Overriding serialization for a particular .NET type

Asked 16 years, 2 months ago Modified 16 years, 2 months ago Viewed 2k times

3

Please consider this example class:

```
[Serializable]
public class SomeClass
{
    private DateTime _SomeDateTime;

    public DateTime SomeDateTime
    {
        get { return _SomeDateTime; }
        set { _SomeDateTime = value; }
    }
}
```

I would like to alter the serialization of any DateTime declared in the class according to my own rules. The members of this class will change frequently and I don't want to maintain a custom serializer for every change. Also I would like this behaviour to be inherited by subclasses and not code a custom serializer for every one. The serialization is being output by a web service. Thanks for any help!

.net serialization overriding

Share
Improve this question
Follow

edited Oct 6, 2008 at 14:07

asked Oct 6, 2008 at 11:06

Alex Angas
60k ● 41 ● 143 ● 212

3 Answers

Sorted by: Highest score (default)

3

look at the OnSerializing and OnDeserializing attributes to call custom methods when your objects are serialized. you can impelment some custom logic in there and decouple the serializing process from the actual datatypes.

Share Improve this answer Follow

answered Oct 6, 2008 at 14:13

Joachim Kerschbaumer
9,841 ● 7 ● 51 ● 84

How, exactly, can one do this, if those delegates don't make the SerializationInfo object available? – skolima Mar 11, 2009 at 14:44

well, you dont need a serializationinfo object as you should get a streamingcontext passed to the methods decorated with the OnSerializing attribute. see msdn.microsoft.com/en-us/library/... for details. – Joachim Kerschbaumer Mar 12, 2009 at 6:48

3

Have you thought about simply using a Nullable date time

```
public DateTime? SomeDateTime {get; set;}
```

This way you can actually have a null value as a legitimate value in your class. Typically you want to avoid custom serialization whenever possible. Once you do custom serialization such as implementing **ISerializable**, then you are stuck with it, and all other derived classes are stuck with it.

Remember what a pain in the butt it is to have to always override the ISerializable members for custom exceptions? This is because System.Exception implements ISerializable and therefore all derived exceptions (that means everything) must implement those members if you ever expect them to cross AppDomains.

Share

Improve this answer

Follow

edited Oct 6, 2008 at 11:14

answered Oct 6, 2008 at 11:08

Josh

44.9k ● 7 ● 104 ● 124

I'm consuming this via a web service so it is automatically converting DateTime? to DateTime.MinValue. I'll correct the question. – Alex Angas Oct 6, 2008 at 11:58

0

Well, you can use a "bool ShouldSerializeSomeDateTime()" method to enable/disable serialization of individual members, but I don't think that is quite what you need. Another common option is to add a member that does the format itself:

public string SomeDateTimeFormatted { get { return theField == DateTime.MinValue ? "" : theField.ToString("R");} // or whatever format set { ... opposite ...} }

It is nicer to stick with the inbuilt serialization if you can, though - in part to reduce the amount of code you need to write. Josh's suggestion for a nullable DateTime (DateTime?) is a good one, although it might still not be quite empty-string vs formatted-string - I expect it would use the xsi:nil markup.

Share Improve this answer Follow

answered Oct 6, 2008 at 11:14

Marc Gravell
1.1m ● 273 ● 2.6k ● 3k