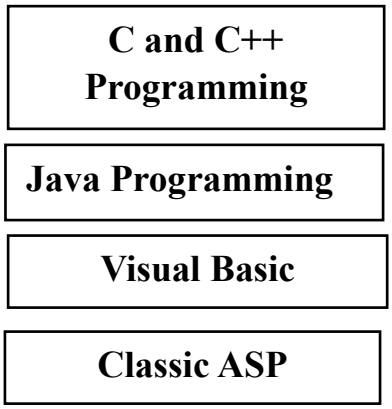
Unit I: Introduction to .NET Framework: NET framework, MSIL, CLR, CLS, CTS, Namespaces, Assemblies The Common Language Implementation, Assemblies, Garbage Collection, The End to DLL Hell – Managed Execution

---

Introduction



To develop the different kinds of application using different programming languages, we required to learn all these languages (as shown above). Putting all the syntaxes in our brain which cause little bit confusion. So, all the problem addressed with .NET. In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#(C sharp), VB.Net, etc.

It is used to develop Form-based applications, Web-based applications, and Web services. There is a variety of programming languages available on the .Net platform, VB.Net and C# being the most common ones. It is used to build applications for Windows, mobile, web, etc.

.NET Framework

.NET Framework is a software development platform develop by Microsoft. The framework was mean to create an application which would run on Windows platform. The first version of .NET framework is 1.0 which was released in the year 2002. And the latest version of .NET framework is 4.8.1 which was released in August 2022. .NET framework can be used to create form based, console-based Application and web-based application. Framework also supports various programming languages such as Visual Basic, C#, ASP, J#.

.NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft. The remaining Non-Microsoft Languages which are supported by .NET Framework but not designed and developed by Microsoft.

11 Programming Languages which are designed and developed by Microsoft are:

- C#.NET
- VB.NET
- C++.NET
- J#.NET
- F#.NET
- JSCRIPT.NET
- WINDOWS POWERSHELL
- IRON RUBY
- IRON PYTHON
- C OMEGA
- ASML(Abstract State Machine Language)

Programmer can choose from a various programming language available on the .NET platform. The most common ones are VB.NET and C#.NET.

Features of .NET

1. Platform independent

Platform independent means it supports **cross platform** that works on multiple platforms. Whenever we make .NET program, we use windows operating system. In windows operating system we have different versions such as (window 10, window 8, window 9 etc). Suppose, if we develop programming code in window 11 and the same code try to run on different version like in window 10 operating system it will work perfectly fine. So, we can say that **it is interoperable between multiple windows operating system**.

2. Language Independent

As Programmer can choose any language from a various programming language available on the .NET platform. So, if I have the code of one language. I can just work with other language as well. We can say code written in one language can be used in other language. Suppose program is develop in VB.NET user can easily reuse that code in C#.NET as well. That means user can **reuse the code from one language to another**. So, as we don't need to write code again and again so **it improves efficiency and performance will be faster** with the help of different languages.

3. Automatic management of resources

It automatically manages the resources(file, memory, database etc). In .NET we have garbage collector that **periodically checks for unused objects**

(memory space) in our code, and automatically free up that space if it is no longer needed in the program.

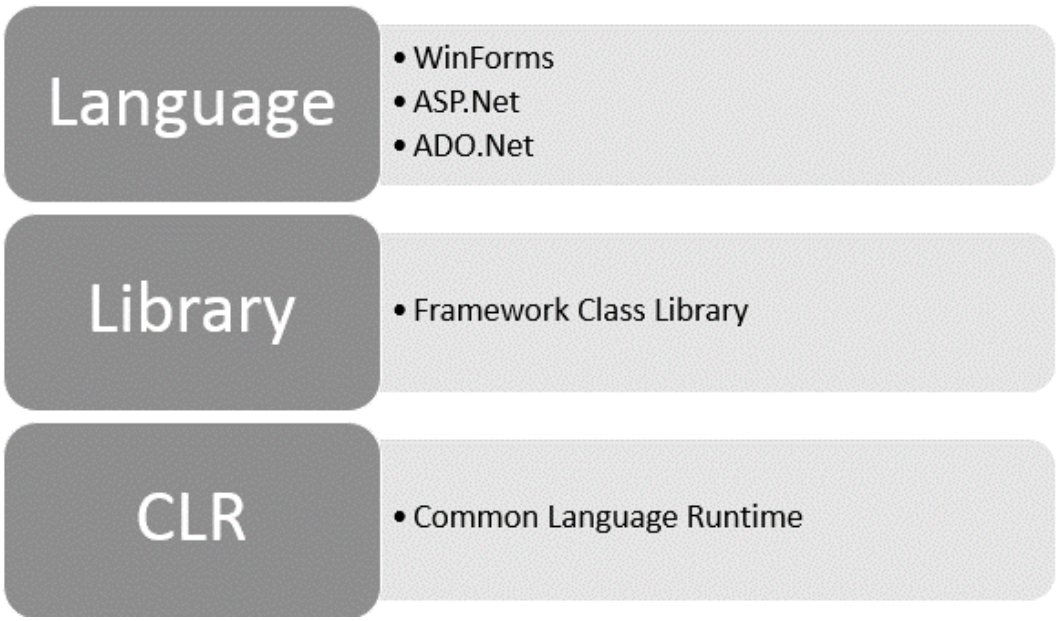
4. Consistent programming model

.NET is the pure object oriented programming language. .NET is a framework inside we are having multiple languages like VB.NET, C#.NET(C sharp.NET) etc and all these languages having object oriented features. So, we can say that it is having consistent object oriented programming model across all languages. We know that object oriented contain features like i) abstraction ii) encapsulation iii) polymorphism iv)inheritance.

