**Dot Net Framework:**

It is a platform for building various applications on windows. It has a list of inbuilt functionalities in the form of class, library, and APIs, which are used to build, deploy and run web services and different applications. It supports different languages such as C#, VB .Net, COBOL, Perl, etc.

**Common Language Runtime (CLR):**

Common Language Runtime (CLR) is a managed execution environment that is part of Microsoft’s .NET framework. CLR manages the execution of programs written in different supported languages.

CLR transforms source code into a form of bytecode known as Common Intermediate Language (CIL). At run time, CLR handles the execution of the CIL code.

Developers write code in a supported .NET language, such as C# or VB.Net. The .NET compiler then converts it into CIL code. During run time, the CLR converts the CIL code into something that can be understood by the operating system. Alternately, the CIL code can be transformed into native code by using the native image generator (NGEN).

The language compilers store metadata that describes the members, types and references in the compiled code. The CLR uses the metadata to lay out instances in memory, locate and load classes, enforce security, set runtime context boundaries, and generate native code.

**Functions of .NET CLR**

* 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**

The key components of CLR includes the following:

* 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 multithreading of our application.
* Exception Handler - It handles exceptions at run time.

**Benefits of .NET CLR**

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, 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.

**Common Type System (CTS):**

The Common Type System (CTS) standardizes the data types of all programming languages using .NET. CTS is a common data type for easy and smooth communication among these .NET languages.

CTS defines a collection of data types, which are used and managed by the run time to facilitate cross-language integration.

CTS is designed as a singly rooted object hierarchy with **System.Object** as the base type from which all other types are derived.

CTS supports two different kinds of types.

1. Value Types
2. Reference Types

**Value Types**:

* Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure.
* Value types can be built-in, user-defined or enumerations types.

**Reference Types**:

* Reference types stores a reference to the value's memory address, and are allocated on the heap.
* Reference types can be self-describing types, pointers types, or interface types.
* The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types are user-defined classes, boxed value types, and delegates.

**Just-In-Time (JIT) Compiler in .NET**

Just-In-Time compiler (JIT) is a part of [Common Language Runtime (CLR)](https://www.geeksforgeeks.org/common-language-runtime-clr-in-c-sharp/) 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.

**Working of JIT Compiler**

The JIT compiler is required to speed up the code execution and provide support for multiple platforms.

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

The JIT compiler converts the Microsoft Intermediate Language (MSIL) or Common Intermediate Language (CIL) into the machine code.

The compiled MSIL or CIL is stored so that it is available for subsequent calls if required.

**Types of Just-In-Time Compiler**

There are three types of JIT compilers, which are as follows.

1. Pre-JIT Compiler
2. Normal JIT Compiler
3. Econo JIT Compiler

**Pre-JIT Compiler:** All the source code is compiled into the machine code at the same time in a single compilation cycle using the Pre-JIT Compiler. This compilation process is performed at application deployment time. In addition, this compiler is always implemented in the **Ngen.exe (Native Image Generator)**.

**Normal JIT Compiler:** The source code methods that are required at run-time are compiled into machine code the first time they are called by the Normal JIT Compiler. After that, they are stored in the cache and used whenever they are called again.

**Econo JIT Compiler:** The source code methods that are required at run-time are compiled into machine code by the Econo JIT Compiler. After these methods are not required anymore, they are removed.

**Advantages of JIT Compiler**:

* The JIT compiler requires less memory usage as only the methods that are required at run-time are compiled into machine code by the JIT Compiler.
* Page faults are reduced by using the JIT compiler as the methods required together are most probably in the same memory page.
* Code optimization based on statistical analysis can be performed by the JIT compiler while the code is running.

**Disadvantages of JIT compiler**:

* The JIT compiler requires more startup time while the application is executed initially.
* The cache memory is heavily used by the JIT compiler to store the source code methods that are required at run-time.

**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 these set of rules are said to be CLS Compliant. In simple words, CLS enables cross-language integration or Interoperability.

**CIL or MSIL | Microsoft Intermediate Language or Common Intermediate Language**

The Microsoft Intermediate Language (MSIL), also known as the Common Intermediate Language (CIL) is a set of instructions that are platform independent and are generated by the language-specific compiler from the source code. The MSIL is platform independent and consequently, it can be executed on any of the Common Language Infrastructure supported environments such as the Windows *.NET* runtime.

* The source code is converted into the MSIL by a language-specific compiler in the compile time of the CLR. Also, along with the MSIL, metadata is also produced in the compilation. The metadata contains information such as the definition and signature of the types in the code, runtime information, etc.
* A Common Language Infrastructure (CLI) assembly is created by assembling the MSIL. This assembly is basically a compiled code library that is used for security, deployment, versioning, etc. and it is of two types i.e. process assembly (EXE) and library assembly (DLL).
* The JIT compiler then converts the Microsoft Intermediate Language(MSIL) into the machine code that is specific to the computer environment that the JIT compiler runs on. The MSIL is converted into the machine code on a requirement basis i.e. the JIT compiler compiles the MSIL as required rather than the whole of it.
* The machine code obtained using the JIT compiler is then executed by the processor of the computer.

**Managed Code**

The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.

Managed code uses CLR which in turns looks after your applications by managing memory, handling security, allowing cross - language debugging, and so on.

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 check, they are not duplicate.

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.

**Unmanaged Code**

The code, which is developed outside .NET, Framework is known as unmanaged code.

Applications that do not run under the control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write such applications, which, for example, access low - level functions of the operating system. Background compatibility with code of VB, ASP and COM are examples of unmanaged code.

