Assemblies in .NET

# Definition

<https://docs.microsoft.com/en-us/dotnet/standard/assembly/>

Assemblies form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions for a .NET-based application. Assemblies take the form of an executable (.exe) file or dynamic link library (.dll) file, and are the building blocks of the .NET applications. They provide the common language runtime with the information it needs to be aware of type implementations. You can think of an assembly as a collection of types and resources that form a logical unit of functionality and are built to work together.

In .NET Core and .NET Framework, an assembly can be built from one or more source code files. In .NET Framework, assemblies can contain one or more modules. This allows larger projects to be planned in such a way that several individual developers work on separate source code files or modules, which are combined to create a single assembly.

# Properties

Assemblies are implemented as .exe or .dll files

Assemblies are only loaded into memory if they are required. If they are not used, they are not loaded. This means that assemblies can be an efficient way to manage resources in larger projects.

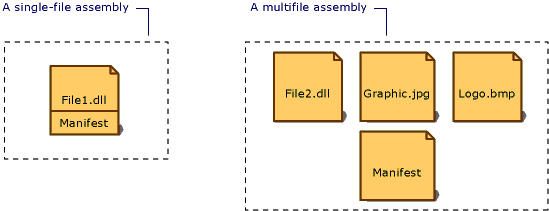
# Manifest

Within every assembly is an *assembly manifest*. Similar to a table of contents, the assembly manifest contains the following:

* The assembly's identity (its name and version)
* A file table describing all the other files that make up the assembly, such as other assemblies you created that your .exe or .dll file relies on, or even bitmap or Readme files.
* An *assembly reference list*, which is a list of all external dependencies — .dlls or other files your application needs that may have been created by someone else. Assembly references contain references to both global and private objects. Global objects are available to all other applications. In .NET Core, they are coupled with a particular .NET Core runtime. In .NET Framework, they reside in the global assembly cache. The [System.IO](https://docs.microsoft.com/en-us/dotnet/api/system.io) namespace is an example of an assembly in the global assembly cache. Private objects must be in a directory at either the same level as or below the directory in which your application is installed.
* Because assemblies contain information about content, versioning, and dependencies, the applications that use them need not rely on Windows registry values to function properly. Assemblies reduce .dll conflicts and make your applications more reliable and easier to deploy. In many cases, you can install a .NET-based application simply by copying its files to the target computer.

Every assembly, whether static or dynamic, contains a collection of data that describes how the elements in the assembly relate to each other. The assembly manifest contains this assembly metadata. An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. The assembly manifest can be stored in either a PE file (an .exe or .dll) with Microsoft intermediate language (MSIL) code or in a standalone PE file that contains only assembly manifest information.

The following illustration shows the different ways the manifest can be stored.



For an assembly with one associated file, the manifest is incorporated into the PE file to form a single-file assembly. You can create a multifile assembly with a standalone manifest file or with the manifest incorporated into one of the PE files in the assembly.

Each assembly's manifest performs the following functions:

* Enumerates the files that make up the assembly.
* Governs how references to the assembly's types and resources map to the files that contain their declarations and implementations.
* Enumerates other assemblies on which the assembly depends.
* Provides a level of indirection between consumers of the assembly and the assembly's implementation details.
* Renders the assembly self-describing.
* mbly's identity.

| **Information** | **Description** |
| --- | --- |
| Assembly name | A text string specifying the assembly's name. |
| Version number | A major and minor version number, and a revision and build number. The common language runtime uses these numbers to enforce version policy. |
| Culture | Information on the culture or language the assembly supports. This information should be used only to designate an assembly as a satellite assembly containing culture- or language-specific information. (An assembly with culture information is automatically assumed to be a satellite assembly.) |
| Strong name information | The public key from the publisher if the assembly has been given a strong name. |
| List of all files in the assembly | A hash of each file contained in the assembly and a file name. Note that all files that make up the assembly must be in the same directory as the file containing the assembly manifest. |
| Type reference information | Information used by the runtime to map a type reference to the file that contains its declaration and implementation. This is used for types that are exported from the assembly. |
| Information on referenced assemblies | A list of other assemblies that are statically referenced by the assembly. Each reference includes the dependent assembly's name, assembly metadata (version, culture, operating system, and so on), and public key, if the assembly is strong named. |

* You can add or change some information in the assembly manifest by using assembly attributes in your code. You can change version information and informational attributes, including Trademark, Copyright, Product, Company, and Informational Version. For a complete list of assembly attributes, see [Setting Assembly Attributes](https://docs.microsoft.com/en-us/dotnet/framework/app-domains/set-assembly-attributes).