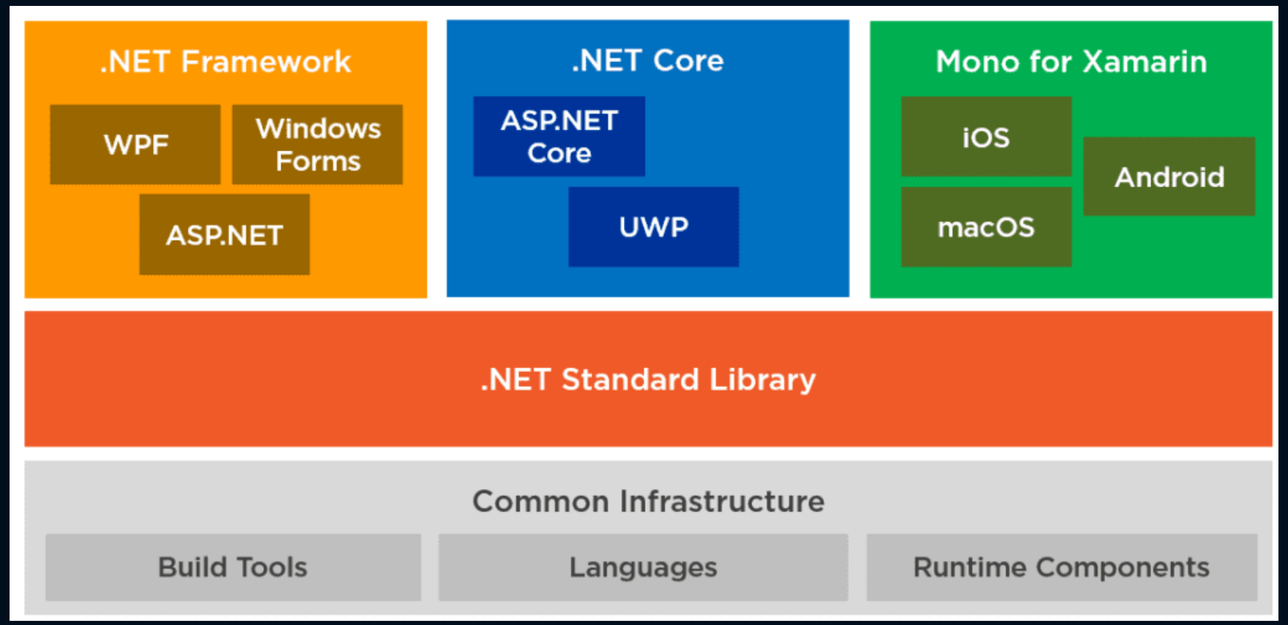
.NET Ecosystem



.NET Ecosystem explained

.NET Runtimes
.NET Framework: Designed for Windows desktop applications, web applications, and services.
.NET Core: A cross-platform framework that supports desktop, web, and cloud-based applications.
Mono is a Cross-platform implementation for .NET Framework.

.NET APIs
.NET ecosystem provides a rich set of APIs (Application Programming Interfaces) that developers can use to build applications. (e.g. Data Access, Networking, Security, more)

.NET applications

.NET is a development platform created by Microsoft which allows us to build or develop applications.

Desktop

WPF

WinForms

UWP

Web

ASP .NET

ASP .NET Core

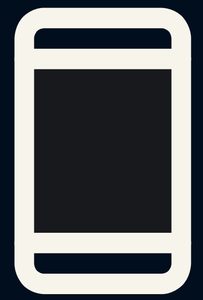
Blazor

Cloud

Azure

Games

Unity



Mobile

.NET Maui (FKA Xamarin)

AI and IoT

ML .NET

ARM32

ARM64

.NET languages

Microsoft offers 3 languages on the .NET platform: C#, F# and Visual Basic.

C#

F#

Visual Basic

.NET Technologies

There are several variants of .NET, each supporting a different type of application. The reason for the multiple variants is partly historical and partly technical.

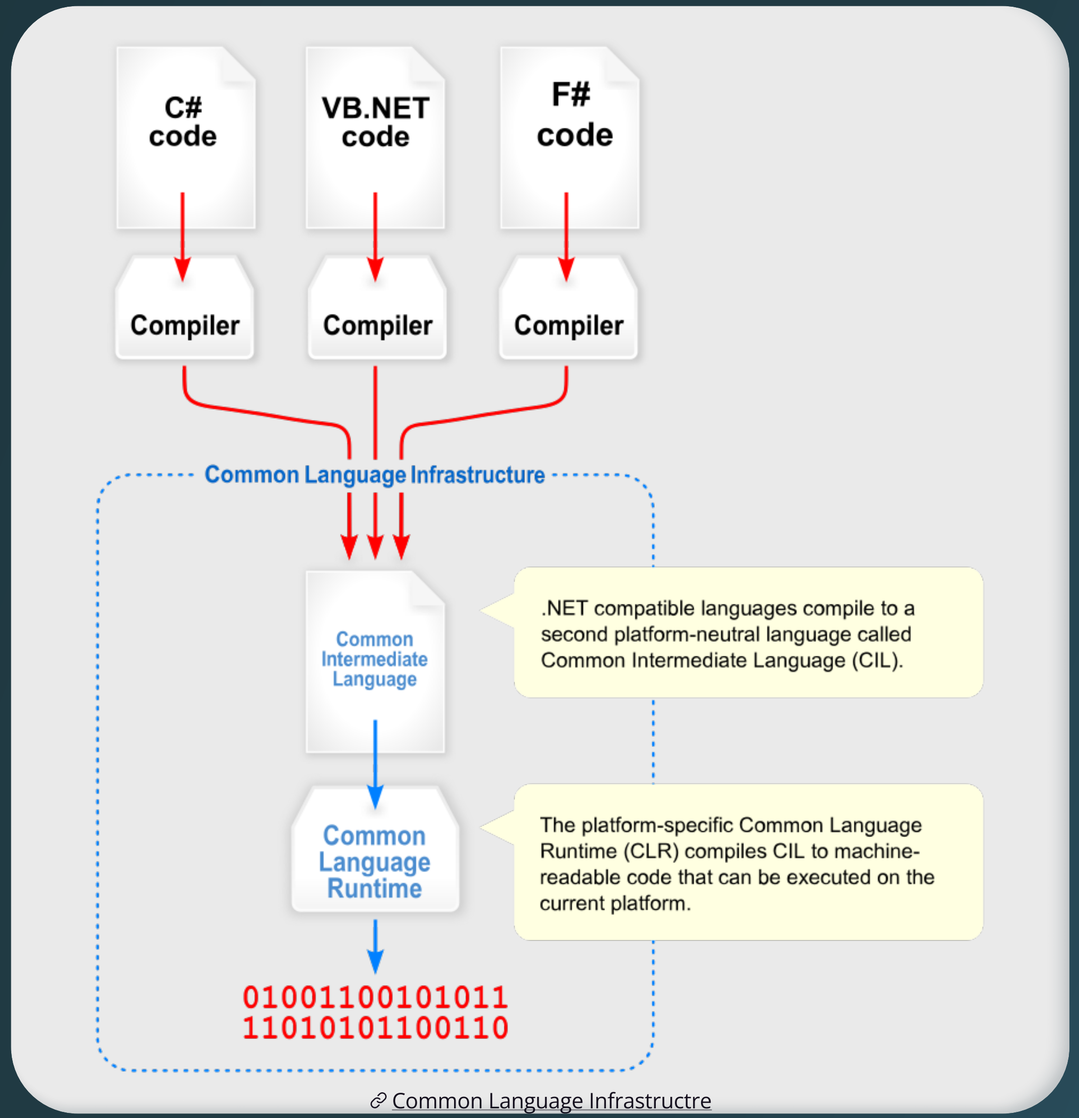
Technology	Description	Host operating systems
Modern .NET	A modern feature set, full C# 8, 9, and 10 support, used to port existing apps or create new desktop, mobile, and web apps and services	Windows, macOS, Linux, Android, iOS
.NET Framework	A legacy feature set, limited C# 8 support, no C# 9 or 10 support, used to maintain existing applications only	Windows only
Xamarin	Mobile and desktop apps only	Android, iOS, macOS

