ASSEMBLY IN .NET

What is an assembly?

The major components .NET framework are:

1. Common Language Runtime (CLR)
2. Framework Class Library (FCL)
3. Base Class Library (BCL)
   It is divided into two parts
   1. User defined class library: It includes **ASSEMBLIES.**
      Assembly: A .NET assembly is considered as the major building block of the .NET Framework. An assembly in the CLI (Common Language Infrastructure) is a logical unit of code, which is used for security, deployment, and versioning. Assembly can be defined in two forms namely Dynamic Link Library(.dll) and executable(.exe) files.

      When compilation of the .NET program takes place, metadata with Microsoft Intermediate Language (MSIL) will be generated and will be stored in a file called Assembly.

   2. Pre-defined class library: It contains NAMESPACE.

4. Common Type System (CTS)
5. Common Language Specification (CLS)

An assembly is a file that is automatically generated by the compiler which consists of a collection of types and resources that are built to work together and form a logical unit of functionality. We can also say, assembly is a compiled code and logical unit of code.

Assemblies are implemented in the form of executable (.exe) or dynamic link library (.dll) files.

Explain the different types of assembly.

Assemblies are classified into 2 types. They are:

Private Assembly:

- It is accessible only to the application.
- We need to copy this private assembly, separately in all application folders where we want to use that assembly. Without copying, we cannot access the private assembly.
- It requires to be installed in the installation directory of the application.

Shared or Public Assembly:

- It can be shared by multiple applications.
- Public assembly does not require copying separately into all application folders. Only one copy of public assembly is required at the system level, we can use the same copy by multiple applications.
- It is installed in the Global Assembly Cache (GAC).

  What Does Global Assembly Cache Mean?

  The Global Assembly Cache (GAC) is a folder in Windows directory to store the .NET assemblies that are specifically designated to be shared by all applications executed on a system.

  The concept of GAC is the result of the .NET architecture whose design addresses the issue of "DLL hell" that existed in COM (Component Object Model). Unlike in COM, there is no need for the assembly in GAC to be registered before its use. Each assembly is accessed globally without any conflict by identifying its name, version, architecture, culture and public key.

  The GAC is a machine-wide code cache used for side-by-side execution of assemblies. The GAC implements the feature of shared library where different applications reuse the code placed in the files located in a common folder.

In console application, compiled code is stored in assemblies with .exe extension. But in class libraries, which is stored with .dll extension.

What is an EXE and a DLL?

EXE and DLLs are assembly executable modules.

EXE is an executable file that runs the application for which it is designed. An EXE is produced when we build an application. Therefore, the assemblies are loaded directly when we run an EXE. However, an EXE cannot be shared with the other applications.

Dynamic Link Library (DLL) is a library that consists of code that needs to be hidden. The code is encapsulated inside this library. An application can consist of many DLLs which can be shared with the other programs and applications.

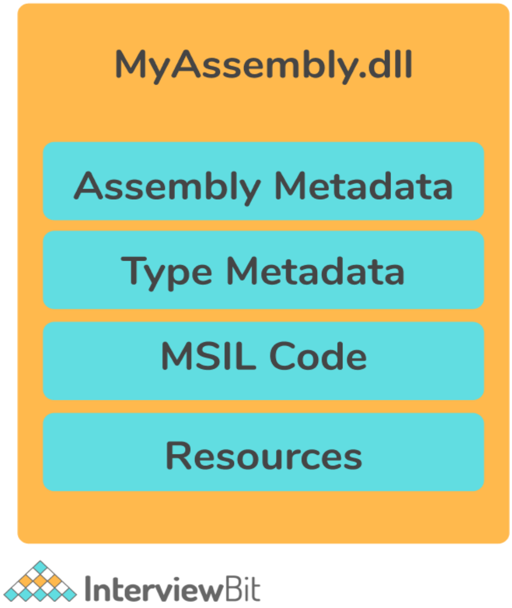
What is the use of manifest in the .NET framework?

Manifest stores the metadata of the assembly. It contains metadata which is required for many things as given below:

- Assembly version information.
- Scope checking of the assembly.
- Reference validation to classes.
- Security identification

Explain the different parts of an Assembly.

The different parts of an assembly are:



The different parts of an assembly are:

- **Manifest** – Every static or dynamic assembly holds a data collection that gives details about how the elements in the assembly relate to each other. An assembly manifest consists of complete metadata required to specify version requirements and security identity of an assembly, and also the metadata required for defining the assembly scope and resolving references to classes and resources. The assembly manifest will be stored in either a standalone PE(Portable Executable) file that holds only assembly manifest information, or in a PE file (a .exe or .dll) with MSIL(Microsoft intermediate language) code.
- **Type Metadata** – Metadata gives you additional information such as types, type names, method names, etc about the contents of an assembly. Metadata will be automatically generated by the Compilers from the source files and the compiler will embed this metadata within target output files like .exe, .dll, or a .netmodule (in the case of multi-module assembly).
- **MSIL** – Microsoft Intermediate Language (MSIL) is a code that implements the types. It includes instructions to load, store, initialize, and call the methods on objects. Along with this, it also includes instructions for control flow, direct memory access, arithmetic and logical operations, exception handling, etc. This is generated by the compiler using one or more source code files. During the runtime, the JIT (Just In Time) compiler of CLR (Common Language Runtime) converts the MSIL code into native code to the Operating System.

  MSIL is the Microsoft Intermediate Language, which provides instructions for calling methods, memory handling, storing and initializing values, exception handling, and so on.

  The instructions provided by MSIL are platform-independent and are generated by the language-specific compiler from the source code. JIT compiler compiles the MSIL into machine code based on the requirement.

- **Resources** – Resources can be a list of related files such as .bmp or .jpg files. These resources are static, which means they do not change during run time. Resources are not executable items.