# JSON – JavaScript Object Notation

* Data Interchange Format
* Open Standard
* Human readable data objects in text represented as Attribute Value Pairs
* Use to pass data between a server and a web application
* Fat Free Alternative to XML
* Used heavily by REST APIs
* Not natively supported in C#

**JSON Specification**

JSON syntax is derived from JavaScript object notation syntax:

* Data is in name/value pairs.
  + Example: "name" : "John"
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

In JSON, values must be one of the following data types:

* Four primitive types:
  + Strings
  + Numbers
  + Booleans
  + Null
* Two structured types:
  + Objects
  + Arrays

Examples:

* Number
  + "courses": 5,
  + "weight": 205.5,
  + "x": 7.02e+23
* String
  + "name": "John Smith",
* Boolean
  + "developer": true,
* Array
  + "counts": [
    - "one",
    - 2,
    - "three",
    - 4
    - ]
* Object (surrounded by curly braces)
  + "author": {
    - "firstName": "John",
    - "lastName": "Smith"
    - }
* null
  + "healthyDiet": null,

JSON values cannot be one of the following data types:

* a function
* a date
* undefined

**JSON vs XML**

JSON is Like XML Because

* Both JSON and XML are "self describing" (human readable)
* Both JSON and XML are hierarchical (values within values)
* Both JSON and XML can be parsed and used by lots of programming languages
* Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

* JSON doesn't use end tag
* JSON is shorter
* JSON is quicker to read and write
* JSON can use arrays

The biggest difference is:

 XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

**Sample JSON/XML:**

The following JSON and XML examples define an employees object, with an array of 3 employees:

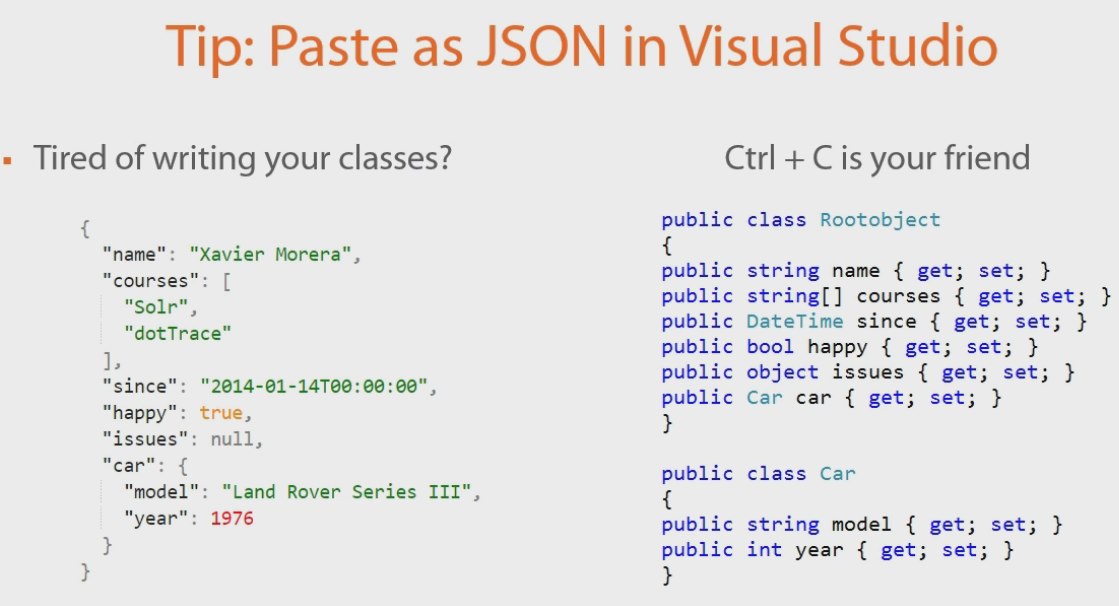
|  |  |
| --- | --- |
| JSON Example | XML Example |
| {"employees":[     { "firstName":"John", "lastName":"Doe" },     { "firstName":"Anna", "lastName":"Smith" },     { "firstName":"Peter", "lastName":"Jones" } ]} | <employees>     <employee>         <firstName>John</firstName> <lastName>Doe</lastName>     </employee>     <employee>         <firstName>Anna</firstName> <lastName>Smith</lastName>     </employee>     <employee>         <firstName>Peter</firstName> <lastName>Jones</lastName>     </employee> </employees> |

