MCA-II

Sub-.Net Programming

Faculty Name-Mr. Rahul Shinde.

An Overview of the Platform

An overall view of .NET, Microsoft's cross-platform development environment that allows you to build any type of application with C# and other languages.

The .NET platform has been powering up web, desktop, and mobile applications in both the startup and enterprise scenes.

.NET plays a central role in the software development industry.

What are the design goals of the .NET platform?

1.Interoperability - Because interaction between new and older applications is commonly required, the .NET Framework provides means to access functionality that is implemented in programs that execute outside the .NET environment.

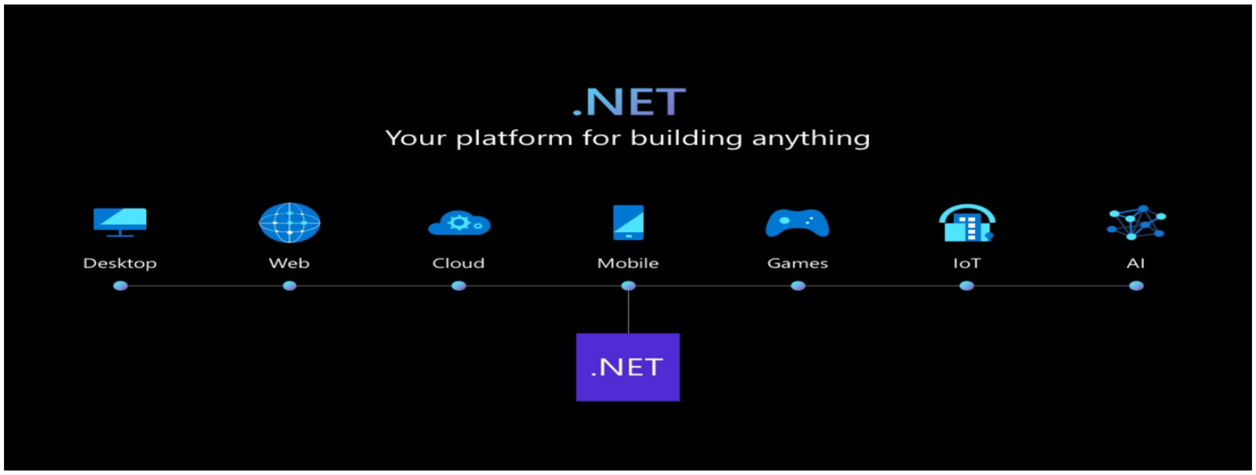
2.Common Runtime Engine - Programming languages on the .NET Framework compile into an intermediate language known as the Common Intermediate Language, or CIL (formerly known as Microsoft Intermediate Language, or MSIL).

3.Language Independence - The .NET Framework introduces a Common Type System, or CTS. The CTS specification defines all possible datatypes and programming constructs supported by the CLR and how they may or may not interact with each other.

4.Base Class Library - The Base Class Library (BCL), sometimes referred to as the Framework Class Library (FCL), is a library of types available to all languages using the .NET Framework.

5.Simplified Deployment - Installation of computer software must be carefully managed to ensure that it does not interfere with previously installed software, and that it conforms to increasingly stringent security requirements.

6.Security - .NET allows for code to be run with different trust levels without the use of a separate sandbox.



What is .NET?

.NET is an open-source and cross-platform development platform for building many types of applications.

Designed by Microsoft, the platform supports multiple programming languages and libraries to build web, mobile, desktop, IoT applications, and more.

The .NET platform has been designed to deliver productivity, performance, security, and reliability. It provides automatic memory management via a garbage collector (GC).

It includes a large set of libraries that have broad functionality and have been optimized for performance on multiple operating systems and chip architectures.

In simple terms, the .NET platform is a software that can do these tasks:

- Translate .NET programming language code into instructions that a computing device can process.
- Provide utilities for efficient software development. For example, it can find the current time or print text on the screen.
- Define a set of data types to store information like text, numbers, and dates on the computer.

What is a .NET implementation?

Various implementations of .NET allow .NET code to execute on different operating systems like Linux, macOS, Windows, iOS, Android, and many others.

.NET Framework

.NET Framework is the original .NET implementation. It supports running websites, services, desktop apps, and more on Windows. Microsoft released .NET Framework in the early 1990s.

.NET Core

Microsoft launched .NET Core in late 2014 to enable cross-platform support for .NET developers. The company released the newest version of the .NET Core, .NET 5.0, in November 2020 and renamed it .NET. The term .NET in this article refers to .NET 5.0. .NET Core is open-source on GitHub.

.NET Standard

.NET Standard is a formal specification of different functions (called APIs). Different .NET implementations can reuse the same code and libraries. Each implementation uses both .NET standard APIs and unique APIs specific to the operating systems it runs on.

Why choose .NET?

Ease of development

Developers like to use .NET because it includes many tools that make their work easier. For example, using the Visual Studio suite, developers can write code faster, collaborate efficiently, and test and fix their code efficiently. The ability to reuse code between implementations reduces the cost of development.