Common Language Infrastructure

It is a set of standards that define how .NET applications are compiled, run, and managed. It is the foundation of the .NET platform and provides a number of benefits, such as portability, security, and scalability.

Let's use the following analogy:

Imagine that a house is being built. The Common Language Infrastructure is like the blueprint of the house. It is a set of instructions that tell you how to build the house.



Common Language Infrastructre

.NET Core toolchain

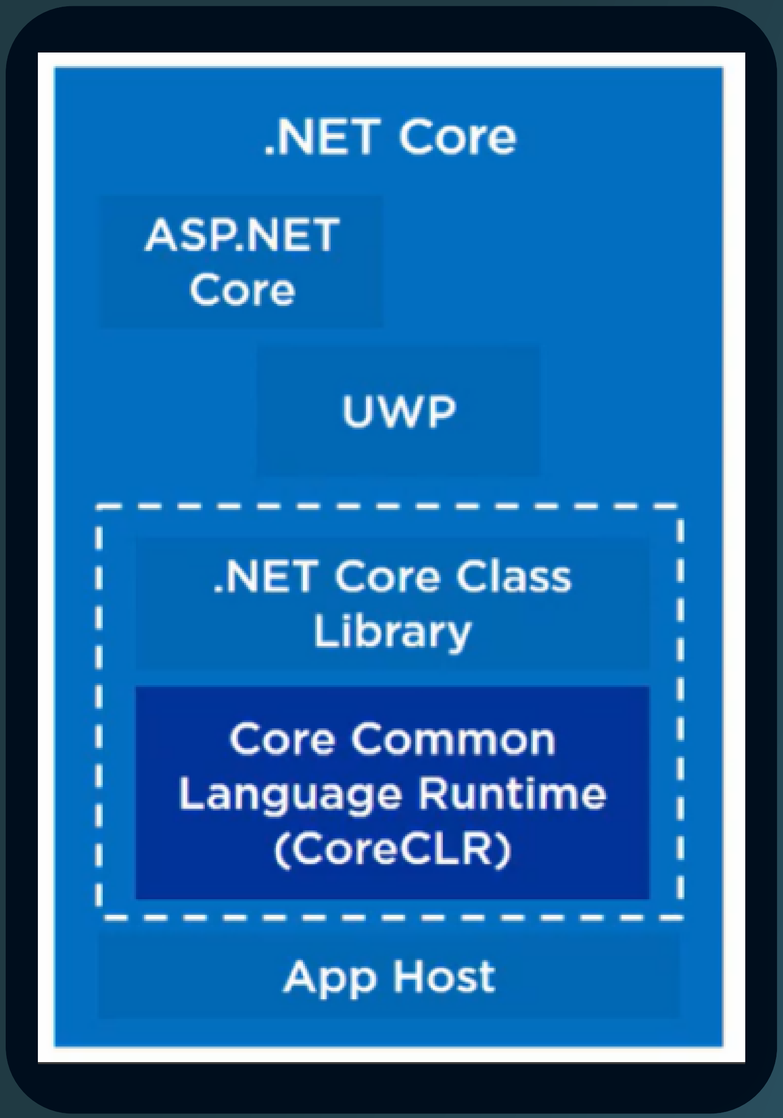
The .NET Core toolchain is a set of tools and utilities specifically designed for developing applications using the .NET Core framework. .NET Core is an open-source, cross-platform development framework that allows you to build applications that run on Windows, macOS, and Linux.

Workloads (application types)

- Console applications
- ASP.NET Core
  - MVC
  - API
- Azure (WebJobs, Cloud Services)
- Universal Windows Platform Apps

CLR

- Assembly loading, Garbage collection
- C#, VB.NET, F#



.NET Framework toolchain

The .NET framework toolchain refers to a set of tools and utilities provided by Microsoft as part of the .NET development platform. These tools assist in various stages of the software development lifecycle, including building, testing, debugging, and deploying .NET applications.

