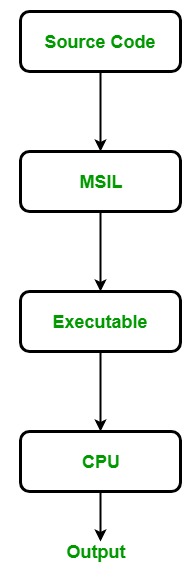
1. Why do we need managed code

# **Managed code and Unmanaged code in .NET**

A code which is written to aimed to get the services of the managed runtime environment execution like [**CLR(Common Language Runtime)**](https://www.geeksforgeeks.org/common-language-runtime-clr-in-c-sharp/) in [.NET Framework](https://www.geeksforgeeks.org/introduction-to-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.

In the case of .NET Framework, the compiler always compiles the manages code in the intermediate language(MSIL) and then create an executable. When the programmer runs the executable, then the [Just In Time Compiler](https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net/) of CLR compiles the intermediate language in the native code which is specific to the underlying architecture. Here this process is taking place under a managed runtime execution environment so this environment is responsible for the working of the code. The execution of managed code is as shown in the below image, the source code is written in any language of .NET Framework.



The managed code also provides platform independence because when the managed code compiled into the intermediate language, then the JIT compiler compiles this intermediate language in the architecture specific instruction.

**What are the advantages of using Managed Code?**

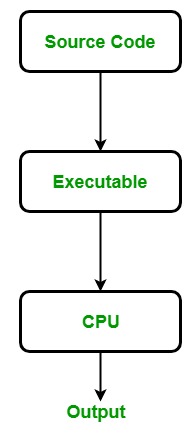
* It improves the security of the application like when you use runtime environment, it automatically checks the memory buffers to guard against buffer overflow.
* It implement the garbage collection automatically.
* It also provides runtime type checking/dynamic type checking.
* It also provides reference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

**What are the disadvantages of Managed Code?**

The main disadvantage of managed language is that you are not allowed to allocate memory directly, or you cannot get the low-level access of the CPU architecture.

#### What is Unmanaged code?

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. Due to this, there are several problems related to memory occur like buffer overflow, memory leak, pointer override, etc. The executable files of unmanaged code are generally in binary images, x86 code which is directly loaded into memory. The application written in VB 6.0, C, C++, etc are always in unmanaged code. The execution of unmanaged code is as shown in the below image:



**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.
* It does not focus on garbage collection.

# **Difference between Managed and Unmanaged code in .NET**

Last Updated: 30-04-2019

**Managed code** is the code which is managed by the CLR(Common Language Runtime) in .NET Framework. Whereas the Unmanaged code is the code which is directly executed by the operating system. Below are some important differences between the Managed code and Unmanaged code:

