly translated.

It provides a bridge between the different object models, enabling .NET applications to interact with COM components and vice versa.

Overall, the COM marshaller simplifies the integration of .NET and COM technologies, enabling cross-platform and cross-language communication.

The Base Class Library (BCL) is a fundamental component of the .NET framework that provides a set of pre-built classes and functionality for common programming tasks.

It includes essential types such as collections, input/output operations, networking, threading, and more.

The BCL provides a consistent and standardized set of classes that can be used across different .NET languages and platforms.

Developers can leverage the BCL to accelerate development, enhance productivity, and ensure code reuse.

It forms the foundation of the .NET framework, offering a rich set of functionality for building a wide range of applications.

CTS (Common Type System) is a component of the .NET framework that defines a standard set of data types and rules for type interoperability among different programming languages.

It ensures that objects written in different .NET languages can seamlessly interact with each other by enforcing a common type structure and behavior.

CTS provides a unified type system, allowing languages to share and exchange data without losing type information.

It establishes guidelines for type definition, inheritance, method signatures, and other aspects of type representation in the .NET framework.

CTS enables language integration, code interoperability, and facilitates the creation of multi-language applications within the .NET ecosystem.

CLS (Common Language Specification) is a subset of the Common Type System (CTS) in the .NET framework that defines a set of rules and guidelines for language interoperability.

It ensures that .NET languages can interoperate seamlessly by specifying a common set of language features and restrictions.

CLS-compliant code adheres to these guidelines, allowing it to be used and accessed by other CLS-compliant languages.

By following CLS, developers can write code that can be easily reused and accessed by a wider range of .NET languages.

CLS promotes language interoperability and helps create components that can be seamlessly integrated into different .NET applications.

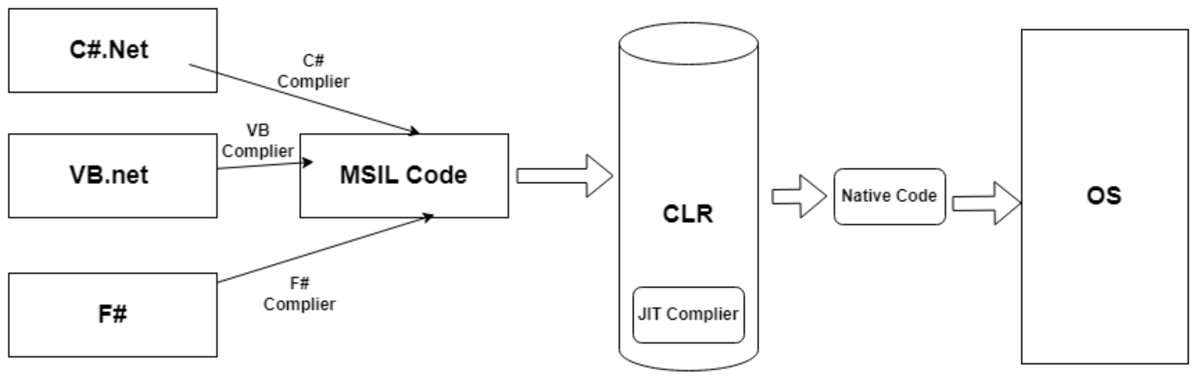
App Domain management in the .NET framework involves creating and managing isolated environments for executing .NET applications.

An app domain provides a secure and isolated boundary for running code, preventing interference between different applications.

It allows for loading and unloading of assemblies, managing memory, and enforcing security boundaries within the application.

App domain management provides a mechanism for isolating errors, managing resources, and enhancing application stability and security.

It enables the execution of multiple applications within a single process, providing flexibility and control over application lifecycles.



