Assemblies

# Definition

A windows executable (win32 exe) or a windows dynamic link library (win32 dll) runs on top of the windows operating system. We have similar exes / dll / library files for the other operating systems ... linux, macos to name a few.

The assembly is the unit of deployment in a .net framework. This means that any output from a compiler like c# / vb.net / f# will produce an assembly and this assembly in turn will run on top of the virtual operating system - .net framework. This also means that anything and everything in .net breakdowns into an assembly.

An Assembly in its physical form is a file in the OS File System. But unlike the binary OS file, which is meant to be executed on top of an OS directly, the assembly will execute on top of the .NET CLR.

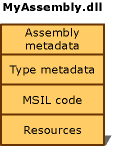
This means that any OS that has a .NET CLR on top of it will be able to JIT compile the assembly (which is an MSIL) and execute it on top of the OS and give the desired results.

# Assembly structure

In general, a static assembly can consist of four elements:

* The [assembly manifest](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-1.1/1w45z383%28v%3dvs.71%29), which contains assembly metadata.
* Type metadata.
* Microsoft intermediate language (MSIL) code that implements the types.
* A set of resources.

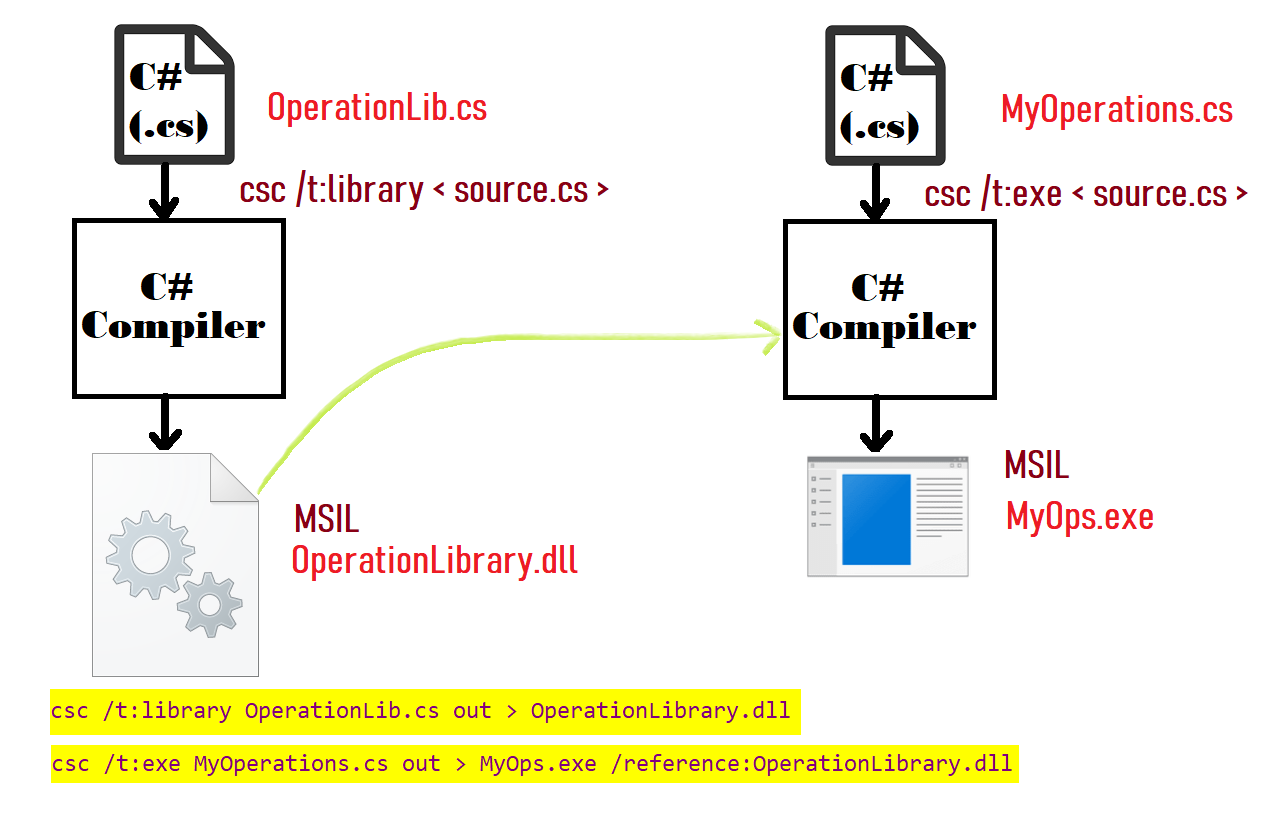
Only the assembly manifest is required, but either types or resources are needed to give the assembly any meaningful functionality.

****

**Fig.1. An assembly structure.**

# Libraries and Executables

A c# source code (.cs) can be compiled into any one of the two forms – Library and Executable. Fig.1 below shows the possible forms of an assembly.



**Fig.2.** **Libraries and Executables.**

## Libraries

If the requirement is of a re-usable .NET component which has a common / generic logic that can be used across multiple scenarios - compiling the c# source code (.cs) into a .NET Library is the perfect choice. Libraries are dependent on an EXE for execution and cannot be executed independently. The files produced are DLL files – Dynamic Linking Library (\*.dll).

## Executables

If the requirement is for an independent component which encapsulates a particular logic - compiling the c# source code (.cs) into a .NET EXE is the perfect choice. Unlike Libraries, Executables are independent and can also include a reference library optionally. The files produced are EXE files – Executables (\*.exe).

IMPORTANT: Library (DLL) and Executables (EXE) are both Assemblies because they are meant to be run on the .NET CLR. Net modules are an outdated compilation option which was not an assembly and is being no more actively used in new projects.

# C# source files compiling options

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl. | TYPE | COMMAND | COMMENTS | RESULTING FILE |
| 1 | .NET EXECUTABLE (EXE) |  | Source file: operations.cs  Desired Exe file:  operationswork.exe  The source file should have at least one class that has a Main() inside it.  And there should be only one (not more) Main() as that serves as the entry point. |  |
| 2 | .NET LIBRARY (EXE) |  | Source file: operations.cs  Desired Exe file:  operationslib.dll  It’s not mandatory to have a Main() entry point.  This is NOT a standalone file and has to be launched with help of an EXE. |  |
| 3 | NET MODULE  (netmodule) |  | Source file: operations.cs  Desired Exe file:  operationslib.dll  It’s not mandatory to have a Main() entry point.  This is NOT a standalone file and has to be launched with help of an EXE.  This file has NO MANIFEST and is not an assembly. |  |

Table.1. Compiling Options of CS Source files to Assemblies.

# Examples

**Example #1:**

1. Create a .net library by the name OperationsLib.
2. In that .net library, let there be a class named Operations. The Operations class should be in the namespace *Ganesh.Operations.Utils.*
3. Create two methods within the Operations class – Square and Cube of which the signatures are given below:
   1. *public double Square(double num)*
   2. *public double Cube(double num)*
4. Create a .net exe named OperationsExec. In the OperationsExec, use the methods of the OperationsLib - Square () and Cube () - and show the results.

Download OperationsExec

Download OperationsLib