**Json.NET**

* <http://www.newtonsoft.com/json>
* Open Source
* In GitHub
* Download from NuGet
* Supports converting between XML and JSON
* LINQ to JSON
* JSON Path – Like XML Path
* Documentation and Code Samples

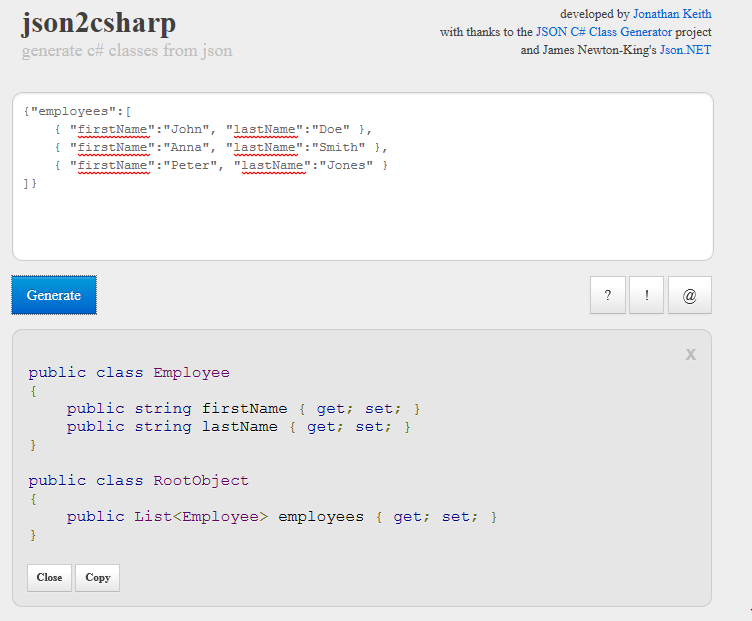
**Serialization / Deserialization / LINQ**

|  |  |
| --- | --- |
|  |  |
|  |  |



Copy the JSON and within Visual Studio select Edit | Paste Special | Paste JSON As Classes. You can do this for XML also.