.Net Execution Process

Language compilers are tools that convert high-level programming code, written in languages such as C#, VB.NET, or F#, into MSIL (Microsoft Intermediate Language) code. MSIL is a low-level, platform-agnostic code that the .NET execution engine can understand and execute. Let's break it down into simpler terms:

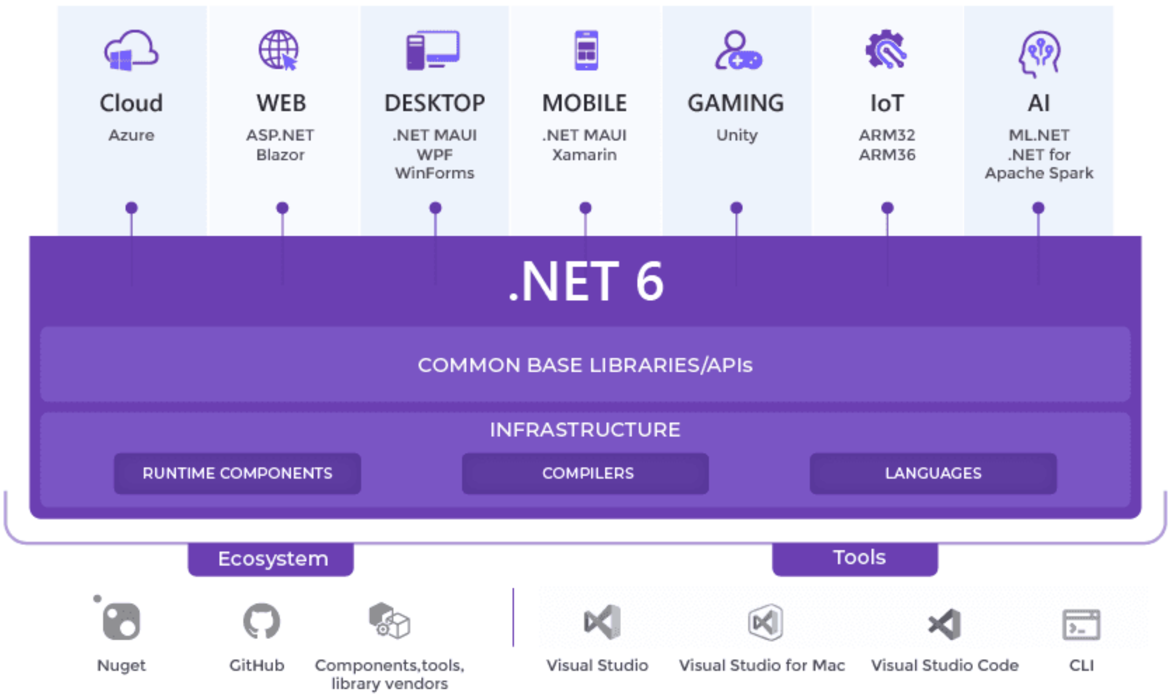
When you write code in a high-level language like C#, you use syntax and structures that are more human-readable and closer to natural language. However, computers cannot directly execute this code. That's where compilers come in.

A compiler is a special program that takes your high-level code as input and translates it into a lower-level code that the computer can execute. In the case of .NET, this lower-level code is called MSIL.

MSIL is a type of bytecode, which is an intermediate representation of the code. It's similar to machine code, but it's not specific to any particular hardware or operating system. Instead, it's designed to be executed by the .NET Common Language Runtime (CLR), which is the execution engine for .NET applications.

The CLR, or .NET runtime, understands and executes MSIL code. When you run a .NET application, the CLR loads the compiled MSIL code and performs just-in-time (JIT) compilation, which converts the MSIL into machine code that is specific to the underlying hardware. This machine code is then executed by the processor.

- Language compilers are responsible for translating high-level code into MSIL, which is a low-level code that the .NET runtime can execute.
- The CLR then takes care of converting the MSIL into machine code and running the application on the computer's hardware.


Cloud
Azure
WEB
ASP.NET
Blazor
DESKTOP
.NET MAUI
WPF
WinForms
MOBILE
.NET MAUI
Xamarin
GAMING
Unity
IoT
ARM32
ARM36
AI
ML.NET
.NET for Apache Spark
.NET 6
COMMON BASE LIBRARIES/APIs
INFRASTRUCTURE
RUNTIME COMPONENTS
COMPILERS
LANGUAGES
Ecosystem
Tools
Nuget
GitHub
Components,tools, library vendors
Visual Studio
Visual Studio for Mac
Visual Studio Code
CLI