

# Introduction to .NET Framework

## What is COM?

COM stands for **Component Object Model**.

The COM is one of Microsoft's Technologies.

Using this technology we can develop Windows applications as well as web applications.

In earlier COM, VB is the programming language that is used to implement Windows applications and ASP is used to implement web applications.



## What are the disadvantages of COM?

The major two disadvantages of COM is

1. Incomplete object-oriented programming means it will not support all the features of OOPs.
2. Platform dependent means COM applications can run on only Windows OS.

To overcome the above problems, the .NET Framework comes into the picture



## What .NET Represents?

NET stands for Network Enabled Technology. In .NET, dot (.) refers to object-oriented and NET refers to the internet. So the complete .NET means through object-oriented we can implement internet-based applications.

## What is a Framework?

A Framework is a Software. Or you can say a framework is a collection of many small technologies integrated together to develop applications that can be executed anywhere

## What is .NET Framework?

According to Microsoft, .NET Framework is a software development framework for building and running applications on Windows. The .NET Framework is part of the .NET platform, a collection of technologies for building apps for Linux, macOS, Windows, iOS, Android, and more.



## Developers often use the terms “library” and “framework” interchangeably. But there is a difference.

There isn't anything magic about frameworks or library. Both libraries and frameworks are reusable code written by someone else. Their purpose is to help you solve common problems in easier ways.

I often use a house as a metaphor (figure of speech-symbolic of something else) for web development concepts.

A library is like going to Ikea (Furniture). You already have a home, but you need a bit of help with furniture. You don't feel like making your own table from scratch. Ikea allows you to pick and choose different things to go in your home. You are in control.

A framework, on the other hand, is like building a model home. You have a set of blueprints and a few *limited* choices when it comes to architecture and design. Ultimately, the contractor and blueprint are in control. And they will let you know when and where you can provide your input.



## The Technical Difference

The technical difference between a framework and library lies in a term called inversion of control.

When you use a library, you are in charge of the flow of the application. You are choosing when and where to call the library. When you use a framework, the framework is in charge of the flow. It provides some places for you to plug in your code, but it calls the code you plugged in as needed.

**Note:** The **Inversion-of-Control (IoC)** pattern, is about providing *any kind* of callback, which "implements" and/or controls reaction, instead of acting ourselves directly



## Different Types of .NET Framework

.NET is a developer platform made up of tools, programming languages, and libraries that are for building many different types of applications such as Desktop, Web, Mobile, etc. There are various implementations of .NET. Each implementation allows .NET code to execute in different places such as Linux, macOS, Windows, iOS, Android, and many more.

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.




## What does the DOTNET Framework Provide?

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

1. CL (Class Libraries)
2. CLR (Common Language Runtime)

### **.NET Framework Class Libraries:**

The .NET Framework Class Libraries are designed by Microsoft. Without Class Libraries, we can't write any code in .NET. So, Class Libraries are also known as the Building block of .NET Programs. These are installed into the machine when we installed the .NET framework. Class Libraries contains pre-defined classes and interfaces and these classes and interfaces are used for the purpose of application development. The Class Library also provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. Starting with the .NET Framework 4, the default location for the Global Assembly Cache is %windir%\Microsoft.NET\assembly. In earlier versions of the .NET Framework, the default location is %windir%\assembly.





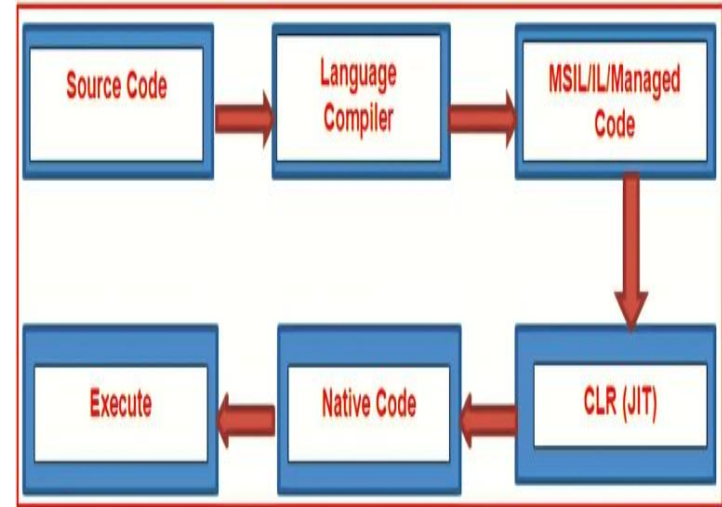
## Common Language Runtime (CLR):

CLR stands for Common Language Runtime and it is the core component under the .NET framework which is responsible for converting the MSIL (Microsoft Intermediate Language) code into native code and then provides the runtime environment to execute the code. That means 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.

In the .NET framework, the Code is Compiled Twice.

1. In the 1st compilation, the source code is compiled by the respective language compiler and generates the intermediate code which is known as MSIL (Microsoft Intermediate Language) or IL (Intermediate language code) Or Managed Code.
2. In the 2nd compilation, MSIL code is converted into Native code (native code means code specific to the Operating system so that the code is executed by the Operating System ) and this is done by CLR.

