

# Re: WCF Contract Design Best Practices Question

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.webservices/2007-08/msg00108.htm>

---

- *From:* [ronscottlangham@xxxxxxxxxx](mailto:ronscottlangham@xxxxxxxxxx)
  - *Date:* Mon, 20 Aug 2007 19:02:11 -0700
- 

See below..

On Aug 19, 1:56 pm, "FlyFishGuy" <FlyFish...@xxxxxxxxxxxxxx> wrote:

Thanks for the feedback. After reflecting on what I've been finding, I had decided to take the approach you have. It seemed that it would be best to create separate contracts for communication only and converting my objects on each end. Given my application structure and the way I'm trying to do this, it's been a real struggle. Perhaps I cannot achieve this. I'm willing to re-shuffle if necessary, but I'd like to stay this course, unless I'm wasting my time on this path.

Here is my situation. First of all, everything I'm doing is internal and I don't have to worry about maintaining client apps. I have several apps (will be adding more). Some are servers, some clients, and some are both. I have a class library, which houses all of my business rules and internal class definitions. This library is core to everything. I share a lot of functionality among many apps with it, but this is where my issues arise.

I had this thought that I could host my WFC client proxies in this library so I could easily reference them from any client app. I've had to do a bit of restructuring to get around circular references and the like. I thought I had it worked out on a client app, but I went back to one of my server apps, my contract interface reference was ambiguous. I then placed the server app's service code in it's own namespace and now I get the 'zero application endpoints' error at runtime. Adding the new namespace to my contract attribute for the endpoint (in client app's app.config) has not resolved this.

Ron-> I would recommend that the client proxies be in their own DLL completely separate from the server code. My current project also has similar design. I have 3 root directories for code in the project, Server, Common, and Client. I put code in Common that I want to be used by both Server and Client. In the Client code

## Re: WCF Contract Design Best Practices Question

I have a separate DLL that holds the client proxies that were generated by the svcutil. You probably really need to keep the client proxies separate from the server mirror objects, if I understand correctly, it sounds like this is the issue that you are having.

I don't mind putting out fires if I my approach is sound and I know I can get there. Does this sound like it's a feasible approach or am I doomed to failure?

Also, even though I've embedded functions for the conversion of my objects to WCF objects in the base objects themselves (and simply call that method in each service app), it seems that since svcutil generates a lookalike of my WCF object (and not that actual object itself) I'm still going to have to recreate the code on each client that converts back to my true base object. Any thoughts?

Ron-> On the server side, you need conversion objects that convert from WCF objects to your internal object, and on the client side you need conversion objects that convert from client side objects to WCF (svcutil generated) objects. You can't really use the same conversion objects on both sides since the client side is using the svcutil generated ones. In my designs, I make my common business classes and code totally separate from the WCF code and do not put the conversion code in this layer. Instead, I put a conversion layer in the service layer that then calls the common internal classes. And, on the client side I have another conversion layer, generally wrappers, around the client side WCF objects.

Regarding the collections, I've gotten a habit (on internal applications) of implementing collections of my objects by simply creating a new class that inherits from a Generic List of that particular object. It makes a simple, strongly typed, collection and virtually no code. I can extend it, but I can't override the base methods, and all I have to do is add the CollectionDataContract attribute to serialize it for WCF. Is there any issue with that approach?

Actually haven't used the CollectionDataContract, will have to look at

Re: WCF Contract Design Best Practices Question

this.

I generally use Generics in my classes to provide strong type collections, e.g. List<myObj>. And, then in my service layer I generally either accept or return a list of this objects (e.g. myObj[] ). I then can easily convert to or from my Generics list, e.g.

```
return myObjList.ToArray();  
or new List<myObj>(myObjArray)
```

Hope this helps.

Thanks again.

<ronscottlang...@xxxxxxxx> wrote in message

[news:1187462041.582696.293290@xx](mailto:news:1187462041.582696.293290@xx)

On Aug 17, 1:05 pm, "FlyFishGuy" <FlyFish...@xxxxxxxxxxxxxxxx> wrote:

I'm fairly new to WCF. I have a project, which I already have a lot of time already invested it, which involves much data processing and cross application communication. WCF seemed like a great way to go, moving forward.

I have been diligently designing thread-safe classes to meet future needs. WCF doesn't seem to like classes without Set methods, so I find myself butchering work I have already done to be able to pass my objects back and forth over the wire.

Being new to all of this, I don't want to take the wrong path, with regard to my design principles. In a situation where I want to utilize

## Re: WCF Contract Design Best Practices Question

my existing code and plan well for future development, I'm in need of a sound approach. It just seems as my classes grow in complexity, I'm faced with the options of stripping them down to work with WCF or creating redundant copies of the classes, which are difficult to maintain.

Should I just forget about readonly properties and private variables and make everything public variables? Should I create a duplicate watered down version of every class I have to accomplish the same thing? Does it make sense to inherit WCF amicable classes from my existing classes and extend my readonly properties with write access? Should I be attempting to add the <DataMember> attribute to my private variables and resurrect public access on the client?

I just don't want to go down the wrong path and regret it later. I can see already that I'm going to have to make compromises, I just don't want to end up in a box.

Any thoughts, suggestions, or references to documentation would be greatly appreciated.

## Re: WCF Contract Design Best Practices Question

In my experience, I generally create separate classes with public properties that are used just for the web service. I tend to keep these separate and hopefully static without being affected by my underlying classes. I then have conversion routines that will convert one from the other. But, it sounds like that you have a lot of classes, and this hasn't always been my case. But, I prefer having the separate web service objects so I can fine tune them for their purpose without having to push this requirement down to my other business objects.

I tend to make them separate in hopefully also keeping the web service mostly static and not changing much since this may require that the client also be kept up to date. Not a big problem for internal products, but could be a bigger issue if distributed to external customers. You may want to update logic and classes on the server without it affecting the clients.

Also, some .NET objects cannot be serialized directly across the wire, such as Collections. So, the web objects may be an array of object, e.g. `foo[] GetFoods()`, while the internal business objects may be collection or other non-web service compliant object.

If you do have a lot of classes, possibly you could use a generic conversion method between web service objects and business objects using reflection without having to write a custom convert function for each one. Could be a little slower if lots of objects, so something to consider.

I have also used general objects for passing values that I expect to change. For example, may create a Property object that has a name and value, and then add array of the Property objects to another class. This allow me to pass general property values without having to create a specific new property for each item. Allows some extensibility without having to change the web service contract.

Ron