High-performing applications

.NET applications provide faster response times and require less computing power. They have strong built in security measures and efficiently perform server-side tasks like database access.

Community support

.NET is open source, which means that anyone can get access to use, read, and modify it freely. An active community of developers maintains and improves the .NET software. The .NET Foundation is an independent nonprofit organization established to support the .NET community. It provides learning resources, open-source .NET projects, and various events for .NET developers.

What is .Net Framework?

.Net Framework is a software development platform developed by Microsoft for building and running Windows applications. The .Net framework consists of developer tools, programming languages, and libraries to build desktop and web applications. It is also used to build websites, web services, and games.
The .Net framework was meant to create applications, which would run on the Windows Platform. The first version of the .Net framework was released in the year 2002. The version was called .Net

framework 1.0. The Microsoft .Net framework has come a long way since then, and the current version is .Net Framework 4.7.2.

The Microsoft .Net framework can be used to create both – **Form-based** and **Web-based** applications. Web services can also be developed using the .Net framework.

The framework also supports various programming languages such as Visual Basic and C#. So developers can choose and select the language to develop the required application.

.Net Framework Design Principle

The following design principles of the .Net framework is what makes it very relevant to create .Net based applications.

1) Interoperability – The .Net framework provides a lot of backward support. Suppose if you had an application built on an older version of the .Net framework, say 2.0. And if you tried to run the same application on a machine which had the higher version of the .Net framework, say 3.5. The application would still work. This is because with every release, Microsoft ensures that older framework versions gel well with the latest version.

2) Portability – Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and Linux.

3) Security – The .NET Framework has a good security mechanism. The inbuilt security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.

4) Memory management – The Common Language runtime does all the work or memory management. The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the “Garbage Collector” which runs as part of the .Net framework. The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.

5) Simplified deployment – The .Net framework also have tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.

.NET Framework is a software development framework for building and running applications on Windows.

.NET Framework is part of the .NET platform, a collection of technologies for building apps for Linux, macOS, Windows, iOS, Android, and more.

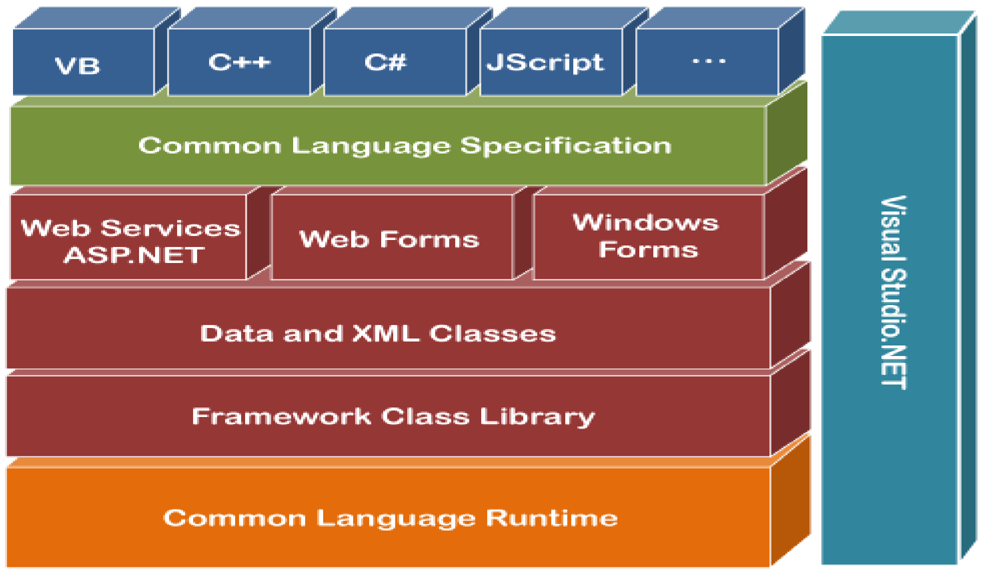
Introduction:

Net Architecture is a software architecture used for building applications that run on Microsoft's .NET platform. It provides a set of libraries and tools that simplify the development of complex applications by providing a consistent programming model and a standardized set of APIs.

Characteristics of .NET Frameworks are including the following:

- Memory management. In many programming languages, programmers are responsible for allocating and releasing memory and for handling object lifetimes. In .NET Framework apps, the CLR provides these services on behalf of the app.
- A common type system. In traditional programming languages, basic types are defined by the compiler, which complicates cross-language interoperability. In .NET Framework, basic types are defined by the .NET Framework type system and are common to all languages that target .NET Framework.
- An extensive class library. Instead of having to write vast amounts of code to handle common low-level programming operations, programmers use a readily accessible library of types and their members from the .NET Framework Class Library.
- Development frameworks and technologies. .NET Framework includes libraries for specific areas of app development, such as ASP.NET for web apps, ADO.NET for data access, Windows Communication Foundation for service-oriented apps, and Windows Presentation Foundation for Windows desktop apps.
- Language interoperability. Language compilers that target .NET Framework emit an intermediate code named Common Intermediate Language (CIL), which, in turn, is compiled at run time by the common language runtime. With this feature, routines written in one language are accessible to other languages, and programmers focus on creating apps in their preferred languages.
- Version compatibility. With rare exceptions, apps that are developed by using a particular version of .NET Framework run without modification on a later version.