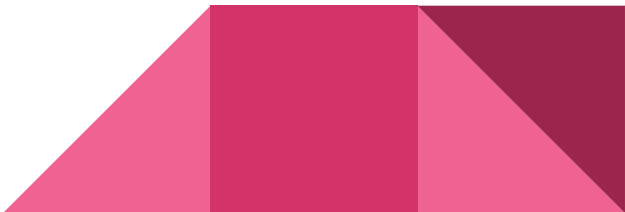
Always 1st compilation is slow and 2nd compilation is fast.



## What is JIT?

JIT stands for the **Just-in-Time compiler**. It is the component of CLR that is responsible for converting MSIL code into Native Code. Native code is the code that is directly understandable by the operating system.

## What is not .NET?


1. .NET is not an Operating system.
  2. It is not an application or package.
  3. .NET is not a database
  4. It is not an ERP application.
  5. .NET is not a Testing Tool.
  6. It is not a programming language.
- 

## What is exactly DOTNET?

.NET is a framework tool that supports many programming languages and many technologies. .NET support 60+ programming languages. In 60+ programming languages, 9 are designed by Microsoft, and the remaining are designed by non-Microsoft. Microsoft-designed programming languages are as follows

1. VB.NET
2. C#.NET
3. VC++.NET
4. J#.NET
5. F#.NET
6. Jscript.NET
7. WindowsPowerShell
8. Iron phyton
9. Iron Ruby

Technologies supported by the .NET framework are as follows

1. ASP.NET (Active Server Pages.NET)
  2. ADO.NET (Active Data Object.NET)
  3. WCF (Windows Communication Foundation)
  4. WPF (Windows Presentation Foundation)
  5. WWF (Windows Workflow Foundation)
  6. AJAX (Asynchronous JavaScript and XML)
  7. LINQ (Language Integrated Query)
- 

## What is a Language and its need?

1. Language acts as the mediator between the programmer and the system.
2. It offers some rules and regulations for writing the program.
3. The language also offers some libraries which are required for writing the program.

## What are Technology and its needs?

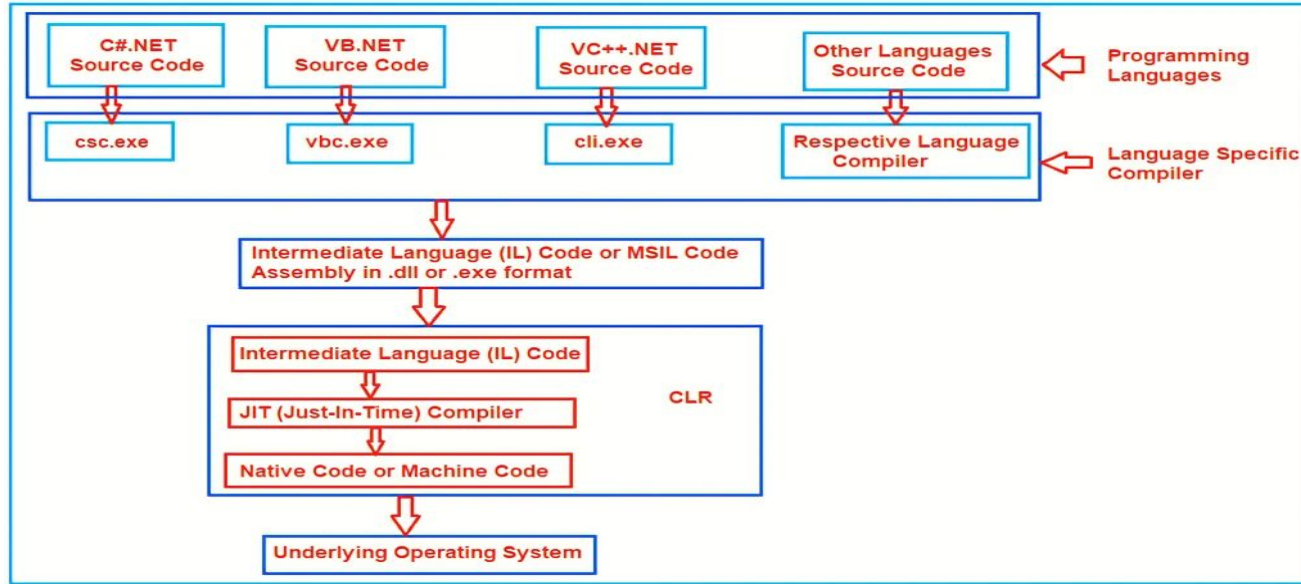
1. Technology is always designed for a particular purpose.
2. For example, the development of web-related applications in .NET using the technology ASP.NET.
3. But the technology does not offer any specific rules for writing the programs. That's why technology can't be implemented individually.
4. VB.NET, and C#.NET both are programming languages. Using these two languages we can implement Windows/Desktop applications individually.
5. Every language is having its own compiler

Language	File Extensions	Compiler
VB .NET	.VB	VBC(Visual Basic Compiler)
C# .NET	.CS	CSC (C Sharp Compiler)

## Common Language Runtime in .NET Framework

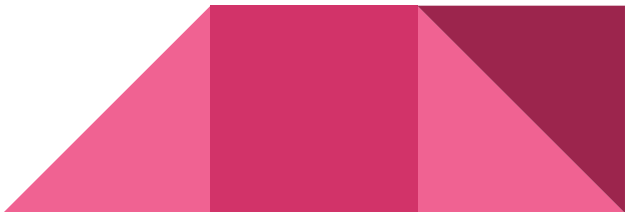
How is a .NET Application Compiled and Run?