Workloads (application types)

- Console applications
- Windows Communication Foundation
- Windows Presentation Foundation
- Windows Forms
- ASP.NET
  - Forms, MVC, Web API
- Azure (WebJobs, Cloud Services)

CLR

- Run code, GC
- C#, VB.NET, F#

Windows APIs

Common Language Runtime

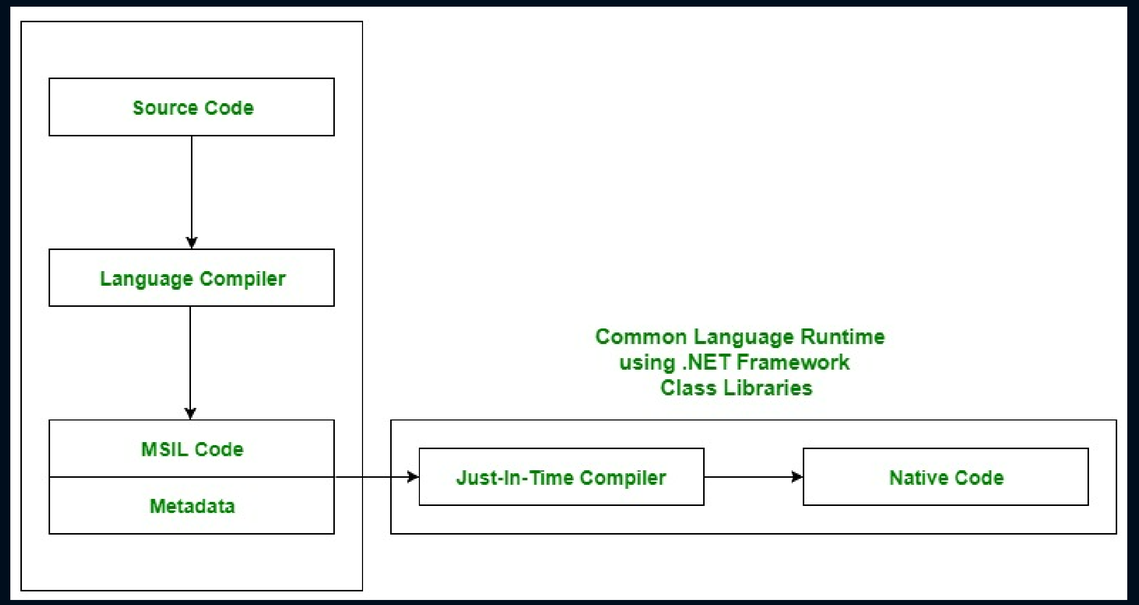
The Common Language Runtime (CLR) is a component of the Microsoft .NET Framework that manages the execution of .NET applications. It is responsible for loading and executing the code written in various .NET programming languages, including C#, VB.NET, F#, and others.

When a C# program is compiled, the resulting executable code is in an intermediate language called Common Intermediate Language (CIL) or Microsoft Intermediate Language (MSIL). This code is not machine-specific, and it can run on any platform that has the CLR installed. When the CIL code is executed, the CLR compiles it into machine code that can be executed by the processor.

Additionally, the CLR provides a framework for developing and deploying .NET applications, including a set of libraries, called the .NET Framework Class Library, which provides access to a wide range of functionality, such as input/output operations, networking, database connectivity, and user interface design.

Language Interoperability can be achieved in two ways :

1. Managed Code: The MSIL code which is managed by the CLR is known as the Managed Code. For managed code CLR provides three .NET facilities:
2. Unmanaged Code: Before .NET development, programming languages like.COM Components & Win32 API do not generate the MSIL code. So these are not managed by CLR rather managed by Operating System.



Source Code to Native Code

Common Language Runtime Architecture

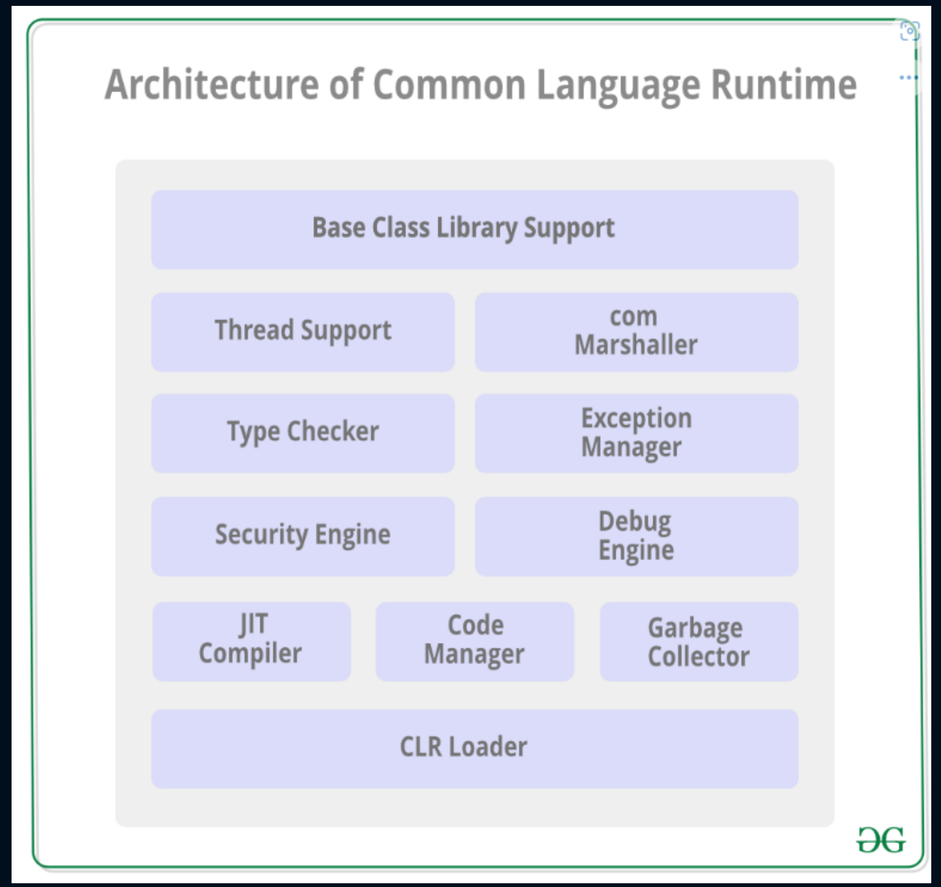
The managed execution environment is provided by giving various services such as memory management, security handling, exception handling, garbage collection, thread management, etc.

