

Re: soapsuds

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.remoting/2004-06/0138.html>

From: Sahil Malik (contactmethrumblog_at_nospam.com)

Date: 06/09/04

Date: Wed, 9 Jun 2004 10:02:09 -0400

I don't know what wise men would say, but here is what I'd say.

> *Thanks for your comments. I have come across this view many times, however,*

> *I do not understand it. I have had no problems using soapsuds at all apart from the one that I have mentioned in this post which I have now resolved.*

> *This would appear to be related to namespacing. If the interface in the 'general' assembly is in the same namespace as the implementation object in the 'server' assembly, then the soapsuds extraction falls over. Placing the objects in different namespaces resolved the issue. Is there a restriction in .Net that namespaces cannot span assemblies? I am not aware of it. In any case, it is easy enough to code the 'proxy' remote class code including the Soap attributes by hand.*

The problem with Soapsuds is, you cannot have one small exe generate this information for every possible case out there. Other such examples in .NET framework, where you are better off implementing stuff yourself are the Serializable Attribute versus ISerializable, CommandBuilder versus specifying commands etc. etc. (fairly long list really). Anyway without wasting time on "Why not to use soapsuds", since there is plenty about that online, just don't use it .. hehe :).

> *I can now use new to instantiate my remote object, access it via an interface defined in a shared assembly (which can be implemented by other remote objects) which leaves me totally free to redevelop the server object.*

> *I have complete control over versioning via the framework. I believe that is what versioning was designed to do. However, nothing is stopping me from adding a V2 interface or allowing old clients to connect to new server assemblies.*

True, and that in my mind is a better approach. You should have a separate dll which sits both on the server and the client allowing objects to talk to each other. Yes you could have a separate V2 interface in the same dll, but the question is, why would you want older clients to talk to newer server assemblies? Ideally the whole product should work as one u