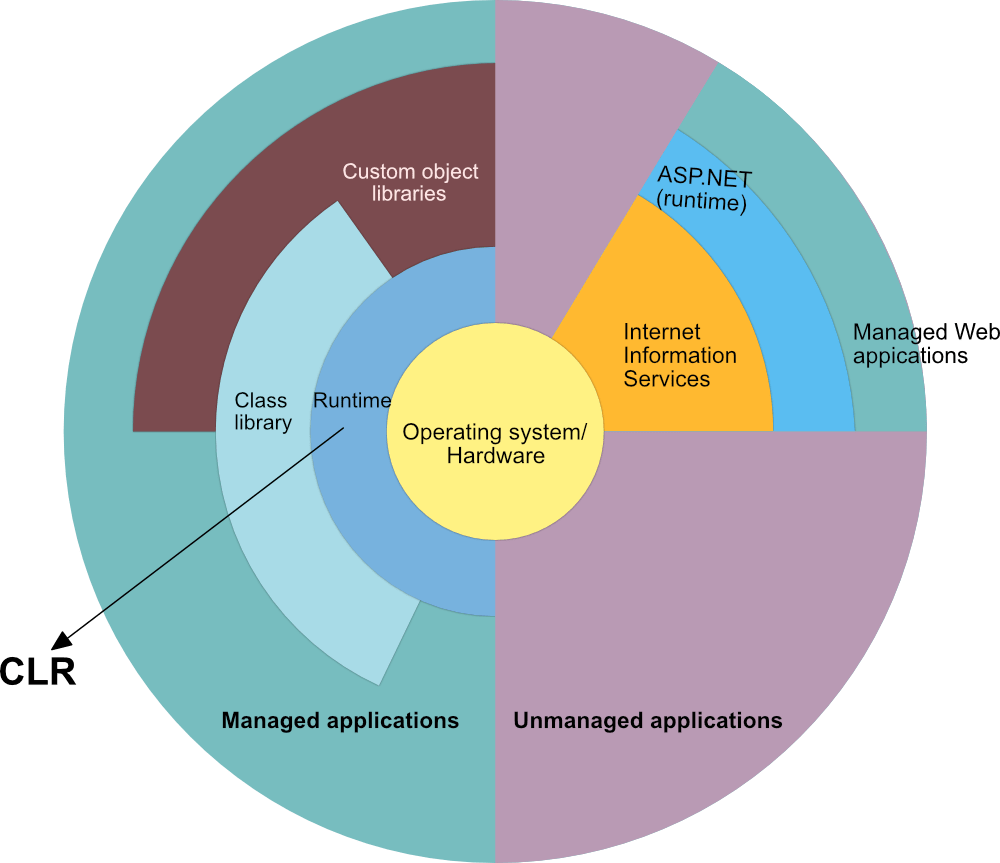
|  |  |
| --- | --- |
| **MANAGED CODE** | **UNMANAGED CODE** |
| It is executed by managed runtime environment or managed by the CLR. | It is executed directly by the operating system. |
| It provides security to the application written in .NET Framework. | It does not provide any security to the application. |
| Memory buffer overflow does not occur. | Memory buffer overflow may occur. |
| It provide runtime services like Garbage Collection, exception handling, etc. | It does not provide runtime services like Garbage Collection, exception handling, etc. |
| The source code is complied in the intermideate language know as IL or MSIL or CIL. | The source code direclty compile into native langugae. |
| It does not provide low-level access to the prgrammer. | It provide low-level access to the prgrammer. |

1. What is CLR/CTS/CLS

# **Common Language Runtime (CLR) in C#**

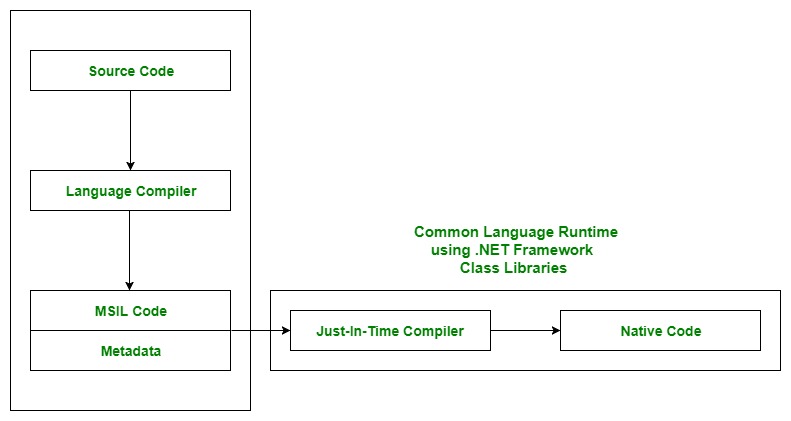
CLR is the basic and Virtual Machine component of the [**.NET Framework**](https://www.geeksforgeeks.org/c-net-framework-basic-architecture-component-stack/). It is the **run-time enviornment in the .NET Framework** that runs the codes and helps in making the development process easier by providing the various services. Basically, it is responsible for managing the execution of .NET programs regardless of any .NET programming language. Internally, CLR implements the VES(Virtual Execution System) which is defined in the Microsoft’s implementation of the CLI(Common Language Infrastructure).  
The code that runs under the Common Language Runtime is termed as the Managed Code. In other words, you can say that CLR provides a managed execution enviornment for the .NET programs by improving the security, including the cross language integration and a rich set of class libraries etc. CLR is present in every .NET framework verison. Below table illustrate the CLR version in .NET framework.

Below diagram illustrate how CLR is associated with the operating system/hardware along with the class libraries. Here, the runtime is actually CLR.