5. Easy database connectivity

ADO.NET is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server. So, .NET can connect easily with database using ADO.NET technology.

Architecture/Components of .NET Framework



[Fig: Architecture/Components of .NET Framework]

The architecture of the .Net framework is based on the following key components;

1. Common Language Runtime

The "Common Language Infrastructure" or CLI is a platform on which the .Net programs are executed. The CLI has the following key features:

Exception Handling - Exceptions are errors which occur when the application is executed. Examples of exceptions are:

- If an application tries to open a file on the local machine, but the file is not present.
- If the application tries to fetch some records from a database, but the connection to the database is not valid.

Garbage Collection - Garbage collection is the process of removing unwanted resources when they are no longer required. Examples of garbage collection is :

A File handle which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.

2. Class Library

The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.

For example, there is a class library with methods to handle all file-level operations. So, there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

3. Languages

The types of applications that can be built in the .Net framework is classified broadly into the following categories.

WinForms – This is used for developing Forms-based applications, which would run on an end user machine. Notepad is an example of a client-based application.

ASP.Net – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome or Firefox.

The Web application would be processed on a server, which would have Internet Information Services Installed.

Internet Information Services or IIS is a Microsoft component which is used to execute an Asp.Net application.

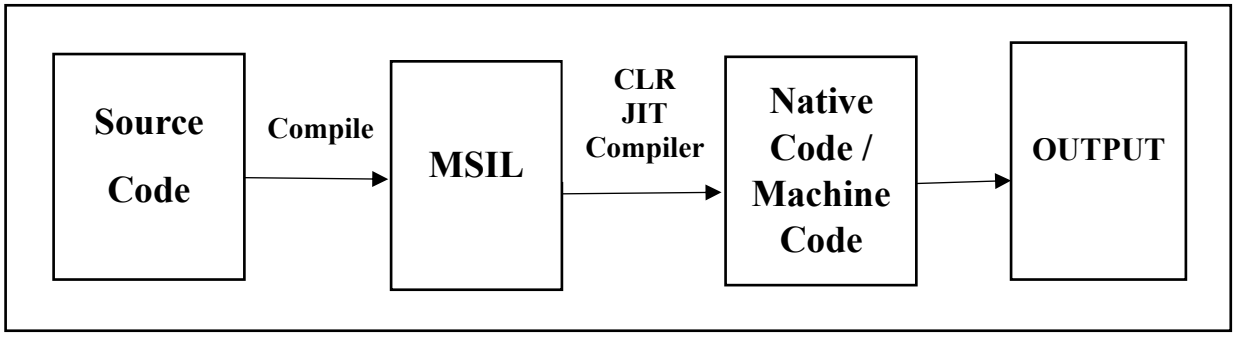
The result of the execution is then sent to the client machines, and the output is shown in the browser.

ADO.Net – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

MSIL

A .NET programming language (C#, VB.NET, J# etc.) does not compile into executable code; instead, it compiles into an intermediate code called Microsoft Intermediate Language (MSIL). As a programmer one need not worry about the syntax of MSIL - since our source code in automatically converted to MSIL. The MSIL code is then send to the CLR (Common Language Runtime) that converts the code to machine language which is then run on the host machine. Once we have compiled our code to MSIL CLR accept the MSIL code then again recompilation is done CLR contains JIT(Just in time) compiler to recompiled MSIL code into native code. **MSIL is similar to Java Byte code**. A Java program is compiled into Java Byte

code (the .class file) by a Java compiler, the class file is then sent to JVM which converts it into the host machine language.

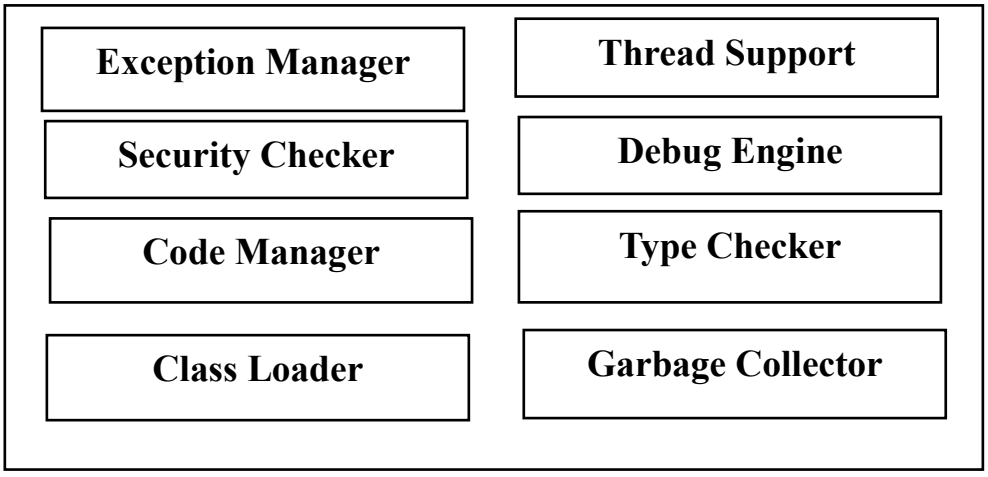


[Fig : Process of compilation converting IL into machine code]

CLR(Common Language Runtime)

CLR is the heart of .NET framework. **CLR is similar to the JVM(Java Virtual Machine) in java**. Whenever we write code that code is converted into intermediate language after first compilation. And this intermediate language is executed on CLR(Common Language runtime).So, we need CLR to execute the intermediate language. Also, whole program is monitored by CLR.

