**Assembly in .Net**

**What is an assembly?**

Assemblies are the fundamental units of deployment, version control, reuse, activation scoping, 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.

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

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.

.NET supports three kinds of assemblies:

1. private
2. shared
3. satellite

**1.Private Assembly**

Private assembly requires us to copy separately in all application folders where we want to use that assembly’s functionalities; without copying, we cannot access the private assembly features and power. Private assembly means every time we have one, we exclusively copy into the BIN folder of each application folder.

**2.Public Assembly**

Public assembly is not required to copy separately into all application folders. Public assembly is also called Shared Assembly. Only one copy is required in system level, there is no need to copy the assembly into the application folder.

Public assembly should install in GAC.

Shared assemblies (also called strong named assemblies) are copied to a single location (usually the Global assembly cache). For all calling assemblies within the same application, the same copy of the shared assembly is used from its original location. Hence, shared assemblies are not copied in the private folders of each calling assembly. Each shared assembly has a four-part name including its face name, version, public key token, and culture information. The public key token and version information makes it almost impossible for two different assemblies with the same name or for two similar assemblies with a different version to mix with each other.

**GAC (Global Assembly Cache)**

When the assembly is required for more than one project or application, we need to make the assembly with a strong name and keep it in GAC or in the Assembly folder by installing the assembly with the GACUtil command.

**3.Satellite Assembly**

Satellite assemblies are used for deploying language and culture-specific resources for an application.

The different parts of an assembly are:

Diagram

Description automatically generated

* **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.
* **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.