Embedding .NET Assemblies inside .NET Assemblies

Posted on August 25, 2018 by Denham Coder

There are times when you create an assembly executable that has dependencies on other assemblies. You may want pass this assembly around without having the need for it to be installed however; if you have to copy DLL's around with the EXE, there could be a chance that they are forgotten or the wrong versions are copied.

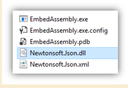
The topic of this post is to explain how we can embed DLL assemblies within our EXE assembly and load them at execution time.

Referencing an Assembly

Take a look at this little console app. It does nothing useful other than serialize an object to JSON. It does however; have a dependency on **Newtonsoft.Json.dll**, the most popular JSON library.

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine();

        var aa = JsonConvert.SerializeObject(GetPeople());

        Console.WriteLine($"{aa}\n\n");

        //Output: [{"Name":"Denham","Age":21},{"Name":"Wifey","Age":102}]
    }

    static List<Person> GetPeople()
    {
        return new List<Person>()
        {
            new Person() {Name = "Denham", Age = 21 },
            new Person() {Name = "Wifey", Age = 102 }
        };
    }
}

class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

If we take a look in the output folder we can see the referenced assembly copied at build time.



If we delete that assembly and execute the **EmbedAssembly.exe** app we get the "**Could not load file or assembly**" error.

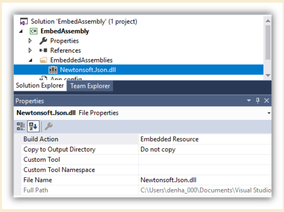
```
Unhandled Exception: System.IO.FileNotFoundException: Could not load file or assembly 'Newtonsoft.Json, ve
rsion=11.0.0.0, culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed' or one of its dependencies. The system c
annot find the file specified.
   at EmbedAssembly.Program.Main(String[] args)
```

So when passing this assembly around we must ensure **Newtonsoft.Json.dll** is passed as well and in some scenarios, a specific version of it.

Embedding an Assembly within an Assembly

Let's look at how we can embed **Newtonsoft.Json.dll** inside **EmbedAssembly.exe**.

- Within my solution I created an **EmbeddedAssemblies** folder and copied the **Newtonsoft.Json.dll** from the output folder into it.
- I right clicked the folder and selected **Add -> Existing Item** and added the DLL.
- I then highlighted **Newtonsoft.Json.dll** and in the **Properties** pane I changed **Build Action** to **Embedded Resource** and **Copy to Output Directory** to **Do not copy**.
- You can then remove the **NewtonSoft** nuget package and add a reference to the DLL in the **EmbeddedAssemblies** folder.



Now we have our assembly embedded, we need to know how to reference it. There is a logical path to a resource however; a simple way is by adding the following method to output all the embedded resources to the **Output** log. Run the app in debug mode to view the output.

```
static void ListEmbeddedResourceNames()
{
    Trace.WriteLine("Listing Embedded Resource Names");

    foreach (var resource in Assembly.GetExecutingAssembly().GetManifestResourceNames())
        Trace.WriteLine("Resource: " + resource);
}
```

```
Listing Embedded Resource Names
Resource: EmbedAssembly.EmbeddedAssemblies.Newtonsoft.Json.dll
```

The Resource listed is the full name to our resource which corresponds to the **namespace.folder.assembly**. Make a note of this name. If you do not see the resource listed, check the steps above were completed.

The last step is to add the following code highlighted below.

- In our Main method I set up an **AssemblyResolve** *event* listener.
- I had to move the code that does **JsonConvert.SerializeObject** into another method. The reason being is that the .NET loader will try and load the **Newtonsoft.Json.dll** before the event handler is wired up.
- When the .NET loader tries to load **Newtonsoft.Json.dll**, the event will fire and execute the **CurrentDomain_AssemblyResolve** method. This method takes care of pulling out the assembly and loading it.

```
static void Main(string[] args)
{
    ListEmbeddedResourceNames();
    AppDomain.CurrentDomain.AssemblyResolve += CurrentDomain_AssemblyResolve;
    Run();
}

static void Run()
{
    Console.WriteLine();

    var aa = JsonConvert.SerializeObject(GetPeople());

    Console.WriteLine($"{aa}\n\n");

    //Output: [{"Name":"Denham","Age":21},{"Name":"Wifey","Age":102}]
}

static Assembly CurrentDomain_AssemblyResolve(object sender, ResolveEventArgs args)
{
    using (var stream = Assembly.GetExecutingAssembly().GetManifestResourceStream("EmbedAssembly.EmbeddedAssemblies.Newtonsoft.Json.dll"))
    {
        var assemblyData = new Byte[stream.Length];
        stream.Read(assemblyData, 0, assemblyData.Length);
        return Assembly.Load(assemblyData);
    }
}
```

That's it. Let's compile and test it. As you can see the folder does not contain **Newtonsoft.Json.dll** however; the exe runs without the failure we first had above.

```
[Development] : dir
 Volume in drive C has no label.
 Volume Serial Number is 948D-EE1B

 Directory of C:\Users\denha_000\Documents\Visual Studio Projects\CS

24/08/2018  17:25    <DIR>          .
24/08/2018  17:25    <DIR>          ..
24/08/2018  17:35           669,184 EmbedAssembly.exe
24/08/2018  15:44               189 EmbedAssembly.exe.config
24/08/2018  17:35            19,968 EmbedAssembly.pdb
               3 File(s)        689,341 bytes
               2 Dir(s)  44,567,048,000 bytes free

C:\Users\denha_000\Documents\Visual Studio Projects\CSLangFeatures\E

[Development] : EmbedAssembly.exe

[{"Name":"Denham","Age":21},{"Name":"Wifey","Age":102}]
```

A nice simple way of carrying around your dependencies.

Share this:

Twitter Facebook

Related

Dynamically Load Different Versions of an Assembly
March 6, 2018
In "C#"

Visual Studio – Synchronize a Version Number across Multiple Assemblies
September 11, 2018
In "Visual Studio"

x86-x64 Registers
July 24, 2018
In "Cheatsheet"

Posted in Uncategorized Tagged C#

← C# Exceptions – Hidden Gems

Five year old piece of code suddenly fails. →

Archives

- October 2021 (2)
- June 2020 (1)
- May 2020 (1)
- November 2019 (2)
- October 2019 (1)
- September 2019 (3)
- December 2018 (1)
- November 2018 (2)
- September 2018 (1)
- August 2018 (5)
- July 2018 (18)
- June 2018 (6)
- May 2018 (5)
- April 2018 (14)
- March 2018 (14)
- February 2018 (6)
- January 2018 (9)

Tags

ACID Azure BASE C# CAP CD Cheatsheet CI Command Prompt Containers CPlusPlus Design DevOps Docker EF6 ES6 Excel Font Git GitHub GraphDB Hidden Gems Interop Javascript JSON Key Value Store Linux LocalDB Log4Net MSTest Neo4J NoSQL NUnit OData PGAgent PostgreSQL Postman Powershell Putty Python Redis Reflection Regex REST API SOLID SQL Server SQL Server 2017 SQL Server Data Tools SSDT SSH TDD TSQL Unit Testing Visual Studio VS2019 VSTS WebApi x64 x86 XUnit

Feed

- RSS - Posts
- RSS - Comments