Architecture of .NET Framework



Components of .NET Framework

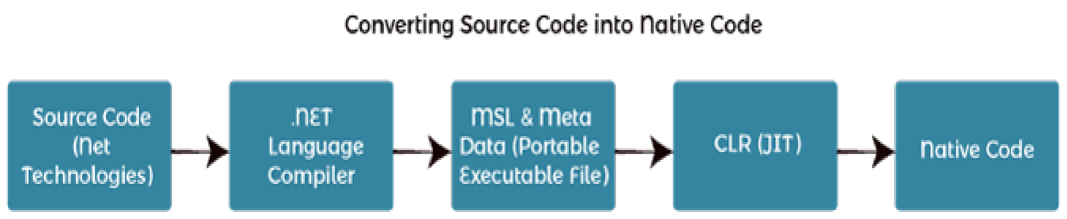
There are following components of .NET Framework:

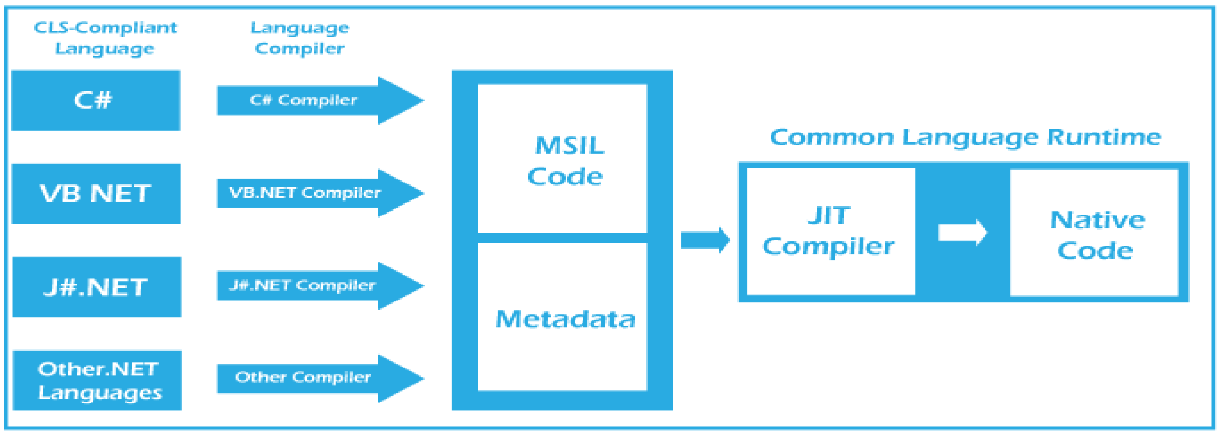
1. CLR (Common Language Runtime)
2. CTS (Common Type System)
3. BCL (Base Class Library)
4. CLS (Common Language Specification)
5. FCL (Framework Class Library)
6. .NET Assemblies
7. XML Web Services
8. Window Services

Common Language Runtime (CLR)

.NET CLR is a runtime environment that manages and executes the code written in any .NET programming language. CLR is the virtual machine component of the .NET framework. That language's compiler compiles the source code of applications developed using .NET compliant languages into CLR's intermediate language called MSIL, i.e., Microsoft intermediate language code. This code is platform-independent. It is comparable to byte code in java. Metadata is also generated during compilation and MSIL code and stored in a file known as the Manifest file. This metadata is generally about members and types required by CLR to execute MSIL code. A just-in-time compiler component of CLR converts MSIL code into native code of the machine. This code is platform-dependent. CLR manages memory, threads, exceptions, code execution, code safety, verification, and compilation.

The following figure shows the conversion of source code into native code.


Converting Source Code into Native Code
Source Code (Net Technologies)
.NET Language Compiler
MSL & Meta Data (Portable Executable File)
CLR (JIT)
Native Code



Exceution of a .NET Application

Functions of .NET CLR

The key functions of the .NET CLR include the following:

- Convert code into CLI
- Exception handling
- Type safety
- Memory management (using the Garbage Collector)
- Security
- Improved performance
- Language independency
- Platform independency
- Architecture independency

Components of .NET CLR

- Class Loader - Used to load all classes at run time.
- MSIL to Native code - The Just In Time (JTI) compiler will convert MSIL code into native code.
- Code Manager - It manages the code at run time.
- Garbage Collector - It manages the memory. Collect all unused objects and deallocate them to reduce memory.
- Thread Support - It supports the multithreading of our application.
- Exception Handler - It handles exceptions at run time.

