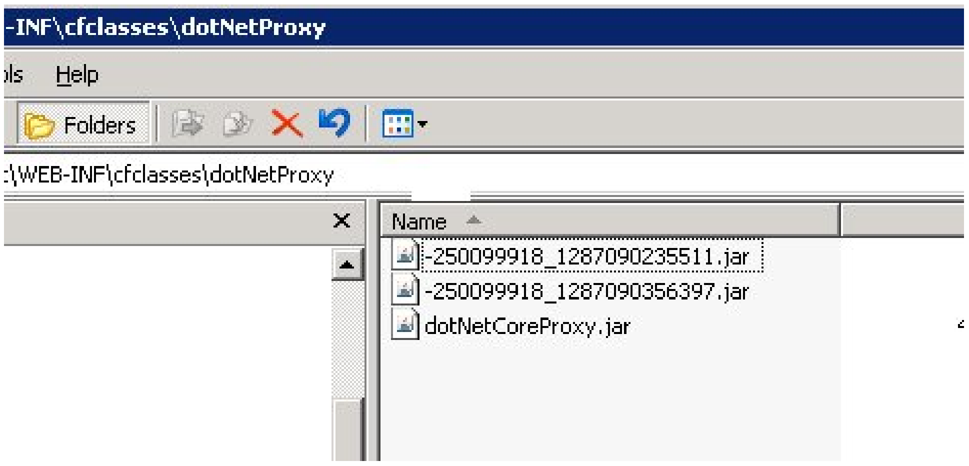
Accessing Both 64bit and 32bit Assemblies

Posted At : January 3, 2011 11:07 AM | Posted By : Mark Kruger
Related Categories: Coldfusion Troubleshooting

Many readers will doubtless recall the war waged by the Muse against .NET on 64bit ColdFusion 8. If not, you can read all about it in this series of posts on Muse Vs. .NET Integration. It was clear from the outset that using 64bit CF against a 32bit assembly was not working for us. Naturally we went down the path of recompiling everything into 64bit and we had multiple obstacles to overcome. But the fact that we could not communicate with 32bit assemblies always puzzled me. The communication (like most things on a web server) happens through a socket to a listener provided by the integration service (just like Verity and Sequelink). I could not reasonably explain to my own satisfaction why it should be that a 32 bit assembly was inaccessible. After all, our tests using the assembly directly worked fine. I assumed in a vague sort of way that differences in how variables were stored and passed back and forth must be to blame.

A few weeks ago I got a tip from the always insightful Rick Root on CF-Talk - who figured this out with the help of the folks at Just CF (SupportObjective). His problem was different than mine. He had a mixed environment with both 32bit assemblies and 64bit assemblies. In his experimentation he discovered something that solves both our problems. When you call and assembly (*any* assembly) using .NET integration ColdFusion creates some jar files under the hood. You will find them in the /cfclasses/dotnetproxy folder. Here's a sample folder where the server is using 2 .NET assemblies:



```
-250099918_1287090235511.jar
-250099918_1287090356397.jar
dotNetCoreProxy.jar
```

Of course, one of the Jar files that ColdFusion creates is the actual assembly interface which "mirrors" the methods and properties of the .NET interface. It's the one with the cryptic name that looks like "-23480983_3023903.jar". The other file however is special. It's the one that is always named *dotNetCoreProxy.jar*. It is compiled only once (unless you delete it) on *the first time you instantiate a .NET assembly*. It provides (presumably) the interface to the proxy listener.

The Fix

Now here's the kicker. If the first assembly you call is a 32bit assembly - in other words, if you call the 32bit framework *first* - then this little jar file is compiled to access the 32 bit framework and *cannot* access the 64bit framework. In fact, there's some good evidence that it can't reliably access 32bit DLLs either (it seems error prone). However, if you **access a 64bit assembly first** then the jar file is compiled in such a way as to be able to access both the 64bit and 32bit frameworks.

Conclusion

To avoid having to run this gauntlet I would suggest accessing a .NET Assembly on the 64 bit framework first and backing up the subsequent dotNetCoreProxy.jar file to avoid future confusion. I'm sure you can find a nice "Hello World" assembly to compile to 64bit. If you have legacy .NET integration code and you are moving from 32bit to 64bit then it will likely *not work* out of the box until you have done this (or it may work once or twice and then give you inscrutable errors). You will need to get the correct dotNetCoreProxy.jar file compiled (the 64bit version) before you can access your 32 bit assembly dlls successfully and reliably. The good news is that you don't *necessarily* need to recompile all your assemblies to use the 64bit CLR - although there may be a performance or compatibility reason to do so.