The Common Language Runtime implements the VES (Virtual Execution System) which is a run time system that provides a managed code execution environment. The VES is defined in Microsoft's implementation of the CLI (Common Language Infrastructure).

COM Marshaller: Communication with the COM (Component Object Model) component in the .NET application is provided using the COM marshaller. This provides the COM interoperability support.

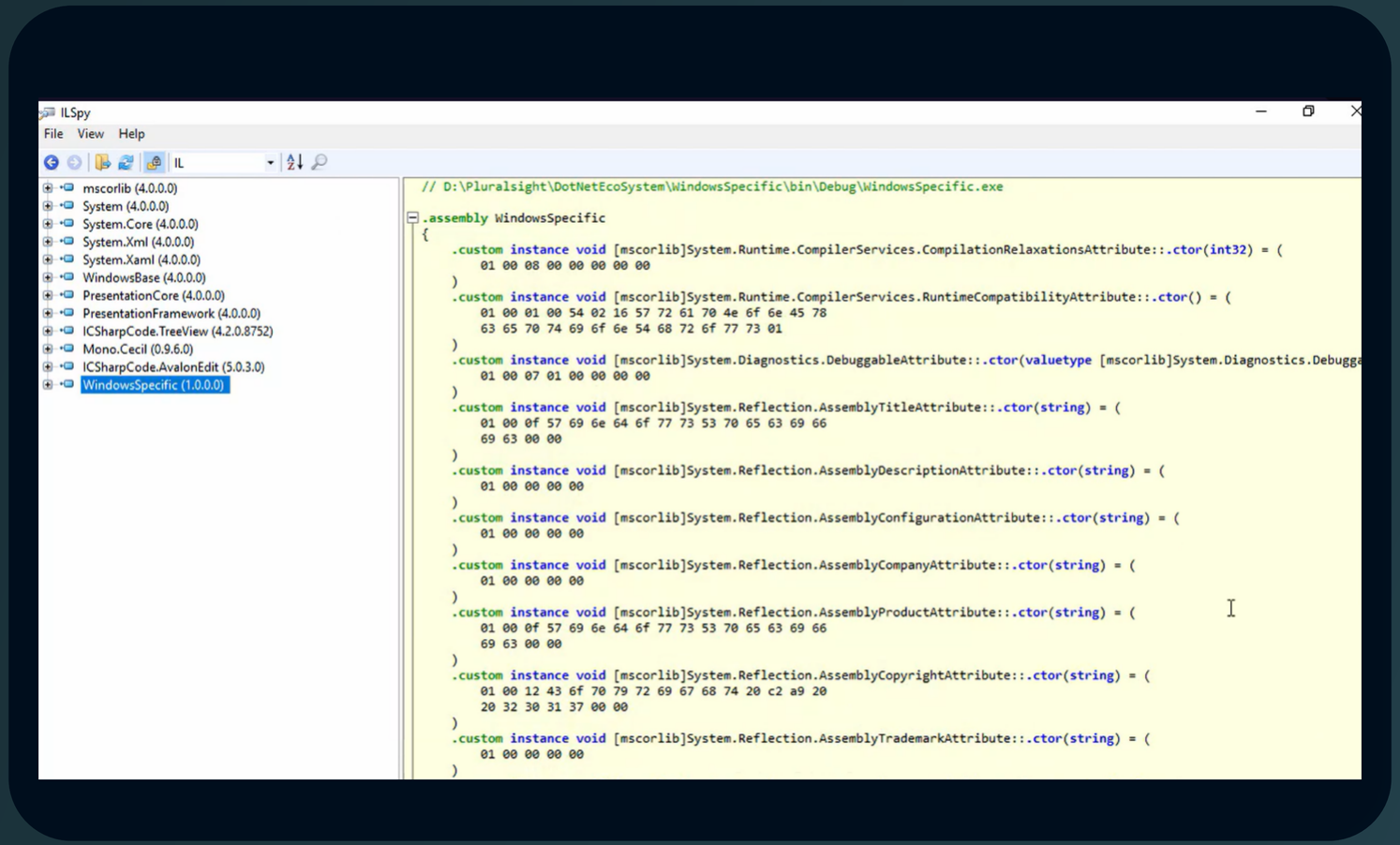
Type Checker: Type safety is provided by the type checker by using the Common Type System (CTS) and the Common Language Specification (CLS) that are provided in the CLR to verify the types that are used in an application.

More information in this post



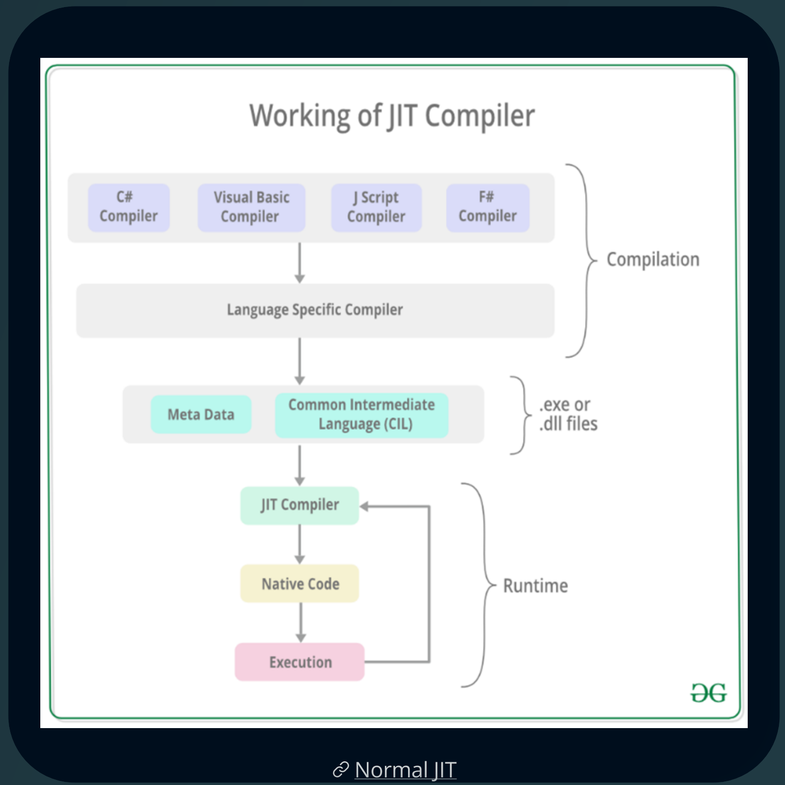
IL Code

Stands for intermediate code. For example in Microsoft VS you have support for too many languages, user have different machine configuration and different operating system whose unknown to visual studio. To avoid this problem Microsoft Creates a code that is called as IL Code at compile time.