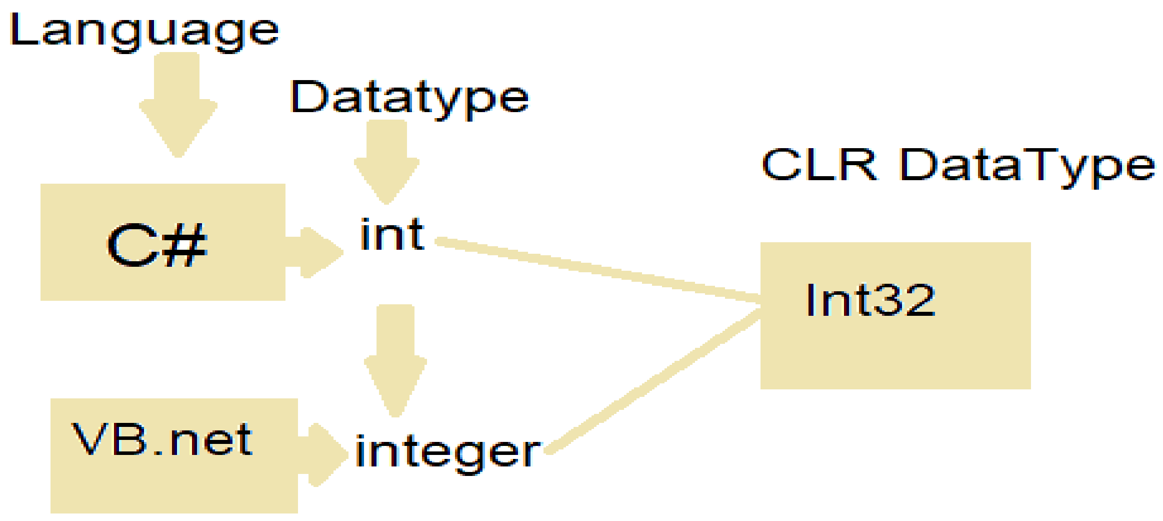
The runtime provides the following benefits:

- Performance improvements.
- The ability to easily use components developed in other languages.
- Extensible types provided by a class library.
- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows creation of multithreaded and scalable applications.
- Support for structured exception handling.
- Support for custom attributes.
- Garbage collection.
- Use of delegates instead of function pointers for increased type safety and security.

CTS Common Type System

CTS and CLS are parts of .NET CLR and are responsible for type safety within the code. Both allow cross-language communication and type safety.

Common Type System (CTS) describes the datatypes that can be used by managed code. CTS defines how these types are declared, used and managed in the runtime. It facilitates cross-language integration, type safety, and high-performance code execution. The rules defined in CTS can be used to define your own classes and values.



All the structures and classes available in CTS are common for all .NET Languages and the purpose of these is to support language independence in .NET. Hence it is called CTS.

functions:

1. It enables cross-language integration, type safety, and high-performance code execution.
2. It provides an object-oriented model for the implementation of many programming languages.
3. It defines rules that every language must follow which runs under .NET **FRAMEWORK** It ensures that objects written in different .NET languages like C#, VB.NET, F# etc. can interact with each other.

There are 2 Types of CTS that every .NET programming language have :

1. **Value Types:** Value Types will store the value directly into the memory location. These types work with stack mechanisms only. CLR allows memory for these at Compile Time.
2. **Reference Types:** Reference Types will contain a memory address of value because the reference types won't store the variable value directly in memory. These types work with Heap mechanism. CLR allot memory for these at Runtime.

CLS

CLS stands for Common Language Specification and it is a subset of CTS. It defines a set of rules and restrictions that every language must follow which runs under the .NET framework. The languages which follow this set of rules are said to be CLS Compliant. In simple words, CLS enables cross-language integration.

The CLS is a specification that defines the rules for supporting the language integration in a certain way that the programs are written in any language, still, it can interoperate with the one another seamlessly while taking the full advantage of concepts such as exceptions handling, inheritance, polymorphism, and other features accordingly. These CLS rules and the specification are documented in the ECMA proposed standard document.

For Example

1. if we talk about C# and VB.NET then, in C# every statement must have to end with a semicolon. it is also called a statement Terminator, but in VB.NET each statement should not end with a semicolon(;).

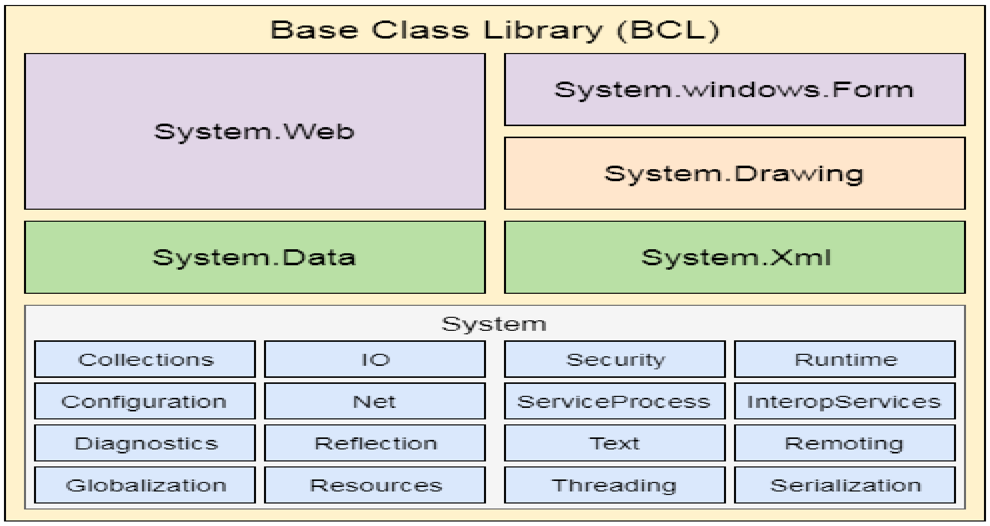
Base Class Library or Framework class library

