Loading Mixed-Mode C++/CLI .dll (and dependencies) dynamically from unmanaged c++

Asked 9 years, 5 months ago Active 1 year, 4 months ago Viewed 5k times

5

3

I have a managed C++ assembly I'm loading dynamically in an unmanaged c++ application through a standard LoadLibrary() call. The managed C++ assembly has dependencies on several more managed (C#) assemblies. Everything worked fine until I moved all the managed assemblies to a subdirectory of the unmananged application. To illustrate:

- Managed C++ .dll (MyCoolDll.dll)
  - Dependent on DotNetDll1.dll
  - Dependent on DotNetDll2.dll
- Unmanaged C++ app (MyCoolApp.exe)
  - Loads MyCoolDll.dll via LoadLibrary("MyCoolDll.dll")

This worked fine, until I moved MyCoolDll.dll, DotNetDll1.dll & DotNetDll2.dll to /someSubDirectory (the code in MyCoolApp.exe was updated to LoadLibrary("someSubDirectory/MyCooldll.dll")

I'm guessing when MyCoolDll.dll is loaded, it's trying to find DotNetDll1.dll and DotNetDll2.dll in the working directory, instead of the directory it lives in.

How can I tell MyCoolDll.dll its dependencies live in a subdirectory? It's a library running inside of an unmanaged app, so I don't think I can specify this in an app.config or anything?

.net c++ c++-cli managed-c++ loadlibrary

Share Edit Follow Flag

asked Aug 10 '11 at 19:33
Jordan0Day
1,316 3 13 25

1 Wow, Hans, that worked! I was really dubious since MyCoolApp.exe is just a plain old Win32 application (not .NET), so I figured adding an app config file for it wouldn't help. Thanks! Do you want to write this up as an answer instead of a comment, and I'll mark it as accepted? – Jordan0Day Aug 11 '11 at 18:27

2 Answers

Active Oldest Votes

The CLR gets loaded in an unusual way in this scenario, through a thunk that the compiler injected when compiling the native export for __declspec(dllexport). Doing this is fine, it just

6

isn't particularly fast.

The CLR will go out hunting for a .config file to initialize the primary AppDomain. And will look for MyCoolApp.exe.config, regardless that this is not a managed executable at all. You can use the `<probing>` element to add subdirectories to search for assemblies.

Share Edit Follow Flag

edited Sep 3 '19 at 7:54
Wai Ha Lee
7,506 14 50 77

answered Aug 11 '11 at 18:38
Hans Passant
861k 130 1523 2336

8

I think what you're looking for is a custom assembly resolver. I had to use one to do what I think you are trying to do -- I wanted to locate some of the DLLs in a folder that wasn't in the tree of the initial unmanaged DLL (which loaded managed code eventually).

Step 1 is to make a function you can call to set up the resolver:

```
void PrepareManagedCode()
{
    // Set up our resolver for assembly loading
    AppDomain^ currentDomain = AppDomain::CurrentDomain;
    currentDomain->AssemblyResolve += gcnew
ResolveEventHandler(currentDomain_AssemblyResolve);
}  // PrepareManagedCode()
```

Then the resolver. This example has a global ourFinalPath which would in your case be the extra folder you were using:

```
/// <summary>
/// This handler is called only when the CLR tries to bind to the assembly and fails
/// </summary>
/// <param name="sender">Event originator</param>
/// <param name="args">Event data</param>
/// <returns>The loaded assembly</returns>
Assembly^ currentDomain_AssemblyResolve(Object^ sender, ResolveEventArgs^ args)
{
    sender;

    // If this is an mscorlib, do a bare load
    if (args->Name->Length >= 8 && args->Name->Substring(0, 8) == L"mscorlib")
    {
        return Assembly::Load(args->Name->Substring(0, args->Name->IndexOf(L",")) +
L".dll");
    }

    // Load the assembly from the specified path
    String^ finalPath = nullptr;
    try
    {
        finalPath = gcnew String(ourAssemblyPath) + args->Name->Substring(0, args-
>Name->IndexOf(",")) + ".dll";
        Assembly^ retval = Assembly::LoadFrom(finalPath);
        return retval;
    }
    catch (...)
    {
        }

        return nullptr;
    }
```

Share Edit Follow Flag

answered Aug 10 '11 at 21:01
Ed Bayiates
10.6k 4 41 61

This is the path I had figured I would have to go down -- the problem seemed to be that I was still required to have MyCoolDll.dll's dependencies in the main app directory, since .net wouldn't search the subfolder for them before even loading up MyCoolDll.dll (so the assembly resolver wouldn't get set up first). To my amazement, Hans suggestion above works (even though MyCoolApp.exe isn't a managed executable.) – Jordan0Day Aug 11 '11 at 18:30

Your unmanaged DLL will load before an assembly resolve is needed. You can then load the mixed mode DLL yourself, and call the assembly resolve setup. For complex scenarios you'd need it; however, if a simple config file works for your case, great! – Ed Bayiates Aug 11 '11 at 21:37

The problem I was seeing was my mixed mode dll had dependencies on a few other .NET assemblies -- so unless I placed those dependencies in the main directory instead of the subfolder, I wouldn't be able to get to any executable code in my mixed mode dll where I could wire up an assembly resolver. – Jordan0Day Aug 12 '11 at 13:47

Thanks, this works, but I have some remarks/questions: - Why do you do something special with mscorlib? It seems to work without doing anything - using AssemblyName to parse the args->Name would be better - Be careful to return the assembly actually requested, and not always the same one. I neglected to do that and bad things ensued :) – Melvyn Aug 15 '17 at 18:13

@Yaurthek This was an example from working code to show you can choose where to load from in this call. In our case we wanted to load those libraries from a different place. – Ed Bayiates Aug 16 '17 at 14:59

@EdBayiates Why do you use try catch for assembly resolver instead of if file exists? – Code Name Jack Dec 10 '18 at 11:46

@CodeNameJack there are many errors that can occur (e.g., can't open, wrong bit format, unloadable, etc.) and that one catch neatly handles all of them. The file not existing is just one possibility, and if you look at the way codebase searching works the load might still work even if the file you expected doesn't exist. All errors are handled the same way (returning nullptr), so there's not much benefit adding many other IF conditions. It'd just be code clutter. While exceptions aren't terribly performant, a DLL load is such a long operation that performance isn't a concern here. – Ed Bayiates Dec 10 '18 at 14:19

I agree to what you say, but In situations where there are 3-4 folders to look for files, it will keep on throwing exceptions, at least in the beginning. Thank you for providing an explanation by the way. – Code Name Jack Dec 10 '18 at 14:53