Components of CLR



[Fig : Components of CLR]

1. Code Manager

As name signify code manager means managing code. Code manager manage code during execution i.e., when we run our code. It also allocates memory to objects.

2. Type Checker

As name signify type checker means checking type but which type. It checks data type of variable. For example, we declare a variable as follows:

Dim a As Integer

Dim b as Double

Dim c as String

So, checking data type of these variables a, b and c is taken care by type checker.

3. Debug Engine

Debugging means finding the errors. Debug engine find and remove the bugs(errors) into the code.

4. Class Loader

Class loader loads data in runtime. We can say that class loader loads all libraries(collection of classes, methods, functions, interfaces) which are associated in the intermediate language code.

5. Exception Manager

Errors are classified into two categories

1. Compile-Time errors
2. Run-Time errors

All the syntax error done in program i.e., mistakes that is done during developing or typing program is called **compile-time error**.

Sometimes program run successfully but it terminated or may cause system to crash i.e., logically program is correct but error comes during execution

of program called **run time error**. So, to manage the normal flow of program cause by run time error CLR contains exception manager. Ex : Division by 0.

It manages the exception in both **managed** and **unmanaged** code.

* Note :

1. Whenever user convert source code to the intermediate language this is the **managed code**.
2. Whenever user convert source code directly to machine code this is **unmanaged code.**

6. Thread support

When user open multiple independent program(like Game, Music app, paint app) on window called as multiprogramming.

When user open multiple tabs on let's say in browser(google chrome) which means same type of tasks perform together (like opening Gmail, Facebook, translator etc) called multithreading. So, .NET supports thread which means multiple application can be created at a same time on same environment.

7. Security checker

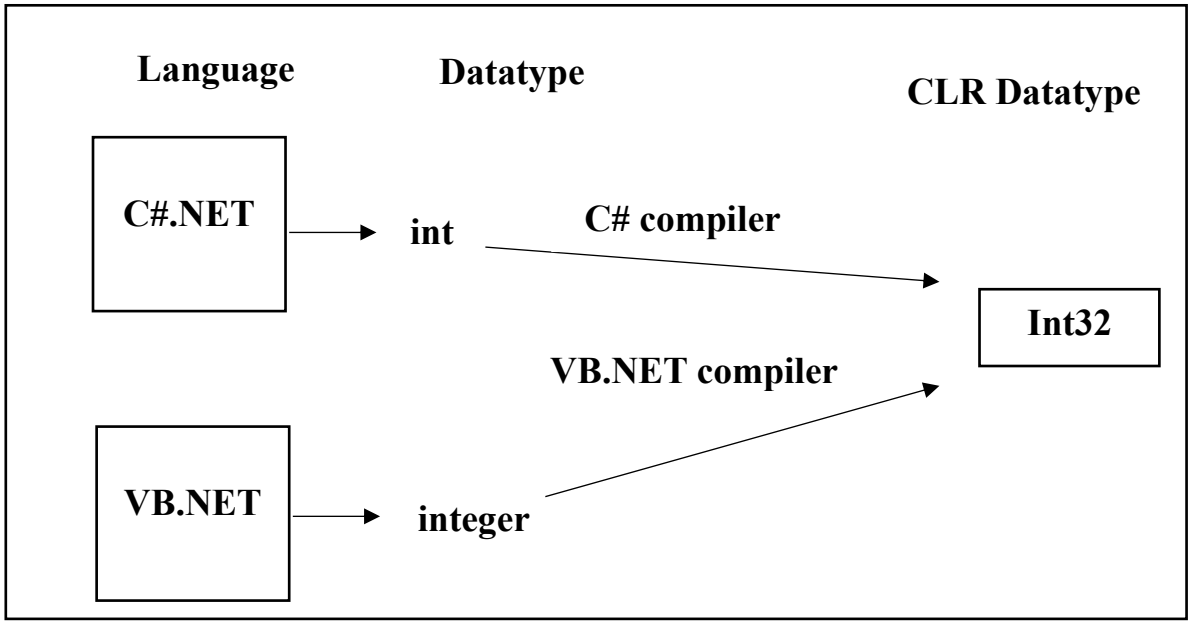
When an application has database where all files of user are saved like some user id, passwords etc. and that data cannot be access by outside world. So, that task is of security checker which resources should be access by which user i.e., anybody cannot access everything.

8. Garbage collector

In .NET we have garbage collector that **periodically checks for unused objects (memory space)** in our code, and automatically free up that space if it is no longer needed in the program.

CTS

CTS stands for Common type System. It deals with the data type. So here we have several languages and each and every language has its own data type and one language data type cannot be understandable by other languages but .NET Framework language can understand all the data types. C# has an **int** data type and VB.NET has **Integer** data type. Hence a variable declared as an int in C# and Integer in VB.NET, finally after compilation, uses the same structure Int32 from CTS.



[Fig : CTS(Common Type System)]

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.

For Example

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(;).

Explanation of the above Example

So, these syntax rules which you have to follow from language to language differ but CLR can understand all the language Syntax because in .NET each language is converted into MSIL code after compilation and the MSIL code is language specification of CLR.

JIT (Just in time Compiler)

The MSIL is the language that all of the .NET languages compile down to. After they are in this intermediate language, a process called Just-In-Time (JIT) compilation occurs when resources are used from your application at runtime. JIT allows “parts” of your application to execute when they are needed, which means that if something is never needed, it will never compile down to the native code. By using the JIT, the CLR can cache code that is used more than once and reuse it for subsequent calls, without going through the compilation process again.