In order to understand how exactly a .NET Application is compiled and run, please have a look at the following image.



First, the developer has to write the code using any dot net-supported programming languages such as C#, VB, J#, etc. Then the respective language compiler will compile the source code and generate something called **Microsoft Intermediate Language (MSIL) or Intermediate Language (IL) code**. For example, if the programming language is C#, then the compiler is CSC and if the programming language is VB, then the compiler will be VBC. This Intermediate Language (IL) code is half-compiled code i.e. partially compiled code and cannot be executed directly by the Operating System. To execute this Microsoft Intermediate Language (MSIL) or Intermediate Language (IL) code on our machine, the .NET Framework provides something called Common Language Runtime (CLR) which takes the responsibility to execute our Microsoft Intermediate Language (MSIL) or Intermediate Language (IL) Code.

The CLR takes the IL (Intermediate Language) code and gives it to something called the JIT (Just-in-Time) Compiler which is part of the CLR. **The JIT compiler reads each and every line of the IL code and converts it to machine-specific instructions (i.e. into binary format) which will be executed by the underlying Operating System.** The CLR in .NET Framework provides the runtime environment to execute the .NET Applications.




## What is Intermediate Language (IL) Code in .NET Framework?

The Intermediate Language or IL code in .NET Framework is a half-compiled or partially-compiled or CPU-independent partially compiled code and this code can not be executed by Operating System.

### Why Partial Compiled Code or Why Not Fully Compiled Code?

As a developer, you might be thinking about why the respective language compiler generates partially compiled code or why not fully compiled code i.e. machine code or binary code in .NET Framework when we compile the source code. The reason is very simple. We don't know in what kind of environment .NET Code is going to be run (for example, Windows XP, Windows 7, Windows 10, Windows 11, Windows Server, etc.).

In other words, we don't know what operating system is going to run our application. We also don't know the CPU configuration, Machine Configuration, Security Configuration, etc. of the corresponding operating system where we want to run our .NET Applications. So, the Microsoft Intermediate language (MSIL) or Intermediate language (IL) code is partially compiled, and at runtime, this Microsoft Intermediate language (MSIL) or Intermediate language (IL) code is compiled to Machine-Specific instructions or you can say binary code based on underlying Operating System, CPU, Machine Configuration, etc. by the CLR in .NET Framework.



## Common Language Runtime (CLR) in .NET Framework:

CLR is the heart of the .NET Framework and it contains the following components.

1. Security Manager
2. JIT Compiler
3. Memory Manager
4. Garbage Collector
5. Exception Manager
6. Common Language Specification (CLS)
7. Common Type System (CTS)

Let us discuss what each of these components does in detail in .NET Framework.





**Security Manager:** There are basically two components to managing security. They are as follows:

1. CAS (Code Access Security)
2. CV (Code Verification)

These two components are basically used to check the privileges of the current user and then check whether the user is allowed to access the assembly or not. The Security Manager also checks what kind of rights or authorities this code has and whether it is safe to be executed by the Operating System or not. So, basically, these types of security checks are managed by the Security Manager Component of CLR in .NET Application.

**JIT Compiler:** The JIT (Just-In-Time) Compiler is responsible for converting the MSIL code into Native Code (Machine Code or Binary code) that is going to be executed by the Operating System. The Native Code (Machine Code or Binary code) is directly understandable by the system hardware. JIT compiles the code just before the execution and then saves this translation in memory.

**Memory Manager:** The Memory Manager component of CLR in the .NET Framework allocates the necessary memory for the variables and objects that are to be used by the application.

**Garbage Collector:** When a dot net application runs, lots of objects are created. At a given point in time, it is possible that some of those objects are not used by the application. So, Garbage Collector in .NET Framework is nothing but a Small Routine or you can say it's a Background Process Thread that runs periodically and try to identify what objects are not being used currently by the application and deallocates the memory of those objects.

**Exception Manager:** The Exception Manager component of CLR in the .NET Framework redirects the control to execute the catch or finally blocks whenever an exception has occurred at runtime. If we have not handled the Runtime Exception, then the Exception Manager will throw the exception and abnormally terminate the program execution at the line where the exception has occurred.



## Common Type System (CTS) in .NET Framework

The .NET Framework supports many programming languages such as C#, VB.NET, J#, etc. Every programming language has its own data type. One programming language data type cannot be understood by other programming languages. But, there can be situations where we need to communicate between two different programming languages. For example, we need to write code in the VB.NET language and that code may be called from C# language. In order to ensure smooth communication between these languages, the most important thing is that they should have a Common Type System (CTS) which ensures that data types defined in two different languages get compiled to a common data type.

CLR in .NET Framework will execute all programming languages data types. This is possible because CLR has its own data types which are common to all programming languages. At the time of compilation, all language-specific data types are converted into CLR's data type. This data type system of CLR is common to all .NET Supported Programming languages and this is known as the Common Type System (CTS).




