JSON

JavaScript Object Notation

Is described in RFC 4627

JSON

- Is a lightweight data-interchange format
- Is easy for humans to read and write
- Is easy for machines to parse and generate
- **Is based on a subset of JavaScript**
- Is a text format that is completely language independent
  - but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others
- These properties make JSON an ideal data-interchange language
  - And JSON is widely used in data communication
  - And several NoSQL databases uses JSON (or BSON) as data storage format.
    - BSON is a binary representation of JSON

History

- Douglas Crockford was the first to specify and popularize the JSON format
- JSON was used at State Software, a company co-founded by Crockford, starting around 2001
- The JSON.org website was launched in 2002
- In 2005 Yahoo began offering some of its web services in JSON
- In 2006 Google started offering JSON feeds for its GData web protocol
- **Is described in RFC 4627**

JSON Structure

- JSON is built on two structures:
  - A collection of name/value pairs
    - In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array
  - An ordered list of values.
    - In most languages, this is realized as an array, vector, list, or sequence

```
{
    name : "10gen HQ",
    address : "578 Broadway 7th Floor",
    city : "New York",
    zip : "10011",
    tags : [ "business", "tech" ]
}
```

Native JSON

- **All modern browsers have native JSON support, via:**
  - **JSON.parse() and**
  - **JSON.stringify()**

- They were added to the Fifth Edition of the ECMAScript standard

- For older browsers, a compatible JavaScript library is available at JSON.org

Json.NET

- Json.NET is a popular high-performance JSON framework for .NET
- Features:
  - Flexible JSON serializer for converting between .NET objects and JSON
  - LINQ to JSON for manually reading and writing JSON
  - High performance, faster than .NET's built-in JSON serializers
  - Write indented, easy to read JSON
  - Convert JSON to and from XML
- The serializer is a good choice when the JSON you are reading or writing maps closely to a .NET class
- LINQ to JSON is good for situations where you are only interested in getting values from JSON, you don't have a class to serialize or deserialize to, or the JSON is radically different from your class and you need to manually read and write from your objects

Serialization Example

```
Product product = new Product();
product.Name = "Apple";
product.Expiry = new DateTime(2008, 12, 28);
product.Price = 3.99M;
product.Sizes = new string[] { "Small", "Medium", "Large" };
string json = JsonConvert.SerializeObject(product);
// . . .

Product deserializedProduct =
             JsonConvert.DeserializeObject<Product>(json);
```

- The serializer is a good choice when the JSON you are reading or writing maps closely to a .NET class

```
{
  "Name": "Apple",
  "Expiry": "2008-12-28T00:00:00",
  "Price": 3.99,
  "Sizes": [
    "Small",
    "Medium",
    "Large"
  ]
}
```

Getting JSON Values

```
string json = @"{
  ""Name"": ""Apple"",
  ""Expiry"": ""2008-12-28T00:00:00"",
  ""Price"": 3.99,
  ""Sizes"": [
    ""Small"",
    ""Medium"",
    ""Large""”
  ]}";

JObject o = JObject.Parse(json);

string name = (string)o["Name"];
// Apple

sizes = (JArray)o["Sizes"];
string smallest = (string)sizes[0];
// Small
```

- `Jobject` is good for situations where:
  - you are only interested in getting values from JSON
  - you don't have a class to serialize or deserialize to
  - or the JSON is radically different from your class and you need to manually read and write from your objects

LINQ to JSON

- JObject/JArray can also be queried using LINQ
  - Children() returns the children values of a JObject/JArray as an IEnumerable<JToken> that can then be queried with the standard Where/OrderBy/Select LINQ operators

```
string json = @"{. . .}";
JObject rss = JObject.Parse(json);

var postTitles =
        from p in rss["channel"]["item"]
        select (string)p["title"];

foreach (var item in postTitles)
{
  Console.WriteLine(item);
}
```