# **Assemblies in .NET**

Assemblies are the basic building blocks required for any application to function in the .NET realm. They are partially compiled code libraries that form the fundamental unit of deployment, versioning, activation scoping, reuse, and security. Typically, assemblies provide a collection of types and resources that work together to form a logical unit of functionality. They are the smallest deployable units of code in .NET. Compared to the executable files assemblies are far more reliable, more secure, and easy to manage. An assembly contains a lot more than the Microsoft Intermediate Language (MSIL) code that is compiled and run by the Common Language Runtime (CLR). In other words, you can say that an assembly is a set of one or more modules and classes compiled in MSIL, and metadata that describes the assembly itself, as well as the functionalities of the assembly classes.

There are two types of assemblies:

* **Private Assembly** - Refers to the assembly that is used by a single application. Private assemblies are kept in a local folder in which the client application has been installed.
* **Public or Shared Assembly** - Refers to the assembly that is allowed to be shared by multiple applications. A shared assembly must reside in Global Assembly Cache (GAC) with a strong name assigned to it.

**Assemblies have the following properties:**

* Assemblies are implemented as .exe or .dll files.
* They are loaded into memory only when they need
* Can share assemblies

**Assembly Manifest**

Assemblies maintain all their information in a special unit called the manifest. Every assembly has a manifest.  
  
The followings are the contents of an Assembly Manifest:

* **Assembly name** - Represents a text string that specifies the assembly's name.
* **Version number** - Represents a major and minor version number, as well as a revision and build number. The CL.R makes use of these numbers to enforce version policy.
* **Culture** - Represents information of the culture or language, which the assembly supports. An assembly is a container of only resources containing culture- or language-specific information.
* **Strong name information** - Represents the public key from the publisher, if a strong name is assigned to an assembly.
* **List of all files in the assembly** - Represents a hash of each file contained in the assembly and a file name.
* **Type reference information** - Represents the information used at the runtime to map a type reference to the file that contains its declaration and implementation.
* **Information on referenced assemblies** - Represents a list of other assemblies that are statically referenced by the assembly. Each reference includes the names of dependent assemblies, assembly metadata (version, culture, operating system, and so on), and public key, if the assembly is strong named.

**Global Assembly Cache (GAC)**

GAC is a central repository (cache) in a system in which assemblies are registered to share among various applications that execute on local or remote machines. .NET Framework provides the GAC tool (gacutil.exe utility), which is used to view and change the content of GAC of a system. Adding new assemblies to GAC and removing assemblies from GAC are some of the tasks that can be performed by using the gacutil.exe utility. GAC can contain multiple versions of the same .NET assembly. CLR checks GAC for a requested assembly before using information of configuration files.  
  
The gacutil.exe /i <assembly name> - is the command that is used to install an assembly in GAC. Users use the Command Prompt of Visual Studio to install an assembly in GAC by using this command.

**Difference between .EXE and .DLL files**

**EXE**

1. It is an **executable file**, which can be run independently.
2. EXE is an out-process component, which means that it runs in a separate process.
3. It cannot be reused in an application.
4. It has a main function.

**DLL**

1. It is **Dynamic Link Library** that is used as a part of EXE or other DLLs. It cannot be run independently.
2. It runs in the application process memory, so it is called as in-process component.
3. It can be reused in an application.
4. It does not have a main function.

## **Assemblies in the common language runtime**

Assemblies provide the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type doesn't exist outside the context of an assembly.

An assembly defines the following information:

* ****Code**** that the common language runtime executes. Each assembly can have only one entry point: DllMain, WinMain, or Main.
* The ****security boundary****. An assembly is the unit at which permissions are requested and granted. For more information about security boundaries in assemblies, see [Assembly security considerations](https://learn.microsoft.com/en-us/dotnet/standard/assembly/security-considerations).
* The ****type boundary****. Every type's identity includes the name of the assembly in which it resides. A type called MyType that's loaded in the scope of one assembly isn't the same as a type called MyType that's loaded in the scope of another assembly.
* The ****reference-scope boundary****: The [assembly manifest](https://learn.microsoft.com/en-us/dotnet/standard/assembly/" \l "assembly-manifest) has metadata that's used for resolving types and satisfying resource requests. The manifest specifies the types and resources to expose outside the assembly and enumerates other assemblies on which it depends. Microsoft intermediate language (MSIL) code in a portable executable (PE) file won't be executed unless it has an associated [assembly manifest](https://learn.microsoft.com/en-us/dotnet/standard/assembly/" \l "assembly-manifest).
* The ****version boundary****. The assembly is the smallest versionable unit in the common language runtime. All types and resources in the same assembly are versioned as a unit. The [assembly manifest](https://learn.microsoft.com/en-us/dotnet/standard/assembly/" \l "assembly-manifest) describes the version dependencies you specify for any dependent assemblies. For more information about versioning, see [Assembly versioning](https://learn.microsoft.com/en-us/dotnet/standard/assembly/versioning).
* The ****deployment unit****: When an application starts, only the assemblies that the application initially calls must be present. Other assemblies, such as assemblies containing localization resources or utility classes, can be retrieved on demand. This process allows apps to be simple and thin when first downloaded. For more information about deploying assemblies, see [Deploy applications](https://learn.microsoft.com/en-us/dotnet/framework/deployment/).
* A ****side-by-side execution unit****: For more information about running multiple versions of an assembly, see [Assemblies and side-by-side execution](https://learn.microsoft.com/en-us/dotnet/standard/assembly/side-by-side-execution).