

Json.net serialize/deserialize derived types?

Asked 10 years, 7 months ago Modified 2 years, 5 months ago Viewed 131k times



json.net (newtonsoft)

125

I am looking through the documentation but I can't find anything on this or the best way to do it.



23



```
public class Base
{
    public string Name;
}
public class Derived : Base
{
    public string Something;
}
```

```
JsonConvert.Deserialize<List<Base>>(text);
```

Now I have Derived objects in the serialized list. How do I deserialize the list and get back derived types?

[c#](#) [json](#) [serialization](#) [json.net](#)

Share Improve this question

Follow

edited Mar 20, 2014 at 8:21



[Madmenyo](#)

8,152 7 50 97

asked Dec 14, 2011 at 23:07



[Will](#)

9,453 7 44 76

That isn't how the inheritance works. You can specify `JsonConvert.Deserialize<Derived>(text);` to include the Name field. Since Derived **IS A** Base (not the other way around), Base doesn't know anything about Derived's definition. – [M.Babcock](#) Dec 14, 2011 at 23:13

- 1 Sorry, clarified a bit. The issue is I have a list which contains both base and derived objects. So I need to figure out how I tell newtonsoft how to deserialize the derived items. – [Will](#) Dec 14, 2011 at 23:18

I did you solve this. I have the same problem – [Luis Carlos Chavarría](#) Jul 29, 2012 at 20:18

4 Answers

Sorted by:

Trending sort available

Highest score (default)



You have to enable Type Name Handling and pass that to the (de)serializer as a settings parameter.

116



```
Base object1 = new Base() { Name = "Object1" };
Derived object2 = new Derived() { Something = "Some other thing" };
List<Base> inheritanceList = new List<Base>() { object1, object2 };
```



```
JsonSerializerSettings settings = new JsonSerializerSettings { TypeNameHandling =
```

```

TypeNameHandling.All };
string Serialized = JsonConvert.SerializeObject(inheritanceList, settings);
List<Base> deserializedList = JsonConvert.DeserializeObject<List<Base>>
(Serialized, settings);

```

This will result in correct deserialization of derived classes. A drawback to it is that it will name all the objects you are using, as such it will name the list you are putting the objects in.

Share Improve this answer Follow

edited Feb 7, 2019 at 21:21

answered Mar 18, 2014 at 17:39



Lee Taylor

7,631 16 32 47



Madmenyo

8,152 7 50 97

- 33 +1. I was googling for 30 minutes until I actually found out that you need to use same settings for SerializeObject & DeserializeObject. I assumed it would use \$type implicitly if it is there when deserializing, silly me. – Erti-Chris Eelmaa Jul 13, 2015 at 18:53
- 28 TypeNameHandling.Auto will do it too, and is nicer because it doesn't write the instance type name when it matches the type of the field/property, which is often the case for most fields/properties. – Roman Starkov Feb 16, 2016 at 10:16
- 5 This doesn't work when deserialization is performed on another solution/project. On serialization the name of the Solution is embedded within as type: "SOLUTIONNAME.Models.Model". On deserialization on the other solution it will throw "JsonSerializationException: Could not load assembly 'SOLUTIONNAME'". – Jebathon May 22, 2020 at 17:06
- 2 I never know it could be so easy! I dug through Google and damn near pull all my hair out trying to figure out how to write a custom serializer/deserializer! Then I finally came across this answer, and finally solved it! You're my lifesaver! Thanks trillions! – Noob001 Oct 27, 2021 at 16:15



55



If you are storing the type in your text (as you should be in this scenario), you can use the JsonSerializerSettings .

See: [how to deserialize JSON into IEnumerable<BaseType> with Newtonsoft JSON.NET](#)

Be careful, though. Using anything other than TypeNameHandling = TypeNameHandling.None could open yourself up to [a security vulnerability](#).

Share Improve this answer Follow

edited Oct 2, 2018 at 3:03

answered Jan 6, 2012 at 20:02



kamranicus

4,067 2 36 52

- 28 You can also use TypeNameHandling = TypeNameHandling.Auto - this will add a \$type property ONLY for instances where the declared type (i.e. Base) does not match the instance type (i.e. Derived). This way, it doesn't bloat your JSON as much as TypeNameHandling.All . – AJ Richardson Mar 25, 2015 at 13:18

I keep receiving Error resolving type specified in JSON '...', ...'. Path '\$type', line 1, position 82. Any ideas? – briba Oct 21, 2016 at 23:09

- 4 Be careful when using this on a public endpoint as it opens up security issues: [alphanot.com/security/blog/2017/net/...](#) – gjvdkamp Aug 13, 2018 at 10:59

- 2 @gjvdkamp JEEZ thanks for this, I did not know about this. Will add to my post. – kamranicus Oct 2,

2018 at 3:02



Since the question is so popular, it may be useful to add on what to do if you want to control the type property name and its value.

29



The long way is to write custom `JsonConverter`s to handle (de)serialization by manually checking and setting the type property.



A simpler way is to use [JsonSubTypes](#), which handles all the boilerplate via attributes:

```
[JsonConverter(typeof(JsonSubtypes), "Sound")]
[JsonSubtypes.KnownSubType(typeof(Dog), "Bark")]
[JsonSubtypes.KnownSubType(typeof(Cat), "Meow")]
public class Animal
{
    public virtual string Sound { get; }
    public string Color { get; set; }
}

public class Dog : Animal
{
    public override string Sound { get; } = "Bark";
    public string Breed { get; set; }
}

public class Cat : Animal
{
    public override string Sound { get; } = "Meow";
    public bool Declawed { get; set; }
}
```

Share Improve this answer Follow

edited Nov 6, 2019 at 10:03

answered Mar 19, 2019 at 11:41



rzippo

849 11 22



- 7 I get the need, but I'm not a fan of having to make the base class aware of all the "KnownSubType"s...
– [Matt Knowles](#) Apr 4, 2019 at 23:18
- 2 There are other options if you look at the documentation. I only provided the example I like more.
– [rzippo](#) Apr 5, 2019 at 9:58
- 2 This is the safer approach that doesn't expose your service to load arbitrary types upon de-serialization.
– [David Burg](#) Dec 19, 2019 at 23:40
- 1 Where is `JsonSubtypes` even defined? I'm using Newtonsoft.Json Version 12.0.0.0 and have no reference to `JsonSubtypes`, `JsonSubTypes` nor `JsonSubtypesConverterBuilder` (mentioned in that article). – [Matt Arnold](#) Nov 16, 2020 at 18:37
- 1 @MattArnold It's a separate Nuget package. – [rzippo](#) Nov 16, 2020 at 19:33



Use this [JsonKnownTypes](#), it's very similar way to use, it just add discriminator to json:

9

```
[JsonConverter(typeof(JsonKnownTypeConverter<BaseClass>))]
[JsonKnownType(typeof(Base), "base")]
```

```
[JsonKnownType(typeof(Derived), "derived")]
public class Base
{
    public string Name;
}
public class Derived : Base
{
    public string Something;
}
```

Now when you serialize object in json will be add "\$type" with "base" and "derived" value and it will be use for deserialize

Serialized list example:

```
[
  {"Name":"some name", "$type":"base"},
  {"Name":"some name", "Something":"something", "$type":"derived"}
]
```

[Share](#) [Improve this answer](#) [Follow](#)

answered Feb 19, 2020 at 9:25



[Dmitry](#)

182 1 3