The Base Class Library (BCL) is a sub-part of the framework that provides library support to common language runtime. And it is primarily developed by using C# language. And it includes the System namespace and core types of the .NET framework. The Base Class Library is also known as .Net Framework Class Library (FCL).

The Base Class Library (BCL) has a rich collection of library features and functions. And the main specialty of this library is that it can be consumed by any .NET language. It provides a set of reusable types, collections, classes, interfaces, and other components that are commonly used in .NET applications.

following functions of BCL.

- Base and user-defined data types
- Support for exceptions handling
- input/output and stream operations
- Communications with the underlying system
- Access to data
- Ability to create Windows-based GUI applications
- Ability to create web-client and server applications
- Support for creating web services



BCL divides into two parts:

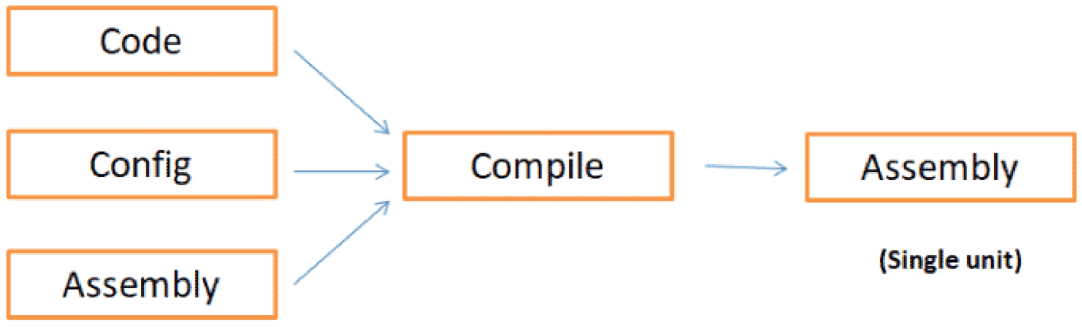
1. **User defined class library**
   - **Assemblies -** It is the collection of small parts of deployment an application's part. It contains either the DLL (Dynamic Link Library) or exe (Executable) file.
     1. In LL, it uses code reusability, whereas in exe it contains only output file/ or application.
     2. DLL file can't be open, whereas exe file can be open.
     3. DLL file can't be run individually, whereas in exe, it can run individually.
     4. In DLL file, there is no main method, whereas exe file has main method.
2. **Predefined class library**
   - **Namespace -** It is the collection of predefined class and method that present in .Net. In other languages such as, C we used header files, in java we used package similarly we used "using system" in .NET, where using is a keyword and system is a namespace.

.NET Assemblies

A .NET assembly is the main building block of the .NET Framework. It is a small unit of code that contains a logical compiled code in the Common Language infrastructure (CLI), which is used for deployment, security and versioning. It defines in two parts (process) DLL and library (exe) assemblies. When the .NET program is compiled, it generates a metadata with Microsoft Intermediate Language, which is stored in a file called Assembly.

Advantages:

- It is implemented as .exe or .dll .
- It is loaded into memory whenever needed.
- We can share assemblies between applications using GAC (Global Assembly Cache).
- The System. Reflection namespace has classes like Assembly which can be used to get the details of the assembly and with that, it is also possible to load an assembly dynamically at runtime.
- It increases the overall performance.