# Common Language Specification (CLS) in .NET Framework

## What is Common Language Specification (CLS) in .NET Framework?

CLS (Common Language Specification) is a part of CLR in the .NET Framework. The .NET Framework supports many programming languages such as C#, VB.NET, J#, etc. Every programming language has its own syntactical rules for writing the code which is known as a language specification. One programming language syntactical rules (language specification) cannot be understood by other programming languages. But, there can be situations where we need to communicate between two different programming languages. In order to ensure smooth communication between different .NET Supported Programming Languages, the most important thing is that they should have Common Language Specifications which ensures that language specifications defined in two different languages get compiled to a Common Language Specification.

CLR in .NET Framework will execute all programming language's code. This is possible because CLR having its own language specification (syntactical rules) which are common to all .NET Supported Programming Languages. At the time of compilation, every language compiler should follow this language specification of CLR and generate the MSIL code. This language specification of CLR is common for all programming languages and this is known as Common Language Specifications (CLS).



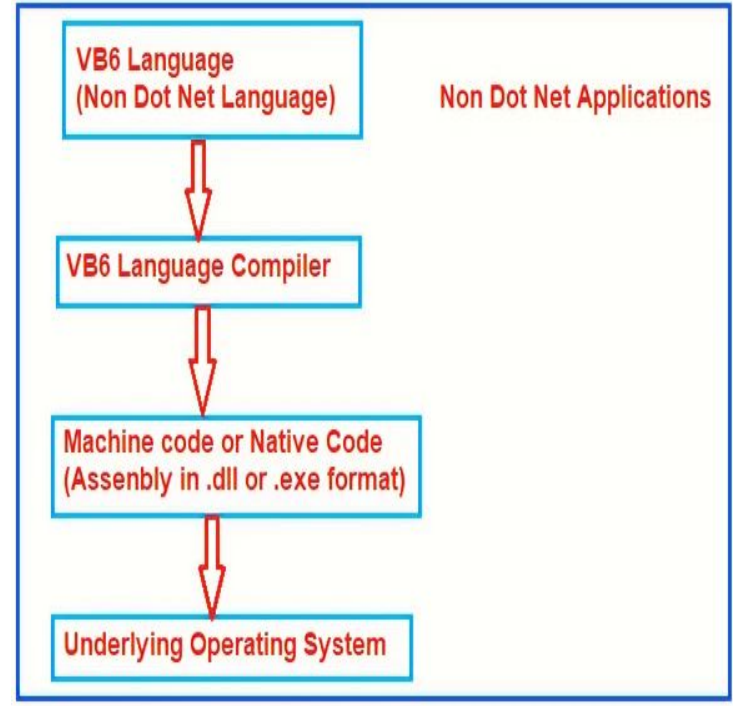
## .NET Program Execution Process Flow

But before understanding the .NET Program Execution process, let us first understand how non-dot net applications such as C, VB6, and C++ programs are executed.

### Non-DOT NET Program Execution Process:

We know that computers only understand machine-level code. The Machine-level code is also known as native code or binary code. So when we compile a C, VB6, or C++ program the respective language compiler compiles the respective language source code and generates the native machine code (also called binary code) which can be understood by the underlying operating system and the system hardware.

The Native code or machine code that is generated by the respective language compiler is specific to the operating system on which it is generated. If we take this compiled native code and try to run it on another operating system, then it will fail. So the problem with this style of program execution is that it is not portable from one platform to another platform. That means it is platform-dependent.



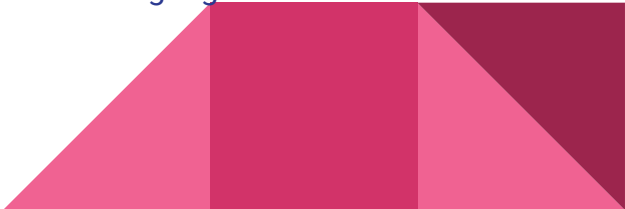
## .NET Program Execution Process

Let us now understand the .NET Program Execution Process in detail. Using .NET we can create different types of applications such as Console, Windows, Web, and Mobile Applications. Irrespective of the type of application when we execute any .NET application the following things are happening in order

The .NET application Source Code gets compiled into Microsoft Intermediate Language (MSIL) which is also called Intermediate language (IL) or Common Intermediate language (CIL) code. Both .NET and Non-DOTNET applications generate an assembly when we compile the application. Generally, the assemblies have an extension of .DLL or .EXE based on the type of application we compiled. For example, if we compile a Window or Console application in .NET, we get an assembly of type .EXE whereas when we compile a Web or Class Library Project in .NET, we get an assembly of type .DLL.

