De/Serialize C# data object with nested derived objects to/from JSON

Asked 6 years, 1 month ago Modified 6 years, 1 month ago Viewed 3k times

-1

Is there C# library that can serialize tree-like structure of .NET/C# strongly typed objects in a single call?

EDIT: I want to be able to deserialize JSON into variable of type object (or some other root class, but object would be preferable) and then find out what I just deserialized by calling GetType() on this variable. So type of each instance of each class (except bool/int/float/string) and each List<> should be stored as part of serialized JSON and retrieved automatically.

For example if I have these C# classes:

```
public class House
{
    public int Number;
    public List<Room> Rooms;
}

public class Room
{
    public string Name;
    public int Floor;
}

public class Bathroom : Room
{
    public bool IsWet;
}
```

And initialize my in-memory data structure like this (note that I use generic List class, not C# array):

```
House myHouse = new House();
myHouse.Number = 13;
myHouse.Rooms = new List<Room>();
myHouse.Rooms.Add(new Room());
myHouse.Rooms[0].Name = "some room";
myHouse.Rooms[0].Floor = 0;
myHouse.Rooms.Add(new Bathroom());
myHouse.Rooms[0].Name = "other room";
myHouse.Rooms[0].Floor = 3;
myHouse.Rooms[0].IsWet = true;
```

After that I want to be able to call something like this:

```
string jsonHouse = JSON.Serialize(myHouse);
```

To get (for example) this in my jsonHouse variable *(EDIT: note that it stores type of each class as part of JSON. Also note the instance of derived class BathRoom stored in List)*:

```
{
  "class": "House",
  "Number": "13",
  "Rooms": [
    {
      "class": "Room",
      "Name": "some room"
      "Floor": "0",
    },
    {
      "class": "Bathroom",
      "Name": "other room"
      "Floor": "0",
      "IsWet": "true",
    },
  ],
}
```

and finally call something like this:

```
House copyOfMyHouse = JSON.Deserialize(jsonHouse);
```

EDIT: this (right above) call seems to be misleading. instead of above call I should expect deserialization to be like this:

```
object copyOfMyHouse = JSON.Desrialize(jsonHouse);
```

To get instance of class House in my copyOfMyHouse variable (of type object) without specifying to JSON.Deserialize what type do I expect from it - I may not know it in advance.

to get exact copy of what I had in myHouse variable, with an instance of List generic class containing references to exact copies of two instances of Room class.

Is there any JSON serializer written in C# that can do this?

Preferably FOSS/FLOSS one that can be (legally) used in commercial projects.

Format of JSON string may be different from what I used above as example (e.g. your proposed solution may store List as JSON objects instead of arrays, other text formats like BSON or XML are acceptable) Single-call use for any number of nesting levels is mandatory - otherwise any JSON serializer would fit. Also ability to serialize Dictionary to JSON object is desirable.

Please ask clarifying questions if my description of what I look for is unclear.

c# json

Triet Doan
10.6k 8 31 67

coder
185 3 12

3 Answers

Sorted by:
Trending sort available
Highest score (default)

2

Basically, you want to serialize/deserialize an object containing derived types. The solution is using Json.NET. It's very simple and easy to use. Here is an example with your data:

```
House myHouse = new House();
myHouse.Number = 13;

myHouse.Rooms = new List<Room>();

Room room1 = new Room();
room1.Name = "some room";
room1.Floor = 0;
myHouse.Rooms.Add(room1);

Room room2 = new Room();
room2.Name = "other room";
room2.Floor = 3;
myHouse.Rooms.Add(room2);

Bathroom bathroom = new Bathroom();
bathroom.Name = "Bathroom";
bathroom.Floor = 2;
bathroom.IsWet = true;
myHouse.Rooms.Add(bathroom);

JsonSerializerSettings settings = new JsonSerializerSettings();
settings.TypeNameHandling = TypeNameHandling.Auto;

string json = JsonConvert.SerializeObject(myHouse, settings);
Console.WriteLine("Serialize finished!");

House house = JsonConvert.DeserializeObject<House>(json, settings);
Console.WriteLine($"House number: {house.Number}; Total rooms:
{house.Rooms.Count}");

foreach (Room room in house.Rooms)
{
    if (room is Bathroom)
    {
        var temp = room as Bathroom;
        Console.WriteLine($"Room name: {temp.Name}, wet: {temp.IsWet}");
    }
    else
    {
        Console.WriteLine($"Room name: {room.Name}");
    }
}
Console.WriteLine("Deserialize finished!");

Console.ReadLine();
```

```
{
  "Number": 13,
  "Rooms": [
    {
      "Name": "some room",
      "Floor": 0
    },
    {
      "Name": "other room",
      "Floor": 3
    },
    {
      "$type": "DemoApp.Bathroom, DemoApp",
      "IsWet": true,
      "Name": "Bathroom",
      "Floor": 2
    }
  ]
}
```

And you got back the object after deserialize the string.