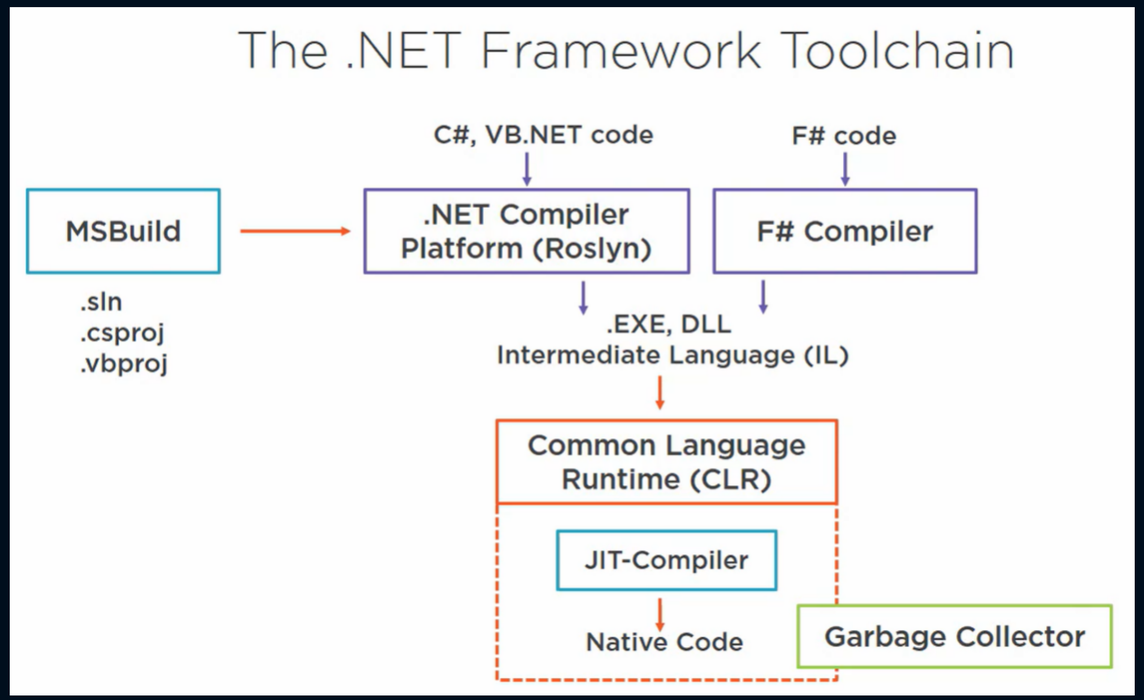
The JIT compiler

Just-In-Time compiler(JIT) is a part of Common Language Runtime (CLR) in .NET which is responsible for managing the execution of .NET programs regardless of any .NET programming language. A language-specific compiler converts the source code to the intermediate language. This intermediate language is then converted into the machine code by the Just-In-Time (JIT) compiler. This machine code is specific to the computer environment that the JIT compiler runs on.