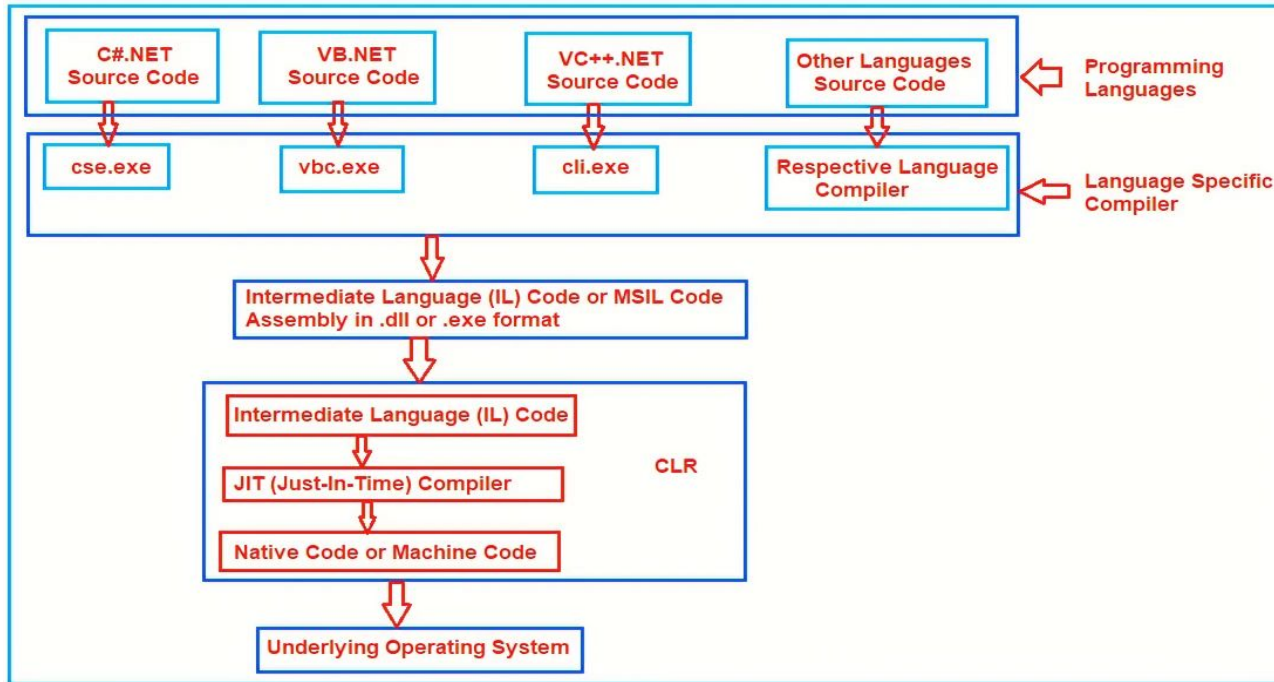
The difference between a .NET and NON-DOTNET assembly is that .NET Assembly is an Intermediate Language format whereas NON-.NET assembly is in native code format.

The NON .NET applications can run directly on top of the operating system as the NON-DOTNET assembly contains the native code whereas .NET applications run on top of a virtual environment called Common Language Runtime (CLR). CLR contains a component called Just-In-Time Compiler (JIT) which will convert the Intermediate language into native code which can be understood by the underlying operating system.



**.NET Program Execution Steps:** In .NET, the application execution consists of 2 steps. They are as follows:

In the step1 the respective language compiler compiles the Source Code into Intermediate Language (IL) and in the 2nd step, the JIT compiler of CLR will convert the Intermediate Language (IL) code into native code (Machine Code or Binary Code) which can then be executed by the underlying operating system. The above process is shown in the image below.




As the .NET assembly is in Intermediate Language (IL) format and not in native code or machine code, the .NET assemblies are portable to any platform as long as the target platform has the Common Language Runtime (CLR). The target platform's CLR converts the Intermediate Language code into native code or machine code that the underlying operating system can understand.

Intermediate Language code is also called managed code. This is because CLR manages the code that runs inside it. For example, in a VB6 program, the developer is responsible for de-allocating the memory consumed by an object. If a programmer forgets to de-allocate memory, then it may get out-of-memory exceptions. On the other hand, a .NET programmer needs not to worry about de-allocating the memory consumed by an object. Automatic memory management also known as garbage collection is provided by CLR. Apart from garbage collection, there are several other benefits provided by the CLR which we will discuss in a later session. Since CLR is managing and executing the Intermediate Language it (IL) is also called the managed code.

.NET supports different programming languages like C#, VB, J#, and C++. C#, VB, and J# can only generate managed code (IL) whereas C++ can generate both managed code (IL) and unmanaged code (Native code).

The native code is not stored permanently anywhere after we close the program the native code is thrown away. When we execute the program again the native code gets generated again.

The .NET program is similar to Java program execution. In Java, we have bytecodes and JVM (Java Virtual Machine) whereas in .NET we have Intermediate Language and CLR (Common Language Runtime).



## Intermediate Language (ILDASM & ILASM) in C#.NET

What happens when we compile a .NET Application?

When we compile any .NET application, it will generate an assembly with the extension of either a .DLL or an .EXE. For example, if you compile a Windows or Console application, then you will get an .EXE, whereas if you compile a Web or Class library project, then you will get a .DLL. Irrespective of whether it is a .DLL or .EXE, an assembly consists of two things i.e. Manifest and Intermediate language. Let us understand how the Intermediate Language and Manifest look like in .NET Framework with an example.





