How to: Build a multifile assembly

Article • 09/15/2021 • 4 minutes to read

This article explains how to create a multifile assembly and provides code that illustrates each step in the procedure.

> **Note**
>
> The Visual Studio IDE for C# and Visual Basic can only be used to create single-file assemblies. If you want to create multifile assemblies, you must use the command-line compilers or Visual Studio with Visual C++. Multifile assemblies are supported by .NET Framework only.

Create a multifile assembly

1. Compile all files that contain namespaces referenced by other modules in the assembly into code modules. The default extension for code modules is *.netmodule*.

   For example, let's say the `Stringer` file has a namespace called `myStringer`, which includes a class called `Stringer`. The `Stringer` class contains a method called `StringerMethod` that writes a single line to the console.

   C#

   ```
   // Assembly building example in the .NET Framework.
   using System;

   namespace myStringer
   {
       public class Stringer
       {
           public void StringerMethod()
           {
               System.Console.WriteLine("This is a line from
   StringerMethod.");
           }
       }
   }
   ```

2. Use the following command to compile this code:

C#

```
csc /t:module Stringer.cs
```

Specifying the *module* parameter with the **/t:** compiler option indicates that the file should be compiled as a module rather than as an assembly. The compiler produces a module called *Stringer.netmodule*, which can be added to an assembly.

3. Compile all other modules, using the necessary compiler options to indicate the other modules that are referenced in the code. This step uses the **/addmodule** compiler option.

   In the following example, a code module called *Client* has an entry point `Main` method that references a method in the *Stringer.dll* module created in step 1.

   C#

   ```
   using System;
   using myStringer;

   class MainClientApp
   {
       // Static method Main is the entry point method.
       public static void Main()
       {
           Stringer myStringInstance = new Stringer();
           Console.WriteLine("Client code executes");
           myStringInstance.StringerMethod();
       }
   }
   ```

4. Use the following command to compile this code:

   C#

   ```
   csc /addmodule:Stringer.netmodule /t:module Client.cs
   ```

   Specify the **/t:module** option because this module will be added to an assembly in a future step. Specify the **/addmodule** option because the code in *Client* references a namespace created by the code in *Stringer.netmodule*. The compiler produces a module called *Client.netmodule* that contains a reference to another module, *Stringer.netmodule*.

> **Note**
>
> The C# and Visual Basic compilers support directly creating multifile assemblies using the following two different syntaxes.
>
> Two compilations create a two-file assembly:
>
> ```
> C#
> csc /t:module Stringer.cs
> csc Client.cs /addmodule:Stringer.netmodule
> ```
>
> One compilation creates a two-file assembly:
>
> ```
> C#
> csc /out:Client.exe Client.cs /out:Stringer.netmodule Stringer.cs
> ```

5. Use the Assembly Linker (Al.exe) to create the output file that contains the assembly manifest. This file contains reference information for all modules or resources that are part of the assembly.

   At the command prompt, type the following command:

   al <*module name*> <*module name*> ... **/main:**<*method name*> **/out:**<*file name*> **/target:**<*assembly file type*>

   In this command, the *module name* arguments specify the name of each module to include in the assembly. The **/main:** option specifies the method name that is the assembly's entry point. The **/out:** option specifies the name of the output file, which contains assembly metadata. The **/target:** option specifies that the assembly is a console application executable (*.exe*) file, a Windows executable (*.win*) file, or a library (*.lib*) file.

   In the following example, *Al.exe* creates an assembly that is a console application executable called *myAssembly.exe*. The application consists of two modules called *Client.netmodule* and *Stringer.netmodule*, and the executable file called *myAssembly.exe*, which contains only assembly metadata. The entry point of the assembly is the `Main` method in the class `MainClientApp`, which is located in *Client.dll*.

Windows Command Prompt

```
al Client.netmodule Stringer.netmodule /main:MainClientApp.Main
/out:myAssembly.exe /target:exe
```

You can use the MSIL Disassembler (Ildasm.exe) to examine the contents of an assembly, or determine whether a file is an assembly or a module.

See also

- Create assemblies
- How to: View assembly contents
- How the runtime locates assemblies
- Multifile assemblies