[](https://media.geeksforgeeks.org/wp-content/uploads/Overview-of-the-.NET-Framework-min.png)

#### Role of CLR in the execution of a C# program

* Suppose you have written a C# program and save it in a file which is known as the Source Code.
* Language specific compiler compiles the source code into the **MSIL(Microsoft Intermediate Language)** which is also know as the **CIL(Common Intermediate Language)** or **IL(Intermediate Language)** along with its metadata. Metadata includes the all the types, actual implementation of each function of the program. MSIL is machine independent code.
* Now CLR comes into existence. CLR provides the services and runtime environment to the MSIL code. Internally CLR includes the JIT(Just-In-Time) compiler which converts the MSIL code to machine code which further executed by CPU. CLR also uses the .NET Framework class libraries. Metadata provides information about the programming language, environment, version, and class libraries to the CLR by which CLR handles the MSIL code. As CLR is common so it allows an instance of a class that written in a different language to call a method of the class which written in another language.

[](https://media.geeksforgeeks.org/wp-content/uploads/Working_CLR.jpg)

**Main Components of CLR**

As the word specify, Common means CLR provides a common runtime or execution environment as there are more than 60 .NET programming languages.

Main components of CLR:

* **Common Language Specification (CLS)**
* **Common Type System (CTS)**
* **Garbage Collection (GC)**
* **Just In – Time Compiler (JIT)**

**Common Language Specification (CLS):**  
It is responsible for converting the different .NET programming language syntactical rules and regulations into CLR understandable format. Basically, it provides the Language Interoperability. Language Interoperability means to provide the execution support to other programming languages also in .NET framework.

**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:  
   * **CAS(Code Access Security)**
   * **Exception Handling**
   * **Automatic Memory Management.**
2. **Unmanaged Code:** Before .NET development the programming language like .COM Components & Win32 API do not generate the MSIL code. So these are not managed by CLR rather managed by Operating System.

**Common Type System (CTS):**  
Every programming language has its own data type system, so CTS is responsible for understanding all the data type systems of .NET programming languages and converting them into CLR understandable format which will be a common format.

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 mechanism 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 allots memory for these at Runtime.

**Garbage Collector:**  
It is used to provide the Automatic Memory Management feature. If there was no garbage collector, programmers would have to write the memory management codes which will be a kind of overhead on programmers.

**JIT(Just In Time Compiler):**  
It is responsible for converting the CIL(Common Intermediate Language) into machine code or native code using the Common Language Runtime environment.

**Benefits of CLR:**

* It improves the performance by providing a rich interact between programs at run time.
* Enhance portability by removing the need of recompiling a program on any operating system that supports it.
* Security also increases as it analyzes the MSIL instructions whether they are safe or unsafe. Also, the use of delegates in place of function pointers enhance the type safety and security.
* Support automatic memory management with the help of Garbage Collector.
* Provides cross-language integration because CTS inside CLR provides a common standard that activates the different languages to extend and share each other’s libraries.
* Provides support to use the components that developed in other .NET programming languages.
* Provide language, platform, and architecture independence.
* It allows easy creation of scalable and multithreaded applications, as the developer has no need to think about the memory management and security issues.

1. What is assembly

An assembly is the compiled output of your code, typically a DLL, but your EXE is also an assembly. It's the smallest unit of deployment for any .NET project.

The assembly typically contains .NET code in MSIL (Microsoft Intermediate language) that will be compiled to native code ("JITted" - compiled by the Just-In-Time compiler) the first time it is executed on a given machine. That compiled code will also be stored in the assembly and reused on subsequent calls.

The assembly can also contain resources like icons, bitmaps, string tables and so on. Furthermore, the assembly also contains metadata in the assembly manifest - information like version number, strong name, culture, referenced assemblies and so forth.

In 99% of your cases, one assembly equals a physical file on disk - the case of a multi-file assembly (one assembly, distributed across more than a single file) appears to be a rather odd-ball edge case which I've never encountered so far in my 5+ years of .NET development.

In a multifile assembly there would still be only one assembly manifest in a DLL or EXE and the MSIL code in multiple netmodule files.

What is a .Net Assembly?

The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

An assembly can be a single file or it may consist of the multiple files. In the case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a subpart of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes.