

## Re: Remoting Objects: style question

**Source:**

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.remoting/2004-07/0258.html>

---

**From:** Ken Kolda (*ken.kolda\_at\_elliemae-nospamplease.com*)

**Date:** 07/12/04

Date: Mon, 12 Jul 2004 15:53:31 -0700

Actually, I got point #4 below incorrect. It should be:

4) If the first class, D, to which you cast (using "is" or "as") the remoted object derives from a second class, C, the "is" and "as" will obey the inheritance rules (i.e. "is" will return true and "as" will return non-null). Note that this is the case even if the remoted object does not derive from either class C or D.

What it comes down to is that the recognized "type" of the remote object is determined by the first cast that is made of it AND by any subsequent casts to types that are derived from the currently recognized type.

Most likely, if you use abstract objects, the first cast will be done in the `Activator.GetObject()` call (the first parameter gives it the type info to use). Once that first cast is made, the object can be reliably downcast ("is" and "as" will work correctly for any base class of the one specified in the `GetObject()` call). But, the object cannot be reliably upcast. In fact, if it's upcast, it will take on the identity of the first derived class to which it is cast, regardless of whether it derives from that class. For example, give the following classes:

```
class A { }  
class AB : A { }  
class AC : A { }
```

Say you remote an AB instance and, on the client side have the following code:

```
A obj = (A) Activator.GetObject(typeof(A), ...);  
AC acobj = obj as AC;  
AB abobj = obj as AB;
```

In the code above, `acobj` will be non-null even though the remoted object is not of this type -- any attempts to use AC's method will result in an exception. Conversely, `abobj` will be null (because it cannot be both AB and AC). However, if you rearrange the code to be:

```
A obj = (A) Activator.GetObject(typeof(A), ...);
AB abobj = obj as AB;
AC acobj = obj as AC;
```

The abobj will be non-null and acobj will be null. Of course, all of this could be avoided if you used typeof(AB) in the original GetObject() call. But, presumably the reason this issue came up is because of the desire to use remoted object polymorphically, so knowing the exact type of the remoted object may not be feasible.

Ken

"Ken Kolda" <ken.kolda@elliemae-nospamplease.com> wrote in message news:um1HRcFaEHA.3708@TK2MSFTNGP10.phx.gbl...

> *I made a quick remoting project and, after some experimentation, I'd say that*  
> *using both "is" and "as" are unreliable when it comes to remoting objects*  
> *(whether via interfaces or abstract base classes). From what I could tell:*  
>  
> *1) "is" will always return true and "as" will always return non-null if the*  
> *right-hand side of the operator is an interface type (regardless of whether*  
> *the object actually implements the interface).*  
>  
> *2) "is" will always return true and "as" will always return non-null for the*  
> *FIRST class (abstract or concrete) against which it is tested. This is true*  
> *regardless of whether the object actually derives from the specified class.*  
>  
> *3) "is" will always return false and "as" will always return null for any*  
> *class beyond the first class against which it is tested.*  
>  
> *4) If class D derives from class C and "o is C == true" (where o is the*  
> *remote object), then "o is D" will return false (since it is not the first*  
> *class against which the "is" operator is used). The "as" operator works*  
> *similarly. Thus, inheritance is not even obeyed by the "is" and "as"*  
> *operators for remote objects.*  
>  
> *This almost seems like the "Remoting Uncertainty Pincipal" -- attempting*  
> *to determine the type of the object actually forces the object to take on*  
> *that type. And I noticed that neither "is" nor "as" actually caused the remote*  
> *object to be instantiated (I was using a SAO and didn't even have the*  
> *server running during the test).*  
>  
> *I never use inheritance with remoted object like this so I'd never*  
> *observed*