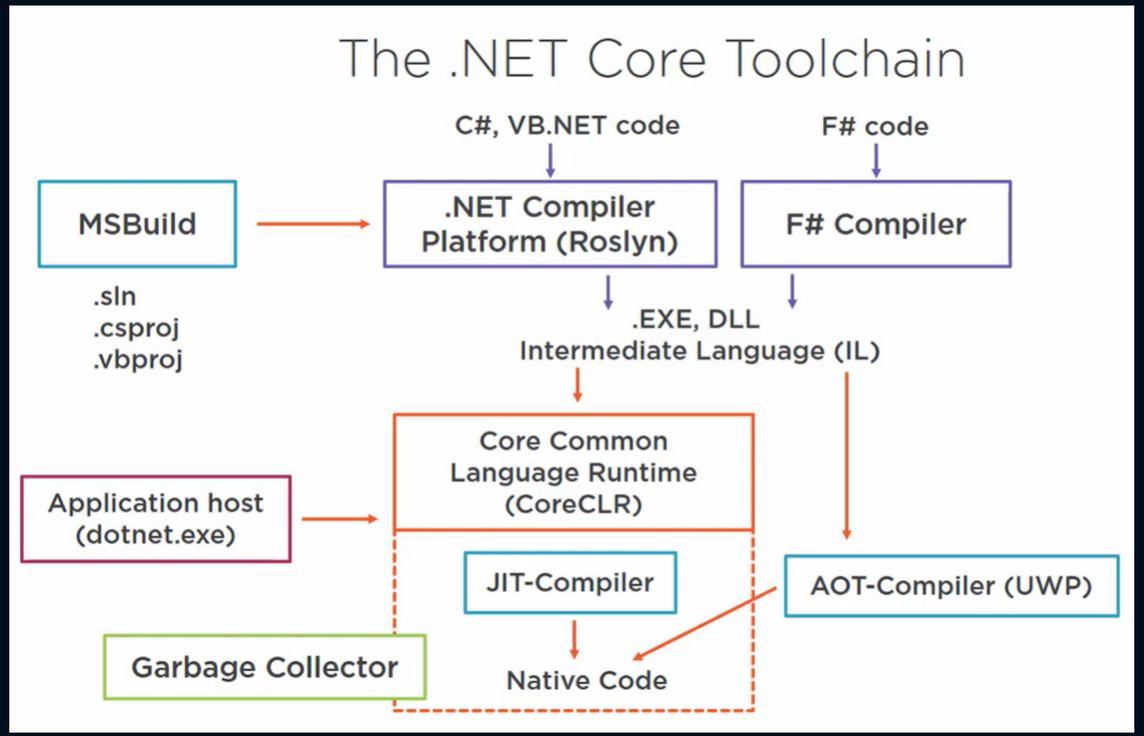


Normal JIT

Runtimes and CLR



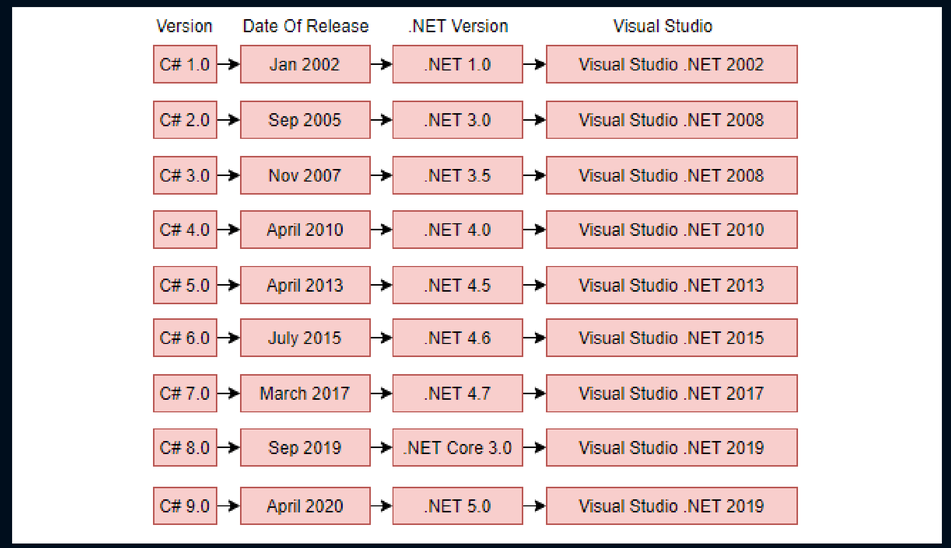
.NET Framework Toolchain to native code



.NET Core Runtime to native code

General overview in C#

Main language for .NET development, launched to the public on 2000 and based on C++, Java, Pascal.
Multiparadigm Language (Structural Programming, Imperative, OOP, FP, Event Driven)
Offers strong typing and type safety
Has memory management with automatic garbage collection
Has language constructs for asynchronous programming
Support for generics, LINQ, and more



Version	Date Of Release	.NET Version	Visual Studio
C# 1.0	Jan 2002	.NET 1.0	Visual Studio .NET 2002
C# 2.0	Sep 2005	.NET 3.0	Visual Studio .NET 2008
C# 3.0	Nov 2007	.NET 3.5	Visual Studio .NET 2008
C# 4.0	April 2010	.NET 4.0	Visual Studio .NET 2010
C# 5.0	April 2013	.NET 4.5	Visual Studio .NET 2013
C# 6.0	July 2015	.NET 4.6	Visual Studio .NET 2015
C# 7.0	March 2017	.NET 4.7	Visual Studio .NET 2017
C# 8.0	Sep 2019	.NET Core 3.0	Visual Studio .NET 2019
C# 9.0	April 2020	.NET 5.0	Visual Studio .NET 2019

C# History

Release history

.NET CLI

The .NET Command-Line Interface (CLI) is a powerful tool for developing, building, and managing .NET applications from the command line.
It allows you to create projects, restore dependencies, build, run, and publish applications across different platforms.

.NET CLI commands

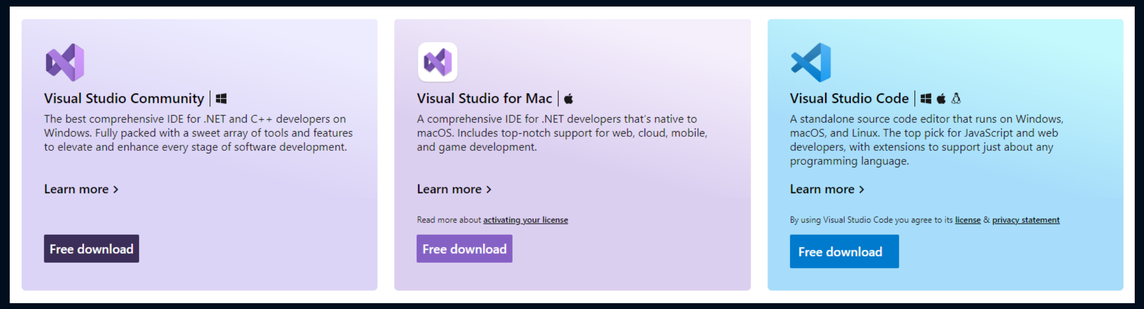
```
# Shows tha installation path of its components
dotnet --info
# All existing commands
dotnet --help
# Check the version
dotnet --version
# Adds a project to the current that its been working on
dotnet add
# Same as Build option from Visual Studio IDE
dotnet build
# Same as Clean option from Visual Studio IDE
dotnet clean
# Creates a Dotnet project
dotnet new

dotnet nuget
dotnet pack
dotnet publish
dotnet remove
# Restore all nuggets packages
dotnet restore
# Compiles the application
dotnet run
# Runs the tests
dotnet test
```

Create new console app using CLI

```
# Create a new console application with name Models
dotnet new console -n "Models"
```

.NET tools



MS Dev Tools

Integrated Development Environments and CLI

Visual Studio and Visual Studio Code offer comprehensive development environments with features like code completion, debugging, and project management.