# Assembly DLL EXE in .NET Framework

## What is an Assembly in .NET?

According to MSDN, Assemblies are the building block of .NET Framework applications; they form the fundamental unit of deployment. In simple words, we can say that Assembly is nothing but a precompiled .NET Code that can be run by CLR (Common Language Runtime).

## Types of Assemblies in .NET Framework:

In the .NET Framework, there are two types of assemblies. They are as follows:

1. EXE (Executable)
2. DLL (Dynamic Link Library)

In .NET Framework when we compile a Console Application or a Windows Application, it generates EXE, whereas when we compile a Class Library Project or ASP.NET web application, then it generates DLL. In .NET framework, both EXE and DLL are called assemblies.

So, in short, the difference between them is an EXE is an executable file and can run by itself as an application whereas DLL is usually consumed by an EXE or by another DLL and we cannot run or execute DLL directly.

Now, the question that should come to your mind why do we need DLLs as it is not invoked by themselves. The reason behind the DLL is reusability. Suppose you want some class, or logic, or something else in many applications, then simply put those classes, and logic inside a DLL, and refer to that DLL wherever it is required.



## Understanding Intermediate Language (ILDASM and ILASM) Code in C#

In order to understand Intermediate Language Code (ILDASM and ILASM) in C#, let us create a simple console application. Once you create the console application, please modify the Program class as shown below.

```
using System;
namespace ILDASMDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Understanding ILDASM and ILASM");
            Console.Read();
        }
    }
}
```

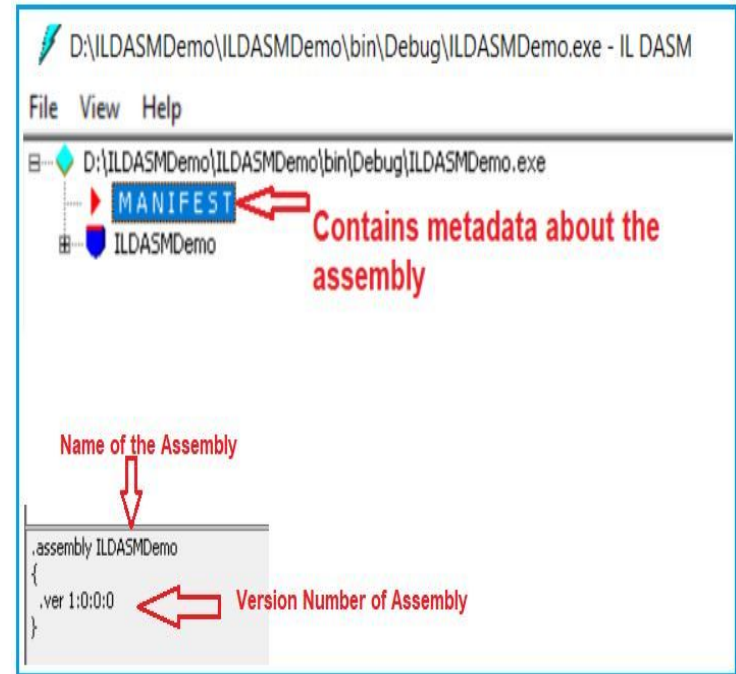
Now, build the application. Once you build the application, the above source code is compiled and intermediate language code generated and packaged into an assembly. In order to see the assembly, just right-click on the Project and select Open Folder in File Explorer option and then go to the bin => Debug folder and you should see an assembly with .exe extension as shown in the below image as it is a console application.

## How to view the Intermediate Language Code in .NET Framework?

The .NET framework provides a nice tool called **ILDASM (Intermediate Language DisAssembler)** to view the code of the intermediate language in C#.NET. In order to use the ILDASM tool, you need to open Visual Studio Command Prompt in Administrator mode or download CIL disassembler app to view the .exe or .dll files.

### What is Manifest?

**Manifest** contains metadata about the assembly like the name of the assembly, the version number of the assembly, culture, and strong name information as shown in the image.




## Managed and Unmanaged Code in .NET

### What exactly is the managed and unmanaged code in .NET?

The codes which run under the complete control of CLR are called Managed Code in .NET. These kinds of code (Managed code in C#) are run by a dot net runtime environment. If the dot net framework is not installed or if dot net runtime is not available, then these kinds of codes are not going to be executed. CLR will provide all the facilities and features of .NET to the managed code execution like Language Interoperability, Automatic memory management, Exception handling mechanism, code access security, etc.

On the other hand, Skype, PowerPoint, and Microsoft Excel do not require dot net runtime, they run under their own environment. So, in short, the code (EXE, Web App) which not run under the control of CLR is called unmanaged code in .NET. CLR will not provide any facilities and features of .NET to the unmanaged code in C# execution like Language Interoperability, Automatic memory management, Exception handling mechanism, code access security, etc.



