

[پس از بررسی مقدماتی](#) امکانات کتابخانه‌ی JSON.NET، در ادامه به تعدادی از تنظیمات کاربردی آن با ذکر مثال‌هایی خواهیم پرداخت.

### گرفتن خروجی از CamelCase در JSON.NET

یک سری از کتابخانه‌های جاوا اسکریپتی سمت کلاینت، به نام‌های خواص [CamelCase](#) نیاز دارند و حالت پیش فرض اصول نامگذاری خواص در دات نت عکس آن است. برای مثال بجای UserName به userName نیاز دارند تا بتوانند صحیح کار کنند. روش اول حل این مشکل، استفاده از ویژگی JsonProperty بر روی تک تک خواص و مشخص کردن نام‌های مورد نیاز کتابخانه‌ی جاوا اسکریپتی به صورت صریح است. روش دوم، استفاده از تنظیمات ContractResolver می‌باشد که با تنظیم آن به CamelCasePropertyNameContractResolver به صورت خودکار به تمامی خواص به صورت یکسانی اعمال می‌گردد:

```
var json = JsonConvert.SerializeObject(obj, new JsonSerializerSettings
{
    ContractResolver = new CamelCasePropertyNamesContractResolver()
});
```

### درج نام‌های المان‌های یک Enum در خروجی JSON

اگر یکی از عناصر در حال تبدیل به JSON، از نوع enum باشد، به صورت پیش فرض مقدار عددی آن در JSON نهایی درج می‌گردد:

```
using Newtonsoft.Json;

namespace JsonNetTests
{
    public enum Color
    {
        Red,
        Green,
        Blue,
        White
    }

    public class Item
    {
        public string Name { set; get; }
        public Color Color { set; get; }
    }

    public class EnumTests
    {
        public string GetJson()
        {
            var item = new Item
            {
                Name = "Item 1",
                Color = Color.Blue
            };

            return JsonConvert.SerializeObject(item, Formatting.Indented);
        }
    }
}
```

با این خروجی:

```
{
  "Name": "Item 1",
  "Color": 2
}
```

اگر علاقمند هستید که بجای عدد 2، دقیقا مقدار Blue در خروجی JSON درج گردد، می‌توان به یکی از دو روش ذیل عمل کرد:  
الف) مزین کردن خاصیت از نوع enum به ویژگی JsonSerializer.Converters از نوع StringEnumConverter:

```
[JsonConverter(typeof(StringEnumConverter))]
public Color Color { set; get; }
```

ب) و یا اگر می‌خواهید این تنظیم به تمام خواص از نوع enum به صورت یکسانی اعمال شود، می‌توان نوشت:

```
return JsonConvert.SerializeObject(item, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    Converters = { new StringEnumConverter() }
});
```

## تهیه خروجی JSON از مدل‌های مرتبط، بدون Stack overflow

دو کلاس گروه‌های محصولات و محصولات ذیل را در نظر بگیرید:

```
public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual ICollection<Product> Products { get; set; }

    public Category()
    {
        Products = new List<Product>();
    }
}

public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual Category Category { get; set; }
}
```

این نوع طراحی در Entity framework بسیار مرسوم است. در اینجا طرف‌های دیگر یک رابطه، توسط خاصیتی virtual معرفی می‌شوند که به آن‌ها خواص راهبری یا navigation properties هم می‌گویند.  
با توجه به این دو کلاس، سعی کنید مثال ذیل را اجرا کرده و از آن، خروجی JSON تهیه کنید:

```
using System.Collections.Generic;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace JsonNetTests
{
    public class SelfReferencingLoops
    {
        public string GetJson()
        {
            var category = new Category
            {
                Id = 1,
                Name = "Category 1"
            };
            var product = new Product
```

```

        {
            Id = 1,
            Name = "Product 1"
        };

        category.Products.Add(product);
        product.Category = category;

        return JsonConvert.SerializeObject(category, new JsonSerializerSettings
        {
            Formatting = Formatting.Indented,
            Converters = { new StringEnumConverter() }
        });
    }
}

```

برنامه با این استثناء متوقف می‌شود:

```

An unhandled exception of type 'Newtonsoft.Json.JsonSerializationException' occurred in
Newtonsoft.Json.dll
Additional information: Self referencing loop detected for property 'Category' with type
'JsonNetTests.Category'. Path 'Products[0]'.

```

اصل خطای معروف فوق «Self referencing loop detected» است. در اینجا کلاس‌هایی که به یکدیگر ارجاع می‌دهند، در حین عملیات Serialization سبب بروز یک حلقه‌ی بازگشتی بی‌نهایت شده و در آخر، برنامه با خطای stack overflow خاتمه می‌یابد.

راه حل اول:

به تنظیمات JSON.NET، مقدار `ReferenceLoopHandling = ReferenceLoopHandling.Ignore` را اضافه کنید تا از حلقه‌ی بازگشتی بی‌پایان جلوگیری شود:

```

return JsonConvert.SerializeObject(category, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
    Converters = { new StringEnumConverter() }
});

```

راه حل دوم:

به تنظیمات JSON.NET، مقدار `PreserveReferencesHandling = PreserveReferencesHandling.Objects` را اضافه کنید تا مدیریت ارجاعات اشیاء توسط خود JSON.NET انجام شود:

```

return JsonConvert.SerializeObject(category, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    PreserveReferencesHandling = PreserveReferencesHandling.Objects,
    Converters = { new StringEnumConverter() }
});

```

خروجی حالت دوم به این شکل است:

```

{
  "$id": "1",
  "Id": 1,
  "Name": "Category 1",
  "Products": [
    {
      "$id": "2",
      "Id": 1,
      "Name": "Product 1",
      "Category": {
        "$ref": "1"
      }
    }
  ]
}

```

```
]
}
```

همانطور که ملاحظه می‌کنید، دو خاصیت `id$` و `ref$` توسط JSON.NET به خروجی JSON اضافه شده‌است تا توسط آن بتواند ارجاعات و نمونه‌های اشیاء را تشخیص دهد.

## نظرات خوانندگان

نویسنده: وحید نصیری  
تاریخ: ۱۵:۱۷ ۱۳۹۳/۰۶/۲۳

گرفتن خروجی مرتب شده بر اساس نام خواص (جهت مقاصد نمایشی):

تعریف DefaultContractResolver:

```
public class OrderedContractResolver : DefaultContractResolver
{
    protected override IList<JsonProperty> CreateProperties(
        System.Type type, MemberSerialization memberSerialization)
    {
        return base.CreateProperties(type, memberSerialization).OrderBy(p =>
p.PropertyName).ToList();
    }
}
```

و بعد معرفی آن به نحو ذیل:

```
return JsonConvert.SerializeObject(data, new JsonSerializerSettings
{
    ContractResolver = new OrderedContractResolver()
});
```