**ASSEMBLIES IN .NET**

An Assembly is a basic building block of .Net Framework applications. It is basically a compiled code that can be executed by the CLR. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An Assembly can be a DLL or EXE depending upon the project that we choose.

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.

**Different components of an assembly.**

An assembly is a logical unit that is made up of the following four different types of components:

* **Assembly manifest.**

Assemblies has a special logical unit called Manifest. It contains:

* Assembly name
* Version number
* Culture
* Strong Name information
* Type Reference Information
* List of all files in Assembly
* Information on referenced assemblies
* An Assembly manifest consists of the information of an assembly
* **MSIL source code**

Represents the .Net program as per Microsoft standards.

* **GAC**
* GAC is a central repository in a system in which assemblies are registered to shared between application.
* GACUtil.exe is used to view and change the content of GAC in system
* GAC can contain multiple versions on .net assemblies

Thegautil.exe/I<assembly\_name> is used to install assembly in GAC

* **Type metadata**
* Assembly metadata describes every data type and member defined in the code.
* it stores name, version, culture, public key of an Assembly
* Also it stores the description of types like name, visibility, base class, interfaces, implemented etc.
* Metadata of an assembly is sharable among applications that execute on different platforms
* **Resources**

It consists of images that are part of the assembly, but not included in the CIL/MSIL code.

**Types of Assemblies**

Assemblies are basically the following two typ*es:*

1. *Private Assembly*
2. *Shared Assembly*

**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.

For example, imagine that you have created a DLL containing information about your business logic. This DLL can be used by your client application. In order to run the client application, the DLL must be included in the same folder in which the client application has been installed. This makes the assembly private to your application. Now suppose that the DLL needs to be reused in different applications. Therefore, instead of copying the DLL in every client application folder, it can be placed in the global assembly cache using the GAC tool. These assemblies are called shared assemblies.

### What is Satellite Assembly?

Satellite assemblies are assemblies that is used to deploy culture and language for an application. A separate product id is assigned to each language and a satellite assembly is installed in language specific sub directory.

### How do you add/remove assembly from GAC?

you can add assembly by using below syntax:

****gacutil /i [assemblyName | assemblyPath]****

you can remove assembly by using below syntax:

****gacutil /u [assemblyName | assemblyPath]****

## **Add a reference to an assembly**

To use an assembly in an application, you must add a reference to it. When an assembly is referenced, all the accessible types, properties, methods, and other members of its namespaces are available to your application as if their code were part of your source file.

**Assembly Settings**Now as we know, when we build our application its assembly is generated either as .exe or .dll. We can always choose what shall be the extension of our assembly, as per our requirement. Following are the steps to achieve the same:

* Open the project Properties in the Solution Explorer of your Visual Studio and go-to Application tab
  + Assembly Name  
    name of the assembly that will be created (default value - project name)
  + Output Type  
    decides the extension of the output file (.exe or .dll)
  + value as Class Library   
    generates an assembly with .dll extension. This assembly is a non-executable, but can be referenced by other applications
  + value as Windows/Console Application  
    generates an assembly with .exe extension. This assembly can be executed directly, but can not be referenced by other application
  + Assembly Information  
    helps you to add more information, like version, description, company, about the assembly. Once you save the information, an AssemblyInfo.cs is added into your project file.

**A comparison between .exe and .dll**

The basic difference between the two types is that the .exe is an executable form of an assembly whereas an assembly takes the form of a .dll when its code is to be reused or referenced by some other applications. A .dll can not be executed as a standalone application. This derives the fact that, while a .exe executes in its own address space, a .dll always requires a host to load and execute it.  
  
Also, the address space is shared between the two. Secondly, the entry point for an executable is the main thread (commonly known as the main method), whereas a library(.dll) does not have a main method and is executed in the context of the running process.  
  
**Conclusion**We shall always create a .dll assembly when we want our code to be reused by other applications as a library reference. And we shall create an executable when we want to run our application as a standalone application.