The JIT process enables a secure environment by making certain assumptions:

- Type references are compatible with the type being referenced.
- Operations are invoked on an object only if they are within the execution parameters for that object.
- Identities within the application are accurate.

By following these rules, the managed execution can guarantee that code being executed is type safe; the execution will only take place in memory that it is allowed to access. This is possible by the verification process that occurs when the MSIL is converted into CPU-specific code. During this

verification, the code is examined to ensure that it is not corrupt, it is type safe, and the code does not interfere with existing security policies that are in place on the system.

Namespaces

Namespaces help you to create logical groups of related classes and interfaces that can be used by any language targeting the .NET framework. Namespaces allow you to organize your classes so that they can be easily accessed in other applications. Namespace can also be used to avoid any naming conflicts between classes that have the same names. For example, you can use two classes with the same name in an application provided they belong to different namespaces. You **can access** the classes belonging to a namespace **by simply importing** the namespace into an application. The .NET framework uses a **dot (.) as a delimiter(separator)** between **classes and namespaces**.

Ex. import System.Windows.Forms represents the windows form button class of the System namespace.

System is the basic namespace used by every .NET code. There are lots of namespace user the system namespace.

For example,

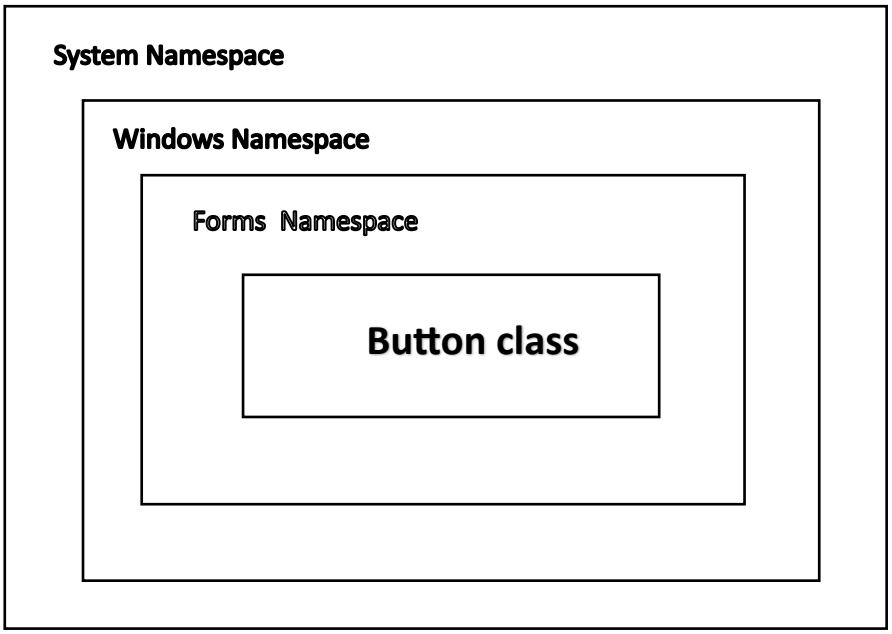
System.IO,

System.Net,

System.Collections,

System.Threading,

System.Drawing, etc.



[Fig : Namespace Representation]

Some inbuilt namespaces are :

System: Includes essential classes and base classes for commonly used data types, events, exceptions and so on.
System.Collections: Includes classes and interfaces that define various collection of objects such as list, queues, hash tables, arrays, etc.
System.IO: Includes classes for data access with Files.
System.Threading: Includes classes and interfaces to support multithreaded applications.
System.Windows.Forms: Includes classes for creating Windows based forms.
System.Web: Includes classes and interfaces that support browser-server communication.
System.Web.Services: Includes classes that let us build and use Web Services.

User can define own namespace as follows :

```
Namespace namespaceName
```

Syntax :

```
Namespace Namespace1
    Public Class class1
        class code
    End Class
End Namespace
```

Ex :

```
Namespace College
    Namespace students
        imports System
            Class Addstudents
                Public Function S_Method()
Console.WriteLine("Adding Students to college using S_Method!")
                End Function
            End Class
    End Namespace
End Namespace
```

In above example we create namespace of College, inside namespace of college we create namespace of students in that we import System namespace and inside student namespace we create class of Addstudents to

add new students in college. In this way we create an hierarchy of namespace and create class. This namespace we can import and reuse in any other application.