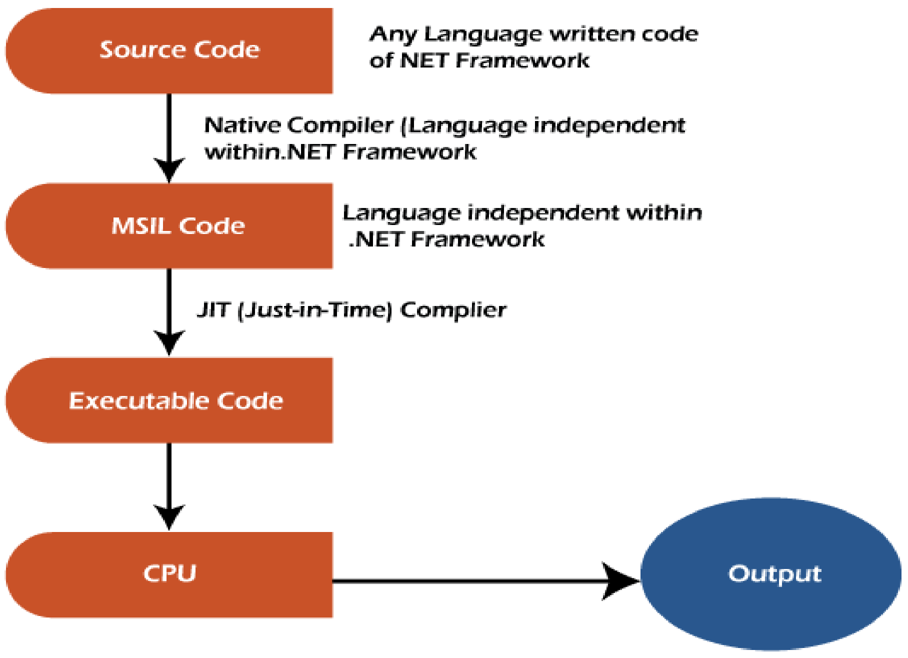


managed code

A code which is written to aimed to get the services of the managed runtime environment execution like CLR(Common Language Runtime) in .NET Framework is known as **Managed Code**. It always implemented by the managed runtime environment instead of directly executed by the operating system. The managed runtime environment provides different types of services like garbage collection, type checking, exception handling, bounds checking, etc. to code automatically without the interference of the programmer. It also provides memory allocation, type safety, etc to the code. The application is written in the languages like Java, C#, VB.Net, etc. are always aimed at runtime environment services to manage the execution and the code written in these types of languages are known as managed code.

What are the advantages of using Managed Code?

- It improves the security of the application like when you use runtime environment, it automatically checks the memory buffers to guard against buffer overflow.
- It implement the garbage collection automatically.
- It also provides runtime type checking/dynamic type checking.
- It also provides reference checking which means it checks whether the reference point to the valid object or not and also check they are not duplicate.

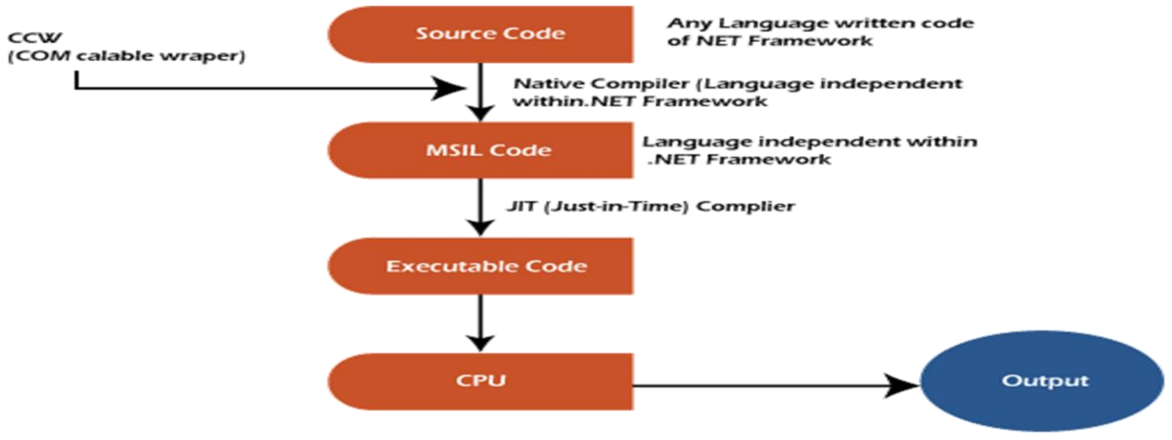

Source Code
Any Language written code of NET Framework
Native Compiler (Language independent within.NET Framework
MSIL Code
Language independent within .NET Framework
JIT (Just-in-Time) Complier
Executable Code
CPU
Output

Managed Code Execution Process

Unmanaged Code

Applications that do not run under the control of the CLR are said to be unmanaged.

A code which is directly executed by the operating system is known as **Unmanaged code**. It always aimed for the processor architecture and depends upon computer architecture. When this code is compiled it always tends to get a specific architecture and always run on that platform, in other words, whenever you want to execute the same code for the different architecture you have to recompile that code again according to that architecture. It always compiles to the native code that is specific to the architecture. In unmanaged code, the memory allocation, type safety, security, etc are managed by the developer


CCW
(COM calable wraper)
Source Code
Any Language written code of NET Framework
Native Compiler (Language independent within.NET Framework
MSIL Code
Language independent within .NET Framework
JIT (Just-in-Time) Complier
Executable Code
CPU
Output

Unmanaged Code Execution Process

What are the advantages of using Unmanaged Code?

- It provides the low-level access to the programmer.
- It also provides direct access to the hardware.
- It allows the programmer to bypass some parameters and restriction that are used by the managed code framework.

What are the disadvantages of Unmanaged Code?

- It does not provide security to the application.
- Due to the access to memory allocation the issues related to memory occur like memory buffer overflow, etc.
- Error and exceptions are also handled by the programmer.