## .NET Core vs .NET Framework

### WHAT'S THE DIFFERENCE?

.NET Core



.NET Framework



Basis	.NET Core	.NET Framework
Platform or Framework	When we talk about .NET Core it is defined as the platform on which frameworks like ASP.NET Core and the Universal Windows Platform rely and extend the .NET Core platform's functionalities.	.Net Framework is a full-fledged development framework. The framework provides all the basic requirements for the development of applications such as UI, DB connectivity, services, APIs, etc.
Open-Source	.NET Core is an open-source platform.	The .Net Framework includes certain open-source components.
Cross-Platform	It is based on the concept of "create once, run anywhere." Because it is cross-platform, it is compatible with a variety of operating systems, including Windows, Linux, and Mac OS.	.NET Framework is compatible with Windows OS (operating system) only
Application models	The Application Model of .Net Core includes ASP.NET and windows universal apps.	The Application Model of the .NET Framework includes WinForms, ASP.NET, and WPF.
Installation	.Net Core is cross-platform, hence it needs to be installed independently.	.NET Framework has a single packaged installation and runtime environment for windows.
Microservices support	.NET Core has support for microservices., .NET Core allows a mix of technologies that can be minimalized for each microservice.	When we talk about the .NET Framework it does not allow for the construction and deployment of microservices in multiple languages.
REST services support	.NET Core has no support for WCF (Windows Communication Foundation) services. You would always need to create a REST API.	When it comes to WCF (Windows Communication Foundation) services, the .NET Framework is a fantastic choice. It also works with RESTful services.
Performance and Scalability	.NET core provides high scalability and performance compared to .NET Framework because of its architecture.	.NET Framework is less scalable and provides low performance compared to .NET Core.
Security	Feature such as Code Access Security is not present in .NET core, so .NET Framework does have the edge in that case.	.NET Framework has this feature called code access security.
Focus on devices	.NET Core focuses to develop apps in a variety of domains like gaming, mobile, IoT, AI, etc.	.NET Framework is limited to Windows OS.
Compatibility	Mobile.NET Core is compatible with various operating systems-Windows, Linux, and Mac OS.	On the other .NET Framework is only compatible with Windows OS.
Mobile Development	Mobile apps are becoming more important for businesses. .NET Core has some support for mobile apps. It's compatible with Xamarin and other open-source platforms for mobile applications.	On the other hand, the .NET Framework does not support their development at all, and that is a problem.
CLI Tools	For all platforms, .NET Core provides a very lightweight CLI (Command Line Interface). There's always the option of switching to an IDE.	.NET Framework is too heavy for CLI. some developers prefer working on CLI rather than on IDE.
Deployment Model	When a new version of .NET Core is installed, it is updated on one computer at a time, resulting in new directories/folders being created in the existing program without affecting it. As a result, .NET Core provides a solid and adaptable deployment model.	IDE In the case of .NET Framework, when the updated version is released, it is first deployed on the internet information service only.
Packaging and shipping	.NET Core is shipped as a collection of Nugets packages.	All the libraries of the .NET Framework are packed and shipped together.

# INTRODUCTION TO C#

## C# HISTORY

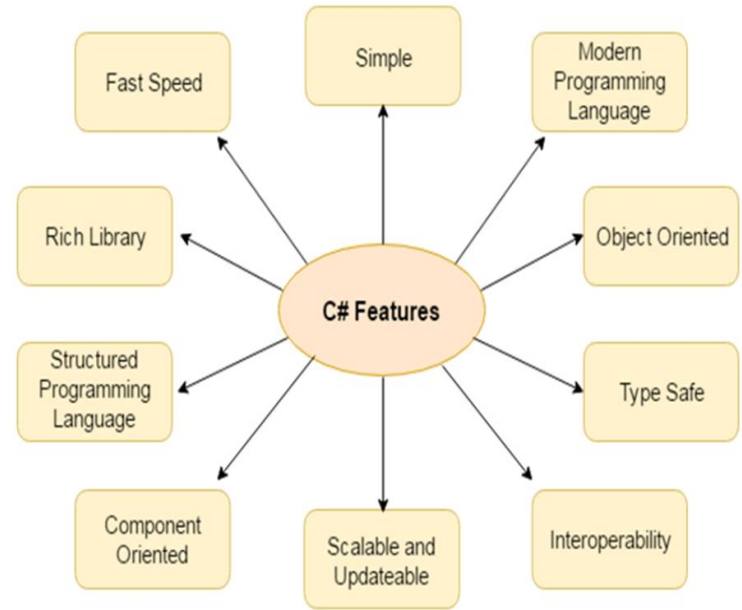
- History of C# language is interesting to know. Here we are going to discuss brief history of C# language.
- C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.
- Anders Hejlsberg is known as the founder of C# language.
- It is based on C++ and Java, but it has many additional extensions used to perform component oriented programming approach.
- C# has evolved much since their first release in the year 2002. It was introduced with .NET Framework 1.0 and the current version of C# is 11.0 (8 November 2022)



# C# FEATURES

C# is **object oriented programming language**. It provides a lot of features that are given below.