Share Follow edited Jun 20, 2016 at 17:47 answered Jun 17, 2016 at 12:35

Triet Doan
10.6k 8 31 67

will it work with List<string> instead of string[] ? – coder Jun 17, 2016 at 13:05

Surely it will :) Take a look at this post. – Triet Doan Jun 17, 2016 at 14:52

it requires to explicitly specify the type of collection to deserialize to - how would it handle nested Lists and/or classes? Like LIst<House> where House has LIst<Room> as one of its fields – coder Jun 18, 2016 at 6:50

Thanks! Can it store the type of each class as part of JSON? i.e. how will it handle some class Bathroom derived from class room being added to Rooms List<> (see my edits to initial question)? – coder Jun 18, 2016 at 7:27

1 Oops, sorry, I misread your question. Please check my edited answer. Notice the `JsonSerializerSettings` . – Triet Doan Jun 20, 2016 at 17:48

1

Newtonsoft.JSON (sometimes known as JSON.NET) is good and available as a NuGet package. Then you can write something like:

```
var value = new { ID = 1, Name="test"};
string serialized = JsonConvert.SerializeObject(value);
```

And to deserialize (the simplest option, but there are lots of good ways to do it):

```
result = JsonConvert.DeserializeObject(jsonResponse);
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs. Sign up

52.5k 13 48 61

will it work with non-anonymous types? if yes then will it preserve types after serialization-deserialization cycle? – coder Jun 17, 2016 at 13:03

1 Yes it will. Obviously when you deserialize you need to know what type you are expecting. You can use generics as well (e.g. so that you can write the serialization routine once and have lots of different methods call it to serialize different types) – ADyson Jun 17, 2016 at 14:36

ADyson, please check EDIT at the beginning of my initial question, part of what I want is to be able to find out what I just deserialized. – coder Jun 18, 2016 at 6:58

1 @Tonklin You could use generics. e.g. `public R CallWebService<U, R>(string uri, U value) { R result; ... string data = JsonConvert.SerializeObject(value); ...[Call webservice]... result = JsonConvert.DeserializeObject<R>(response); return result; }` . Doing it this way means the code that calls the webservice doesn't need to know explicitly what data types it's dealing with. Only the code which calls it needs to know. So you can call the same webservice method from many different places using different data types. Does that help? – ADyson Jun 20, 2016 at 8:34

what you described is certainly useful in some situations yet what I look for is ability to know the types of data objects from their JSON representation, not ability to ignore them... – coder Jun 20, 2016 at 9:19

1

While other answers already provide good solutions, here's one using the DataContractJsonSerializer class.

```
var jsonString = "";
var jsonSerializer = new DataContractJsonSerializer(typeof(House));

using (var memoryStream = new MemoryStream())
using (var streamReader = new StreamReader(memoryStream))
{
    jsonSerializer.WriteObject(memoryStream, myHouse);
    memoryStream.Position = 0;
    jsonString = streamReader.ReadToEnd();
}

Console.Write("JSON form of Person object: ");
Console.WriteLine(jsonString);
```

Share Follow

answered Jun 17, 2016 at 12:40
Abbas
13.8k 6 40 69

seems grossly overcomplicated to me, despite the fact that it can be adapted for single-call use by tucking this code into custom static method – coder Jun 17, 2016 at 13:01

Abbas, will DataContractJsonSerializer handle nested collections like List<>? – coder Jun 18, 2016 at 7:05

Yes, it will also handle collections. – Abbas Jun 20, 2016 at 8:16

Abbas, for example: "Rooms": [{ "class": "Room", "Name": "some room" "Floor": "0", }, - in this piece of JSON there is a field "class" for that, so that if element in serialized List will be of some type derived from Room (not Room itself) it will be deserialized as na instance of that same derived class, not as base class Room – coder Jun 21, 2016 at 13:21