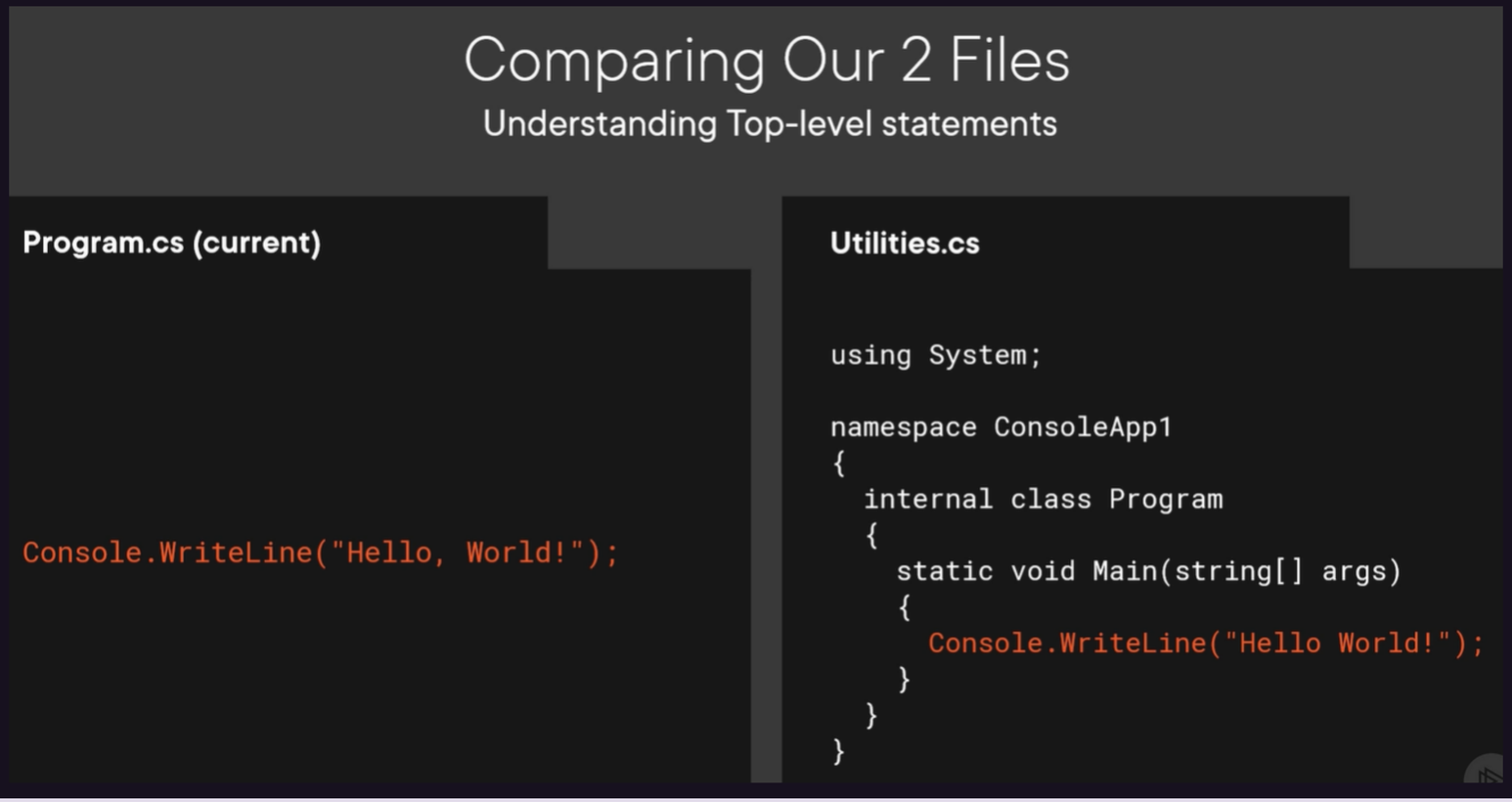
nuget

Nuget

A package manager for .NET that simplifies the process of adding and managing third-party libraries and dependencies.

The main method

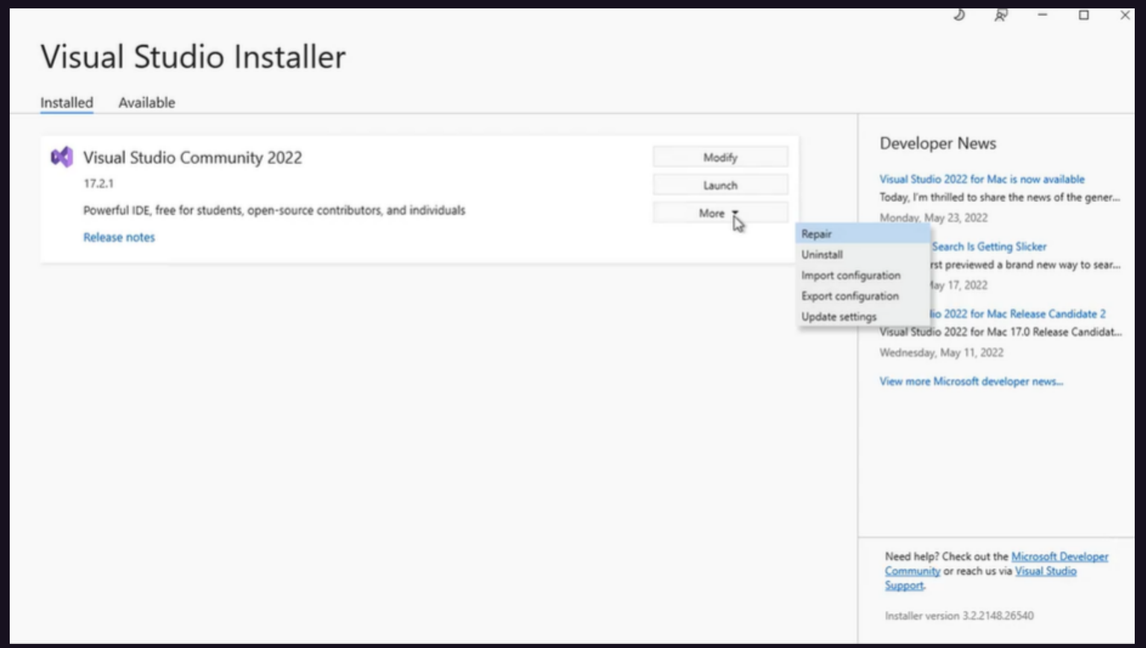
The main method is the entry point of a C# application, it gets called upon start of the app, recently from C#10 version this has been hidden to the user but it is created implicitly