JIT Compiler in .NET

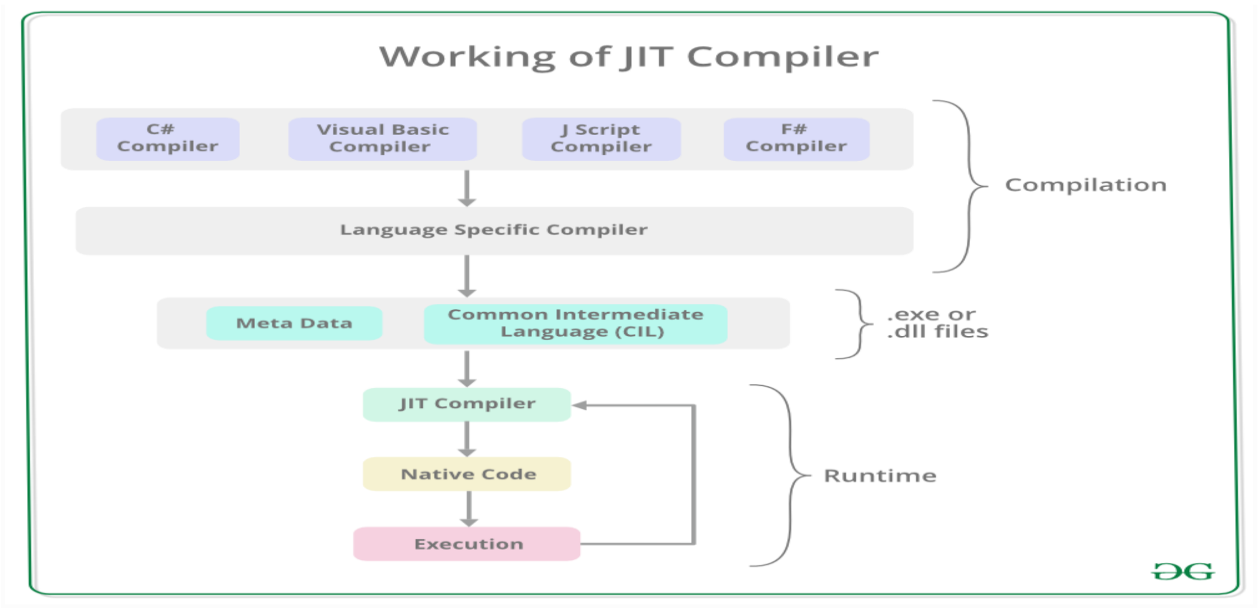
Compilers are tools that translate source code to machine understandable language.

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.

JIT stands for just-in-time compiler. It converts the MSIL code to CPU native code as it is needed during code execution. It is called just-in-time since it converts the MSIL code to CPU native code; when it is required within code execution otherwise it will not do anything with that MSIL code.

A language-specific compiler converts the source code to the intermediate language.

Working of JIT Compiler: The JIT compiler is required to speed up the code execution and provide support for multiple platforms. Its working is given as follows:



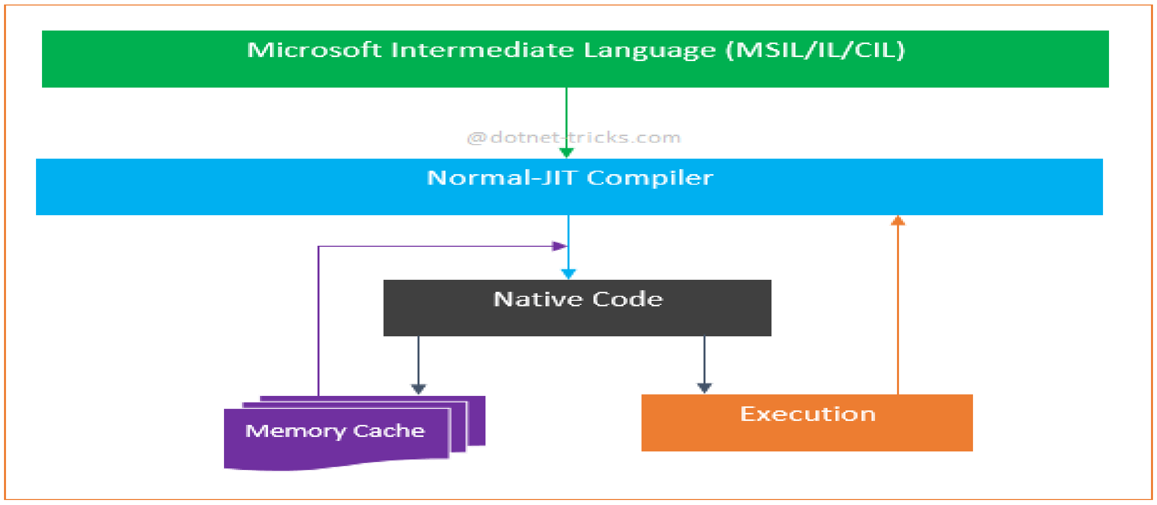
There are **3** types of JIT compilers which are as follows:

- Pre - JIT.
- Econo - JIT.
- Normal - JIT.