* Unmanaged code can be unmanaged source code and unmanaged compile code.
* Unmanaged code is executed with help of wrapper classes.
* Wrapper classes are of two types: **CCW (COM Callable Wrapper)** and **RCW (Runtime Callable Wrapper).**

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.

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.

**Garbage Collection**

Automatic memory management is made possible by **Garbage Collection in .NET Framework**. When a class object is created at runtime, certain memory space is allocated to it in the heap memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

Garbage collection will always work on Managed Heap and internally it has an Engine which is known as the Optimization Engine.

Garbage Collection occurs if at least one of multiple conditions is satisfied. These conditions are given as follows:

* If the system has low physical memory, then garbage collection is necessary.
* If the memory allocated to various objects in the heap memory exceeds a pre-set threshold, then garbage collection occurs.
* If the GC.Collect method is called, then garbage collection occurs. However, this method is only called under unusual situations as normally garbage collector runs automatically.

There are mainly **3** phases in garbage collection

1. **Marking Phase:** A list of all the live objects is created during the marking phase. This is done by following the references from all the root objects. All of the objects that are not on the list of live objects are potentially deleted from the heap memory.
2. **Relocating Phase:** The references of all the objects that were on the list of all the live objects are updated in the relocating phase so that they point to the new location where the objects will be relocated to in the compacting phase.
3. **Compacting Phase:** The heap gets compacted in the compacting phase as the space occupied by the dead objects is released and the live objects remaining are moved. All the live objects that remain after the garbage collection are moved towards the older end of the heap memory in their original order.

The heap memory is organized into 3 generations so that various objects with different lifetimes can be handled appropriately during garbage collection. The memory to each Generation will be given by the [Common Language Runtime(CLR)](https://www.geeksforgeeks.org/common-language-runtime-clr-in-c-sharp/) depending on the project size. Internally, Optimization Engine will call the *Collection Means Method* to select which objects will go into Gneration 1 or Generation 2.

**Generation 0 :**

All the short-lived objects such as temporary variables are contained in the generation 0 of the heap memory. All the newly allocated objects are also generation 0 objects implicitly unless they are large objects. In general, the frequency of garbage collection is the highest in generation 0.

**Generation 1 :**

If space occupied by some generation 0 objects that are not released in a garbage collection run, then these objects get moved to generation 1. The objects in this generation are a sort of buffer between the short-lived objects in generation 0 and the long-lived objects in generation 2.

**Generation 2 :**

If space occupied by some generation 1 objects that are not released in the next garbage collection run, then these objects get moved to generation 2. The objects in generation 2 are long lived such as static objects as they remain in the heap memory for the whole process duration.

Note**:**

Garbage collection of a generation implies the garbage collection of all its younger generations. This means that all the objects in that particular generation and its younger generations are released. Because of this reason, the garbage collection of generation 2 is called a full garbage collection as all the objects in the heap memory are.released. Also, the memory allocated to the Generation 2 will be greater than Generation 1’s memory and similarly the memory of Generation 1 will be greater than Generation 0’s memory(**Generation 2 > Generation 1 > Generation 0**).

**Methods**:

GC.MaxGeneration -> To find the maximum number of generations that are **supported by the system.**

GC.GetGeneration(obj)-> To find the no of generation of object.

GC.GetTotalMemory() -> To find the total no of memory.

GC.Collect() -> Garbage collection can be forced in the system.

**Benefits of Garbage Collection**

* Garbage Collection succeeds in allocating objects efficiently on the heap memory using the generations of garbage collection.
* Manual freeing of memory is not needed as garbage collection automatically releases the memory space after it is no longer required.
* Garbage collection handles memory allocation safely so that no objects use the contents of another object mistakenly.
* The constructors of newly created objects do not have to initialize all the data fields as garbage collection clears the memory of objects that were previously released.

**Assembly:**

Assembly is the smallest unit of deployment of a .net application. It can be a **dll** or an **exe.**

It is portable and executable.

The different parts of an Assembly are:

* **Manifest** – It contains the information about the version of an assembly. It is also called as assembly metadata.
* **Type Metadata** – Binary information of the program.
* **MSIL** – Microsoft Intermediate Language code.
* **Resources** – List of related files.

In .Net 3 types of Assemblies are available

1. **Private Assembly:**

The dll or exe which is sole property of one application only. It is generally stored in application root folder

1. **Public/Shared assembly:**

It is a dll which can be used by multiple applications at a time. A shared assembly is stored in **GAC** i.e [Global Assembly Cache](http://msdn.microsoft.com/en-us/library/yf1d93sz).

GAC is simply **C:\Windows\Assembly** folder where you can find the public assemblies/dlls of all the softwares installed in your PC.

1. [**Satellite Assembly**](http://blogs.msdn.com/b/global_developer/archive/2011/07/22/introduction-to-satellite-assemblies.aspx)**.**

A Satellite Assembly contains only static objects like images and other non-executable files required by the application.

**GAC**:

GAC stands for**Global Assembly Cache**. Whenever CLR gets installed on the machine, GAC comes as a part of it. GAC specifically stores those assemblies which will be shared by many applications. A Developer tool called Gacutil.exe is used to add any file to GAC.

**Globalization and Localization:**

Internationalization is the process of designing applications that support multiple languages. This is divided into **Localization and Globalization**.

Globalization is nothing but developing applications to support different languages. Existing applications can also be converted to support multiple cultures.

Whereas Localization means changing the already globalized app to cater to a specific culture or language Microsoft.Extensions.Localization is used for localizing the app content. Some of the other keywords that are used for Localization are IHtmlLocalizer, IStringLocalizer, IViewLocalizer and so on.