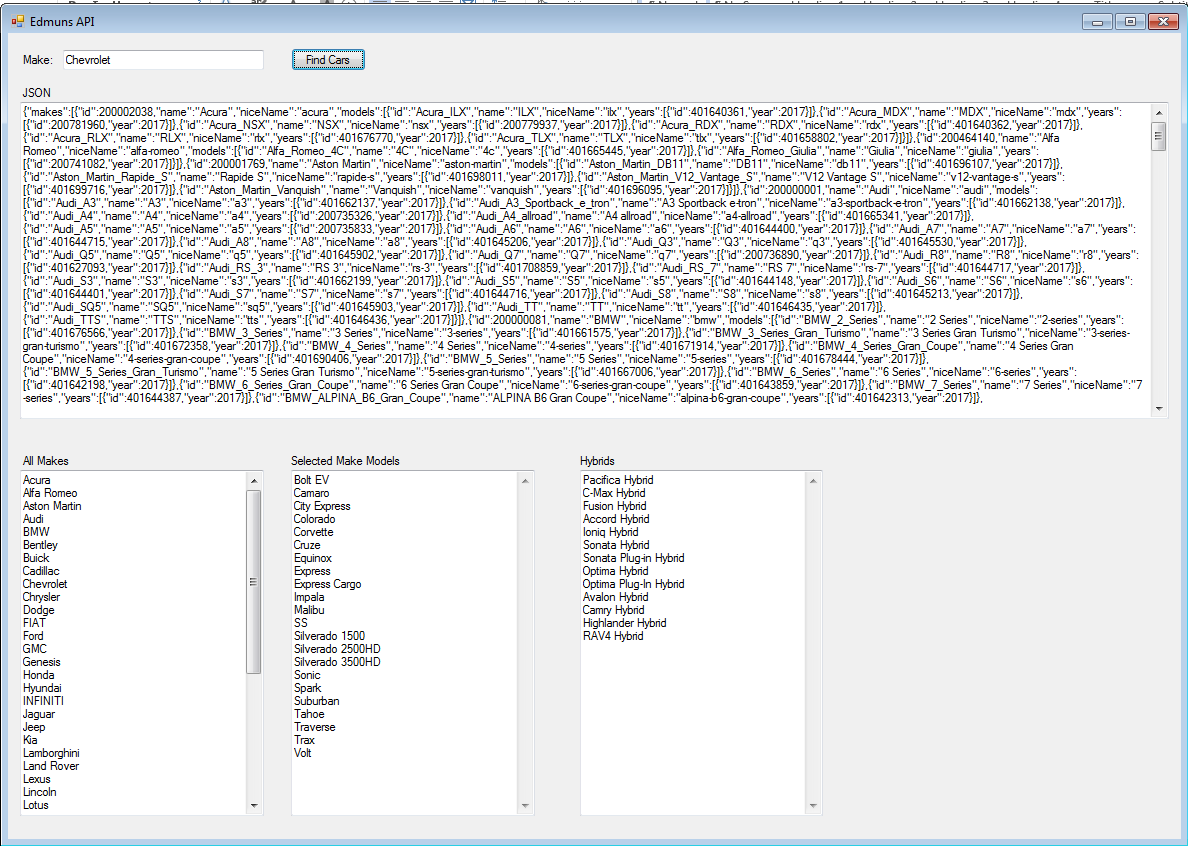
<http://json2csharp.com/> This site allows you to paste JSON into a window and generate C# classes.



### Edmunds API Console

The API Console (*aka I/O Docs*) is an interactive tool that enables you to interact with the Edmunds API without writing a single line of code! Discover, learn, test and debug live API calls within seconds.

I wrote a small application to call the Edmunds API to retrieve data and display the original JSON along with selected portions of the data.



**Settings & Attributes (40m 1s)**

You can control the serialized output by setting Attributes on your classes.

See Demo: Attributes

See Demo: MemberSerialization OptIn and OptOut

See Demo: JsonConverter Attribute

See Demo: JsonConstructor Attribute

See Demo: Conditional Serialization

**Performance Tips**

Demo: Serialize & Deserialize Manually

You can use JsonTextReader and JsonTextWriter to speed up reading/writing JSON.

Demo: JSON Fragments

Deserialize only what you need

Serialize only what you want

Demo 17: Attributes for Performance

Serializae and Deserialize only what you need

**LINQ to JSON**

API to work with JSON objects.

Language-Integrated Query.

Under "Newtonsoft.JSON.Linq" namespace

Create JSON Using LINQ

1. Imperative
2. Declarative
3. FromObject

Demo 19: Create JSON using LINQ

Imperative: You create JSON object step by step. Very easy.

Declarative: You write code that describes what you want, but not individual steps to get there.

FromObject: Use "ExpandoObject" to add properties by name. Can be error prone because it bypasses compile time checking.

**Parsing JSON Using LINQ**

JObject.Parse

JArray.Parse

Annotations – JSON.Net specific. Usefor for auditing and logging scenarios.

URLs and Training

|  |  |
| --- | --- |
| URL / Training Material | Details |
| <https://www.w3schools.com/js/js_json_intro.asp> | Some of the details above were from this location.  Good introduction to JSON. |
| <https://app.pluralsight.com/> | Has a very good course titled:  "Getting Started with JSON in C# Using Json.NET" |
| <http://www.jsoneditoronline.org/> | JSON online editor |
|  |  |
| <http://edmunds.mashery.com/io-docs> | Edmunds API Console |
|  |  |