1. Simple
2. Modern programming language
3. Object oriented
4. Type safe
5. Interoperability
6. Scalable and Updateable
7. Component oriented
8. Structured programming language
9. Rich Library
10. Fast speed






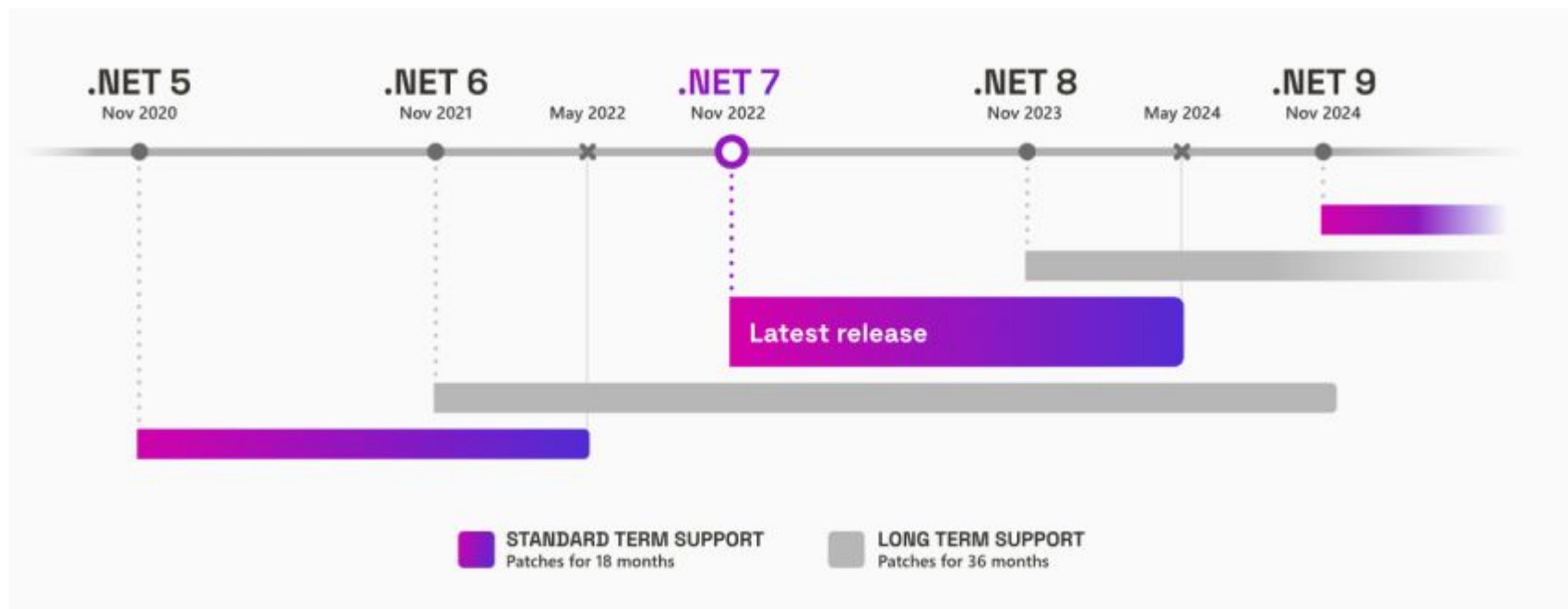
# Simplifying Development With .NET 7 And C# 11 - An Introduction To The Latest Features

Let's look at the important features and general topics that were introduced in .Net 7.0

- .Net 7.0 is the fastest ever, and there are more than 1000 performance-affecting changes in .Net 7.0, including reflection start of and much more
- Microsoft released a 250-page blog post on performance improvements in .Net 7.0. There are also noticeable performance improvements in Arm64
- If you want to spend less on building your application running in the cloud or reduce the resource required to run .Net 7.0 on-premises, updating to .Net 7.0 latest framework could be a better choice.

## Release Types

- Long-Term Support (LTS) releases are supported for 3 years after the initial release of its version.
  - Short-Term Support (STS) release. There will be 18 months of free support and patches. If you update to .NET 7, you are supposed to update to .NET 8 no later than six months after its release, or around May 2024.
- 



## **.NET 7**

.NET 7 is the seventh major release of the .NET framework. It is an open-source, cross-platform framework that can be used to build various applications, including web applications, desktop applications, and mobile applications. Some of the key features of .NET 7 include:

### **Improved performance**

Several performance improvements, including a faster runtime and better memory management.

### **Improved cloud support**

.NET 7 includes new APIs and features that make it easier to build cloud-native applications, including support for Kubernetes and Azure Functions.

### **Improved language support**

.NET 7 includes improved support for F#, C#, and Visual Basic, making it easier to develop applications in these languages.

### **Improved security**

.NET 7 includes several security enhancements, including support for hardware-based security features and improved cryptography.

### **Improved developer experience**

.NET 7 includes several improvements to the developer experience, including improved diagnostics and debugging tools.



## C# 11

C# 11 is the latest version of the C# programming language. It is designed to work seamlessly with .NET 7 and includes several new features and improvements that make it easier to write high-quality, efficient code. Some of the key features of C# 11 include:

### Improved pattern matching

C# 11 includes several improvements to pattern matching, including support for more patterns and better performance.

### Improved performance

C# 11 includes several performance improvements, including faster start-up times and improved garbage collection.

### Improved async programming

C# 11 includes several improvements to async programming, including best cancellation support and error handling.

### Improved language features

C# 11 includes several new language features, including interpolated strings, static abstract members, and global usings.

### Improved source generators

C# 11 includes several improvements to source generators, which allow developers to generate code at compile-time, including support for incremental generators.



LET'S BEGIN OUR JOURNEY TO THE CORE OF .NET TECHNOLOGY...