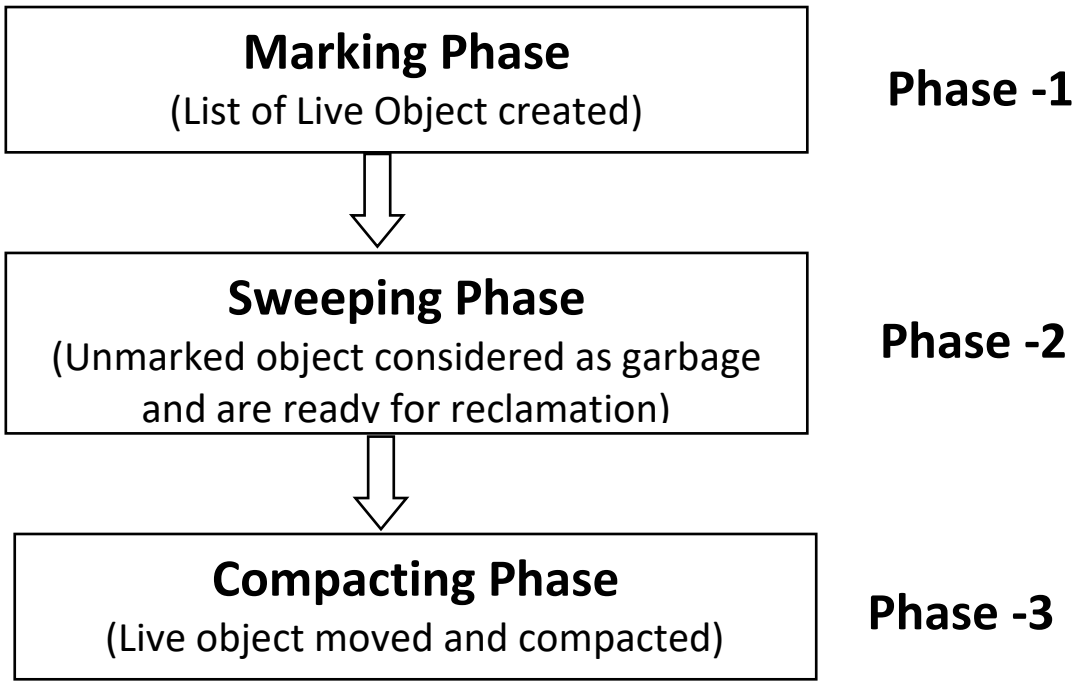
Garbage Collection

Garbage collection is a fundamental aspect of memory management in the .NET Framework. It is an automatic process that relieves developers from the burden of manual memory allocation and deallocation, allowing them to focus on application logic rather than memory management details.

The garbage collector provides the following benefits:

- Frees developers from having to manually release memory.
- Allocates objects on the managed heap efficiently.
- Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Managed objects automatically get clean content to start with, so their constructors don't have to initialize every data field.
- Provides memory safety by making sure that an object cannot use the content of another object.

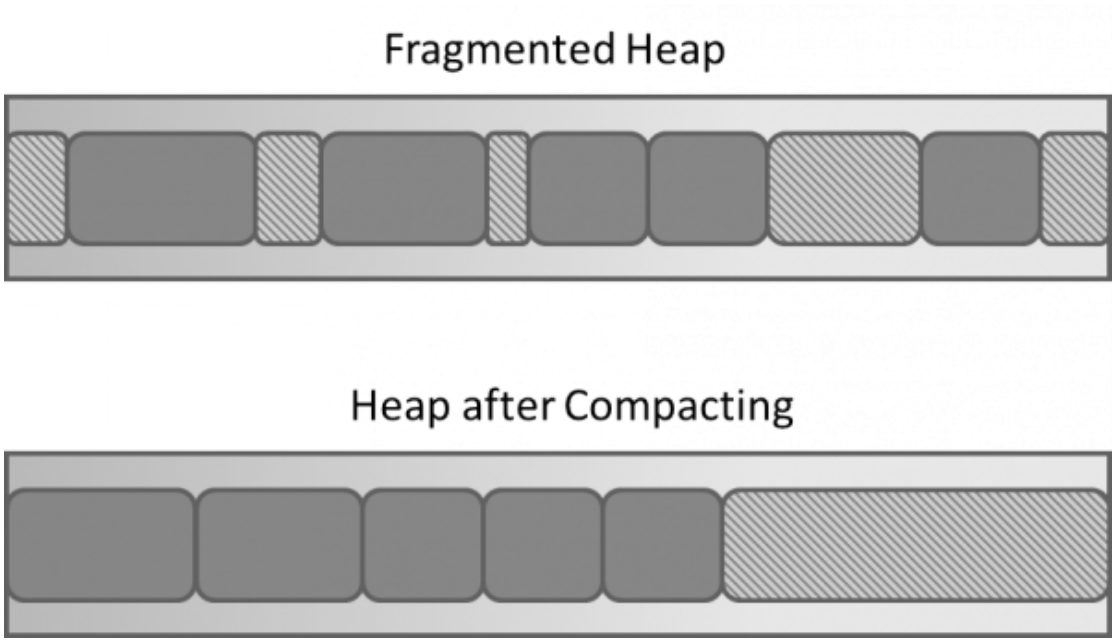
Phases in Garbage Collection



Marking Phase: During a garbage collection cycle, the GC starts by marking all reachable objects. It traverses the object graph, starting from the roots (global variables, references on the stack, etc.), marking each reachable object as "in-use".

Sweeping Phase: Once the marking phase is complete, the GC sweeps through the managed heap, identifying all the objects that were not marked during the marking phase. These unmarked objects are considered garbage and are ready for reclamation.

Compact Phase: In the final phase, the garbage collector reclaims the memory occupied by the garbage objects. It compacts the live objects, moving them together to eliminate memory fragmentation, and updates the heap's free space (as shown in fig below).



Heap Generations in Garbage Collection

The heap memory is organized into 3 generations so that various objects with different lifetimes can be handled appropriately during garbage collection.

Generation 0

Generation 1

Generation 2

Heap Memory

Generation - 0
Generation – 1
Generation – 2

Example :

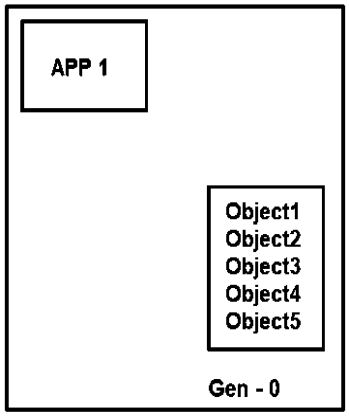


Fig 1 : newly created object store in heap Gen-0.