> *this behavior, but it's certainly not what I would have expected.*  
>  
> *Ken*  
>  
>  
>  
> *"Bob Rundle" <rundle@rundle.com> wrote in message*  
> *news:ORy8Gp3ZEHA.2216@TK2MSFTNGP10.phx.gbl...*  
> > *Sunny,*  
> >  
> > *I seem to have found a serious problem with the interface route.*  
> >  
> > *I have created a number of interfaces and compiled them into an*  
*interface*  
> > *library.*  
> >  
> > *I have discovered that the "is" operator cannot reliably distinguish*  
> *between*  
> > *the interfaces that are marshalled through remoting. For example*  
> >  
> > *if(item is IJobQueues)*  
> > {  
> > *IJobQueues jqs = item as IJobQueues;*  
> > *jqs.AddJobQueue("asdf");*  
> > }  
> >  
> > *will throw an exception in the AddJobQueue method if item does not*  
> *actually*  
> > *implement the IJobQueues interface.*  
> >  
> > *However if I implement everything with abstract classes, everything*  
*works*  
> > *fine. The client never gets confused about the type of the object it is*  
> > *holding.*  
> >  
> > *This appears to be a serious bug in .NET remoting. What am I missing?*  
> >  
> > *Regards,*  
> > *Bob Rundle*  
> >  
> >  
> > *"Sunny" <sunny@newsgroups.nospam> wrote in message*  
> > *news:uTSi8CcZEHA.2776@TK2MSFTNGP10.phx.gbl...*  
> > > *That's true. I need to use (MBR) casting, but it does not brake*  
> > > *anything.*  
> > >  
> > > *Sunny*  
> > >  
> > >  
> > > *In article <ezBYWjYZEHA.1152@TK2MSFTNGP09.phx.gbl>, nn@nnnnnn.nl*  
*says...*

>>>> *What I know is that one of the disadvantage of using interface is that*  
>> *you*  
>>>> *can't pass interface-objects as parameters in methods from*  
>> *remoting-objects*  
>>>> *because the interface cannot be casted back to the MarshalByRefObject.*  
>>>>  
>>>> *So it depends a little of what you want to do.*  
>>>>  
>>>> *kinds regards,*  
>>>> *Henrik*  
>>>>  
>>>>  
>>>> *"Sunny" <sunny@newsgroups.nospam> schreef in bericht*  
>>>> *news:uNpB3lQZEHA.2216@TK2MSFTNGP10.phx.gbl...*  
>>>>> *I preffer the interface approach, while Allen likes the abstract*  
>>>>> *classes. I can not go deep in any pros and cons on both*  
*approaches.*  
>>>>> *Maybe the bigger pros in the abstract class approach is that Allen*  
> *has*  
>>>>> *very good article about it :)*  
>>>>>  
>>>>> *Sunny*  
>>>>>  
>>>>> *P.S. here is the link to Allen's abstract class article:*  
>>>>> *[http://www.glacialcomponents.com/ArticleDetail.aspx?](http://www.glacialcomponents.com/ArticleDetail.aspx?articleID=RemoteObject)*  
>>>>> *articleID=RemoteObject*  
>>>>>  
>>>>> *In article <uSeUtCQZEHA.4092@TK2MSFTNGP11.phx.gbl>,*  
> *rundle@rundle.com*  
>>>>> *says...*  
>>>>>> *I have a style question wrt Remoting Objects.*  
>>>>>>  
>>>>>> *I'm trying to avoid putting all my remoting objects in a class*  
> *library*  
>>>>> *to be*  
>>>>>> *built into both client and server. I'm using C# exclusively.*  
>>>>>>  
>>>>>> *Two ideas:*  
>>>>>>  
>>>>>> *1. I have a class library of base class remoting objects.*  
The  
>> *server*  
>>>>> *uses class derived from these base classes. The client links in*  
> *the*  
>>>>>> *baseclasses and accesses derived class functionality through*  
*base*  
>> *class*  
>>>>>> *methods.*  
>>>>>>

>>>>> 2. I create interfaces for all the remoting objects and build  
a  
>> class  
>>>>> library from the interfaces which is then referenced by both  
> client  
>> and  
>>>>> server.  
>>>>>  
>>>>> Advise or comments?  
>>>>>  
>>>>> Regards,  
>>>>> Bob Rundle  
>>>>>  
>>>>>  
>>>>>  
>>>>  
>>>>  
>>>>  
>>  
>>  
>  
>