Different Types of JIT

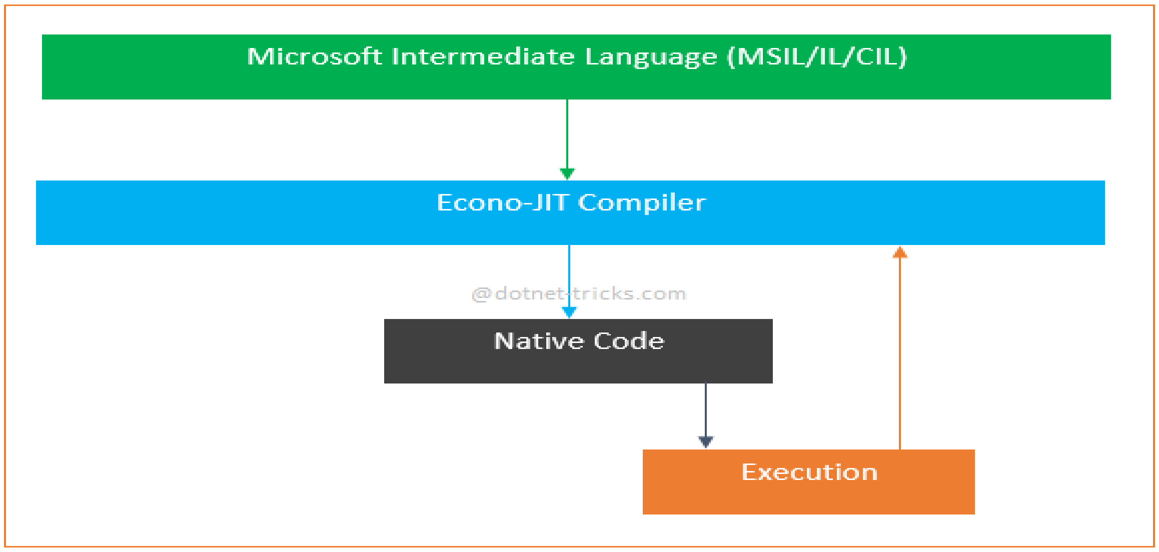
1. Normal JIT

This complies only those methods that are called at runtime. These methods are compiled only first time when they are called, and then they are stored in memory cache. This memory cache is commonly called as JITTED. When the same methods are called again, the complied code from cache is used for execution.



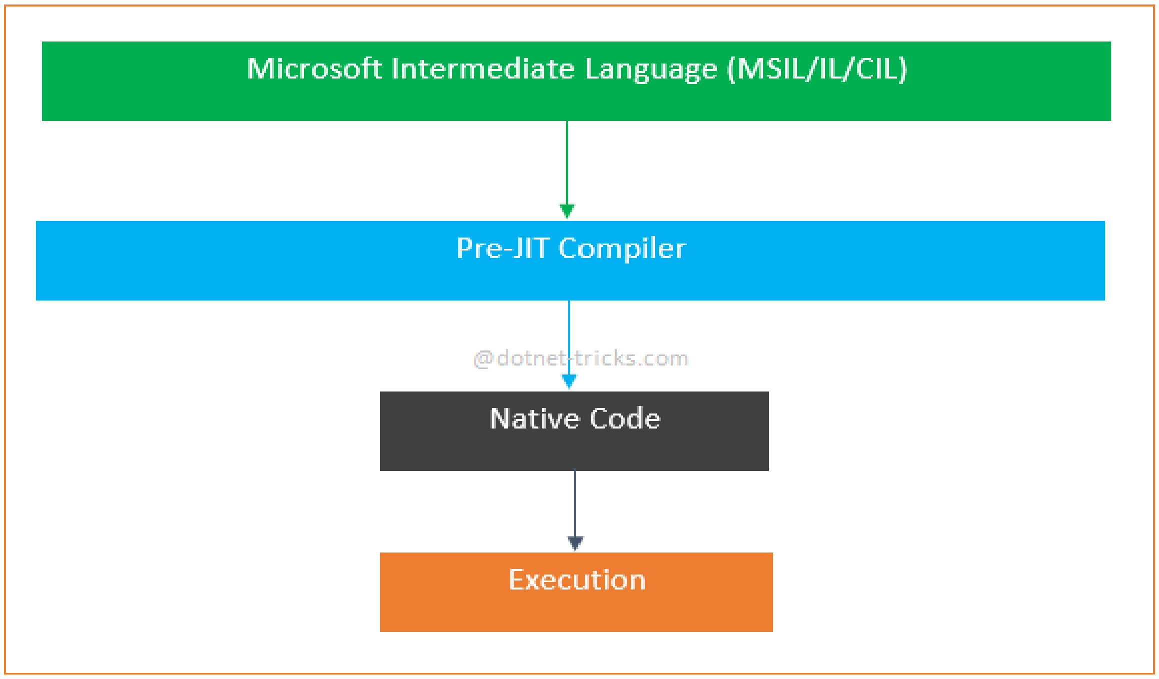
2. Econo JIT

This complies only those methods that are called at runtime and removes them from memory after execution.



3. Pre JIT

This complies entire MSIL code into native code in a single compilation cycle. This is done at the time of deployment of the application.


Microsoft Intermediate Language (MSIL/IL/CIL)
Pre-JIT Compiler
@dotnet-tricks.com
Native Code
Execution