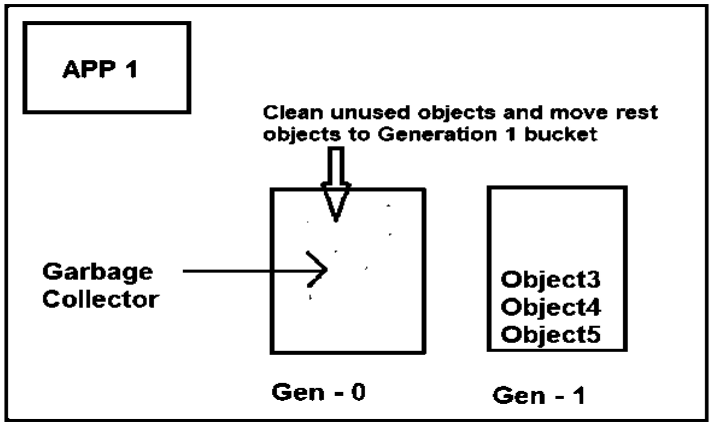


Fig 2 : survive in memory for short time store in heap Gen-1.

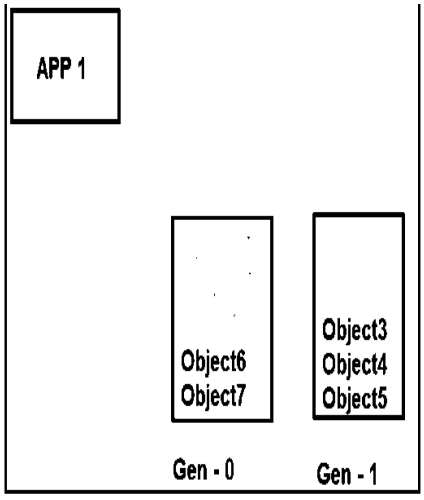


Fig 3 : after revisit GC will store newly created object in heap Gen-0.

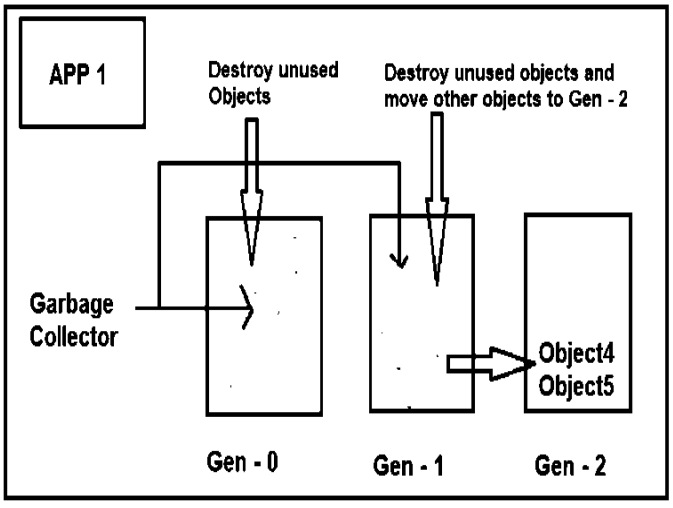


Fig 4 : survive in memory for longer time will store in heap Gen-2.

When we are working with an application, inside that we create a lot of objects. There may be programmer will create unused objects inside an application that are not in used during execution of program. So, for that purpose we need a garbage collector to collect an unused object in our application. Consider an above fig where we create one application name as APP 1. Inside that application we create an objects i.e., object1, object2, object3, object4, object5. In the .NET Framework, the garbage collector (GC) is responsible for reclaiming memory that is no longer in use by the application. It tracks and manages objects allocated on the managed heap, which is a dedicated portion of memory allocated by the runtime for storing objects. The garbage collector uses a generational approach to manage memory. Objects are categorized into different generations based on their age. The heap is divided into three generations: 0, 1, and 2. Newly created objects are allocated in Generation 0 (as shown in fig 1, fig 3) . If an object survives for short lifetimes in a garbage collection, it gets promoted to the next generation i.e., Gen 1 (as shown in fig 2).

The garbage collector employs various algorithms to determine which objects are eligible for collection. One such algorithm is the **mark-and-sweep algorithm.** It starts by marking all objects that are reachable from the root of the object graph, typically starting with global and local variables. Any objects that are not marked as reachable are considered garbage and can be safely collected. The sweep phase then reclaims the memory occupied by the garbage objects, making it available for future allocations. As shown in fig 4, which cleans all unused objects from App 1 and moves all use objects which will stay in memory for longer time to gen 3.

The garbage collector also includes a compaction phase, where it rearranges the objects in memory to reduce fragmentation and improve memory

locality. Compaction involves moving live objects closer together, eliminating fragmented gaps and improving memory access efficiency.

Need of heap generation

- Normally, when we are working with big applications, they can create thousands of objects. So, for each of these objects, if the garbage collector goes and checks if they are needed or not.

- it's really a bulky process. By creating such generations what it means if an object in Generation 2 buckets it means the Garbage Collector will do fewer visits to this bucket.

- The reason is, if an object move to Generation 2, it means it will stay more time in the memory. It's no point going and checking them again and again.

Assemblies

Assemblies are the fundamental units of deployment, version control, reuse, and security permissions for .NET-based applications. An assembly is a **collection of types and resources** that are built to work together and form a logical unit of functionality. Assemblies **take the form of executable (*.exe*) or dynamic link library (*.dll*) files**, and are the building blocks of .NET applications. They provide the common language runtime with the information it needs to be aware of type implementations.

