pawelpabich / Private Setter
Created 3 years ago

Trying to serialize and deserialize a class with private setter in the base class

Private Setter

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

namespace ConsoleApplication12
{
    class Program
    {
        static void Main(string[] args)
        {
            var original = new Derived("Base", "Derived");

            var serializerSettings = new JsonSerializerSettings();
            serializerSettings.ContractResolver = new IncludePrivateStateContractResolver();
            var jsonCopy = JsonConvert.SerializeObject(original, serializerSettings);
            var clonedObject = JsonConvert.DeserializeObject<Derived>(jsonCopy, serializerSettings);

            Console.WriteLine(clonedObject.DerivedProperty);
        }
    }

    public class IncludePrivateStateContractResolver : DefaultContractResolver
    {
        protected override List<MemberInfo> GetSerializableMembers(Type objectType)
        {
            const BindingFlags BindingFlags = BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public;
            var properties = objectType.GetProperties(BindingFlags);//.Where(p => p.HasSetter() && p.HasGetter());
            var fields = objectType.GetFields(BindingFlags);

            var allMembers = properties.Cast<MemberInfo>().Union(fields);
            return allMembers.ToList();
        }

        protected override JsonProperty CreateProperty(MemberInfo member, MemberSerialization memberSerialization)
        {
            var prop = base.CreateProperty(member, memberSerialization);

            if (!prop.Writable)
            {
                var property = member as PropertyInfo;
                if (property != null)
                {
                    prop.Writable = property.HasSetter();
                }
                else
                {
                    var field = member as FieldInfo;
                    if (field != null)
                    {
                        prop.Writable = true;
                    }
```

```
                }
            }

            if (!prop.Readable)
            {
                var field = member as FieldInfo;
                if (field != null)
                {
                    prop.Readable = true;
                }
            }

            return prop;
        }
    }

    public class Derived : Base
    {
        public Derived(string baseValue, string derived):base(baseValue)
        {
            DerivedProperty = derived;
        }

        public string DerivedProperty { get; private set; }
    }


    public class Base
    {
        public string BaseProperty { get; private set; }

        public Base(string value)
        {
            BaseProperty = value;
        }
    }

    public static class TypeExtensions
    {
        public static bool HasSetter(this PropertyInfo property)
        {
            //In this way we can check for private setters in base classes
            return property.DeclaringType.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public)
                                        .Any(m => m.Name == "set_" + property.Name);
        }

        public static bool HasGetter(this PropertyInfo property)
        {
            //In this way we can check for private getters in base classes
            return property.DeclaringType.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public)
                                        .Any(m => m.Name == "get_" + property.Name);
        }
    }


}
```