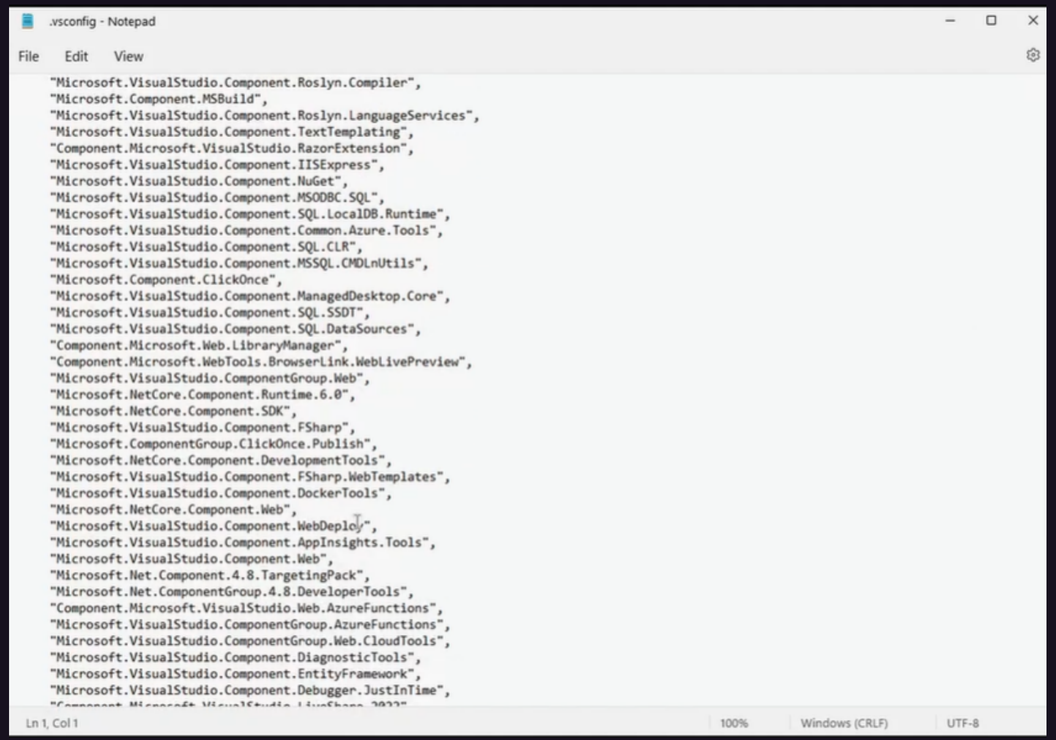
Old vs new Main method comparison

Visual studio installer options

The Installer for Visual Studio IDE is a tool that facilitates the installation and management of Visual Studio on your development machine. It provides a user-friendly interface to customize your Visual Studio installation, select the components and workloads you need, and manage updates and extensions.



Export setup



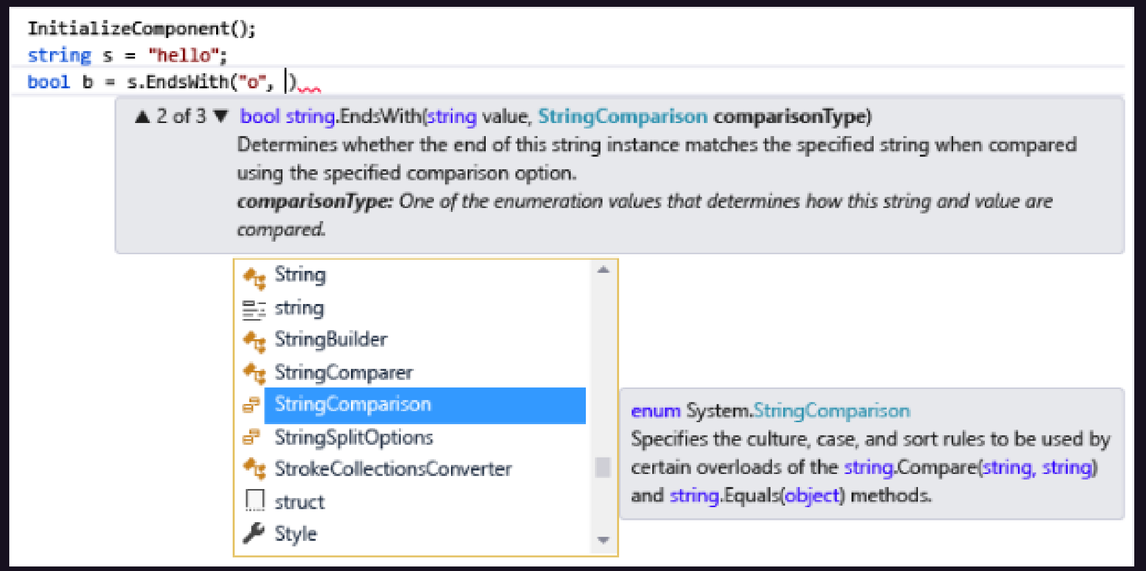
Setup config file

Intellisense

IntelliSense is a code-completion aid that includes a number of features: List Members, Parameter Info, Quick Info, and Complete Word. These features help you to learn more about the code you're using, keep track of the parameters you're typing, and add calls to properties and methods with only a few keystrokes.

Icono	Accesibilidad	Descripción
	Clase pública	El acceso no está restringido.
	Clase protegida	El acceso está limitado a la clase contenedora o a los tipos derivados de la clase contenedora.
	Clase interna protegida	El acceso está limitado al ensamblado actual o a los tipos derivados de la clase contenedora.
	Clase interna	El acceso está limitado al ensamblado actual.
	Clase privada	El acceso está limitado a la clase contenedora o a los tipos derivados de la clase contenedora que hay en el ensamblado actual. (Disponible desde C# 7.2.)

Quick info



Icon for access modifiers

Demo

We will check the following:

Overview of a console and desktop application

Analyze Compilation process

Analyze IL Code

Overview of Visual Studio

You can check the following Repository for some examples:

C# fundamentals

Tasks

Quick Exercises

Use an IL code inspector for the following implementations:

- A class implementation
- A foreach implementation

Indicate what is the implementation behind each example, does the implementation comes from an already existing one?

Homework

Using the C# Common Language Infrastructure shown (Compilar, CLR, JIT, etc) compare with other Language Infrastructure, you can pick any other language other than F# or Visual Basic.

Use images, pictures, diagrams if it helps you