In .NET and .NET Framework, you can build an assembly from one or more source code files. In .NET Framework, assemblies can contain one or more modules. This way, larger projects can be planned so that several

developers can work on separate source code files or modules, which are combined to create a single assembly.

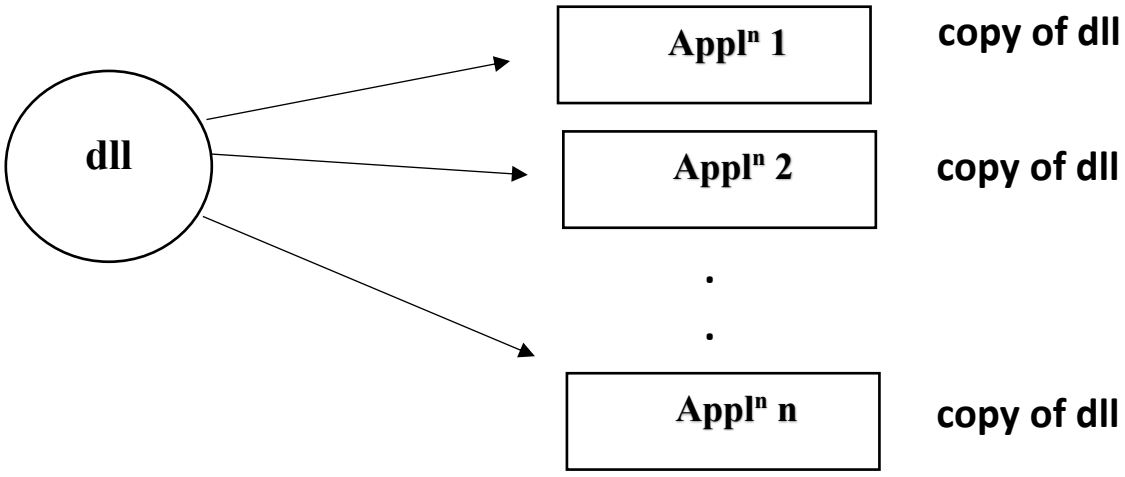
Assemblies have the following properties:

- Assemblies are implemented as *.exe* or .dll files.
- For libraries that target .NET Framework, you **can share assemblies** between applications **by putting them** in **the global assembly cache (GAC).** You must strong-name assemblies before you can include them in the GAC.

Assembly classified into two types :

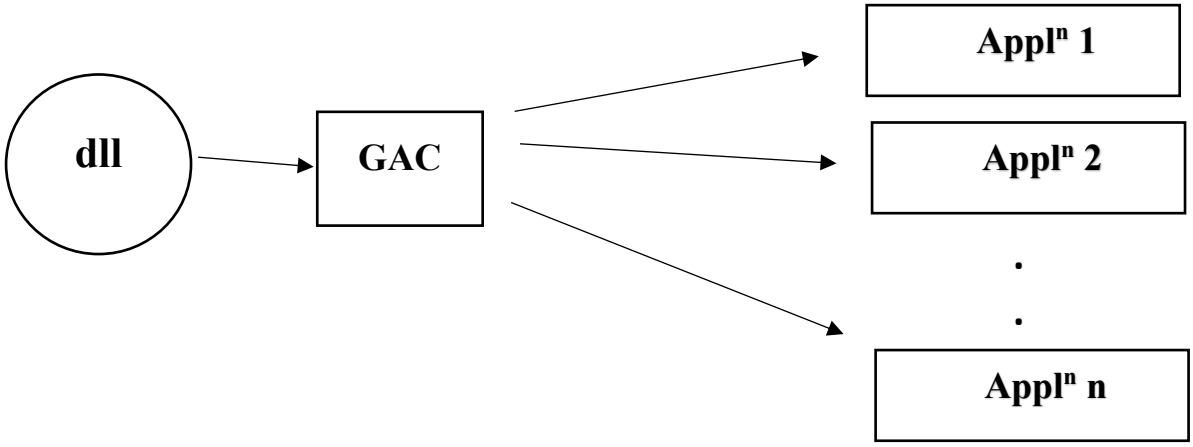
1. Private Assembly

An Assembly that is solely **used by one application** is referred to as a Private Assembly. It is typically found in its own folder or directory. Private Assemblies are not intended to be shared with other applications. They are used to store application- directory specific code and resources. **dll location is in each application of bin folder**.

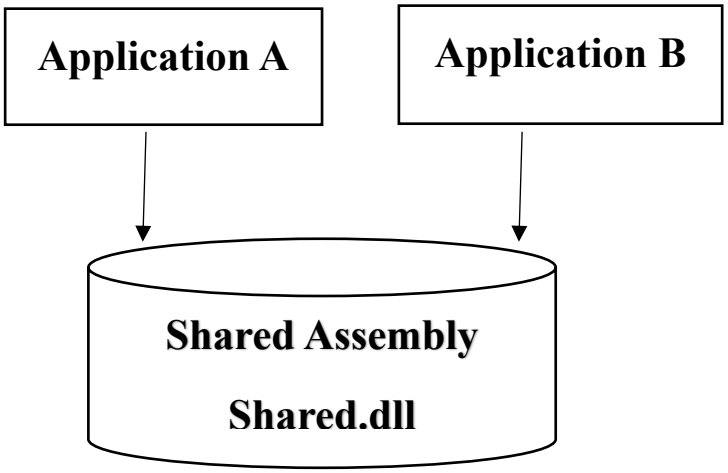


2. Public assembly

An assembly that is **used by several programmes** is referred to as a shared assembly. It is typically **found in** the **Global Assembly Cache (GAC)** or a common directory. Multiple applications are supposed to share a shared assembly. They are used to store resources and code that are shared by various applications. Shared Assemblies are created using the strong name tool ***(sn.exe)***. The **GAC** location is in the Windows directory ***(C:\Windows\assembly)***.



End to dll hell-managed execution



[Fig : Shared.dll Assembly]

From above fig. we can understand that two applications A and B, use the same shared assembly.

Now a few scenarios occur during production; they are:

1. We have two applications, A and B installed, both of them installed on our PC.
2. Both of these applications use the same shared assembly SharedApp.dll.
3. Somehow, we have the latest version of SharedApp.dll installed on our PC.
4. The latest SharedApp.dll replaces the existing DLL, that App A was also using earlier.
5. Now App B works fine whereas App A doesn't work properly due to the newer SharedApp.dll.

In short, a newer version of a DLL might not be compatible with an older application. Here SharedApp.dll is a newer version that is not backward compatible with App A. So, **DLL HELL is the problem where one application will install a newer version of a shared component that is not backward compatible with the version already on the machine**, **causing other existing applications that rely on the shared component to break**. With .NET versioning, we don't have the DLL Hell problem anymore.

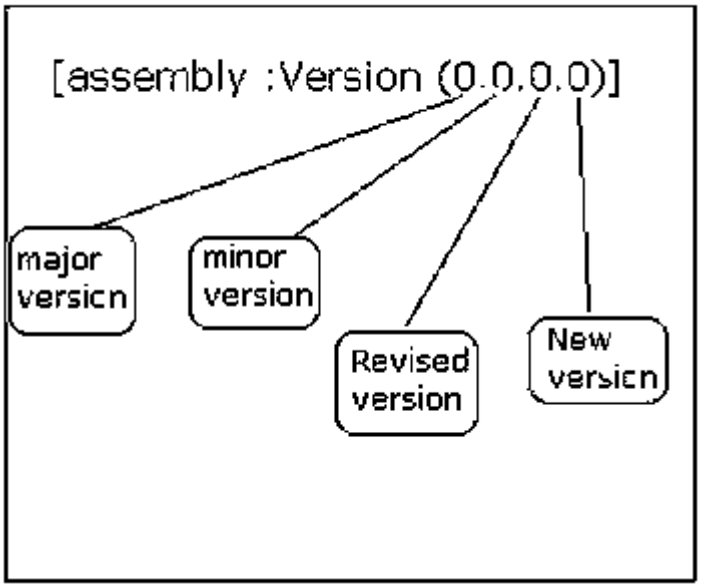
Now the resolution of this is after introducing Versioning in .Net with shared assemblies. Which is placed in the GAC (Global Assembly Cache). Its path is usually "C:\Windows\assembly".

The GAC contains strong-named assemblies. Strong-named assemblies in .NET have 4 pieces in their name as listed below.

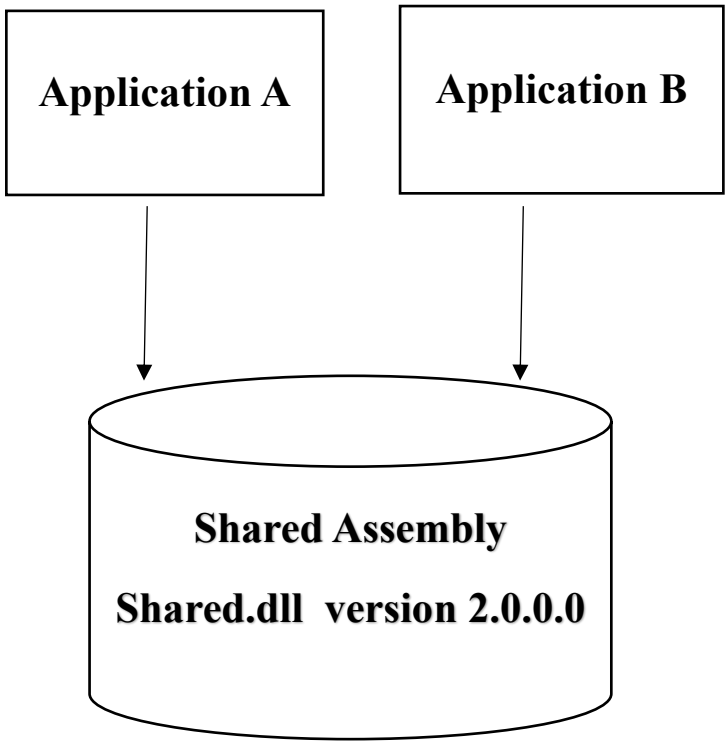
1. Name of assembly
2. Version Number
3. Culture

4. Public Key Token

Each DLL has its own version number that describes it as in the following:



[Fig : Assembly versioning]



[Fig : Shared.dll with 2.0.0.0 version]

Now recall the DLL Hell problem with the newer face that uses versioning, as described in:

1. We have two applications, A and B installed, both of them installed on our PC.
2. Both of these applications use the shared assembly SharedApp.dll having version 1.0.0.0.
3. Somehow, we have the latest version (2.0.0.0) of SharedApp.dll installed in the GAC.
4. So, in the GAC we now have 2 versions of SharedApp.dll
5. Now App A uses its old DLL with version 1.0.0.0 and App B works perfectly with SharedApp.dll version 2.0.0.0.

So, .Net DLL versioning helped to resolve this DLL Hell problem.