JSON

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. Conventions used by JSON are known to programmers, which include C, C++, Java, Python, Perl, etc.

- JSON stands for JavaScript Object Notation.
- The format was specified by Douglas Crockford.
- It was designed for human-readable data interchange.
- It has been extended from the JavaScript scripting language.
- The filename extension is .json.
- JSON Internet Media type is application/json.
- The Uniform Type Identifier is public.json.

Uses of JSON

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

Characteristics of JSON

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.

Simple Example in JSON

Example to use JSON to store information related to books based on their topic and edition.

```
{
  "book": [
  {"id":"01",  "language": "Java",  "edition": "third",  "author": "Herbert Schildt"},
  {"id":"07",  "language": "C++",  "edition": "second",  "author": "E.Balagurusamy"}
  ]
}
```

Store the following file as **jsonex1.html** and click on it :

```
<html>
  <head><title>JSON example</title>
    <script language = "javascript" >
       var object1 = { "language" : "Java", "author"  : "herbert schildt" };
      document.write("<h1>JSON with JavaScript example</h1>");
      document.write("<br>");
      document.write("<h3>Language = " + object1.language+"</h3>");
      document.write("<h3>Author = " + object1.author+"</h3>");

      var object2 = { "language" : "C++", "author"  : "E-Balagurusamy" };
      document.write("<br>");
      document.write("<h3>Language = " + object2.language+"</h3>");
      document.write("<h3>Author = " + object2.author+"</h3>");

      document.write("<hr />");
      document.write(object2.language + " programming language can be studied " + "from book
written by " + object2.author);
      document.write("<hr />");
    </script>
  </head>

  <body></body>
</html>
```

JSON - Syntax

Let's have a quick look at the basic syntax of JSON. JSON syntax is basically considered as a subset of JavaScript syntax; it includes the following –

- Data is represented in name/value pairs.
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).
- Square brackets hold arrays and values are separated by ,(comma).

Simple example –

```
{   "book": [
   { "id":"01",  "language": "Java",  "edition": "third",  "author": "Herbert Schildt",},
   { "id":"07",  "language": "C++",  "edition": "second",  "author": "E.Balagurusamy", }
 ]  }
```

JSON supports the following two data structures –

- Collection of name/value pairs – This Data Structure is supported by different programming languages.
- Ordered list of values – It includes array, list, vector or sequence etc.

JSON - DataTypes

JSON format supports the following data types –

Type	Description
Number	double precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
Null	empty

Number

- It is a double precision floating-point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number.

The following table shows the number types –

Type	Description
Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like .3, .9
Exponent	Exponent like e, e+, e-, E, E+, E-

Syntax: var json-object-name = { string : number_value, ...}

Example : for Number datatype, value should not be quoted –
var obj = {marks: 97}

String

- It is a sequence of zero or more double quoted Unicode characters with backslash escaping.

- Character is a single character string i.e. a string with length 1.

The table shows String types –

Type	Description	Type	Description
"	double quotation	\	reverse solidus
B	backspace	/	Solidus
F	form feed	N	new line
R	carriage return	T	horizontal tab

Syntax : var json-object-name = { string : "string value", ...}

Example : for String Datatype the value should be double quoted-
var obj = {name: 'Amit'}

Boolean
It includes true or false values.
Syntax : var json-object-name = { string : true/false, ...}
Example :
var obj = {name: 'Amit', marks: 97, distinction: true}

Array

- It is an ordered collection of values.
- These are enclosed in square brackets which means that array begins with .[. and ends with .]..
- The values are separated by , (comma).
- Array indexing can be started at 0 or 1.
- Arrays should be used when the key names are sequential integers.

Syntax : [value, ...]

Example : Array containing multiple objects –
```
{   "books": [
      { "language":"Java" , "edition":"second" },
      { "language":"C++" , "lastName":"fifth" },
      { "language":"C" , "lastName":"third" }
   ]
}
```

Object

- It is an unordered set of name/value pairs.
- Objects are enclosed in curly braces that is, it starts with '{' and ends with '}'.
- Each name is followed by ':'(colon) and the name/value pairs are separated by , (comma).
- The keys must be strings and should be different from each other.
- Objects should be used when the key names are arbitrary strings.

Syntax : { string : value, ...}

Example : Object –
```
{
   "id": "011A",
   "language": "JAVA",
   "price": 500,
}
```
Whitespace
It can be inserted between any pair of tokens. It can be added to make a code more readable. Example shows declaration with and without whitespace –
Syntax : {string:" ",....}

Example
```
var i = " sachin";
var j = " saurav"
```

null : It means empty type.
Syntax : null
Example
```
var i = null;
if(i == 1){
  document.write("<h1>value is 1</h1>");
}
else{
  document.write("<h1>value is null</h1>");
}
```

JSON Value

We can declare a variable in JSON by var keyword , variable name & its value.
```
 var  x = value.
```

JSON supports values of following types

number (integer or floating point)	boolean	object
String	array	null

Example
```
var  i = 1;
var  j = "sachin";
var  k = null;
```

JSON - Objects

JSON objects can be created in various ways –
Using Javascript to create object :

- Creation of an empty Object –

```
var JSONObj = {};
```

- Creation of a new Object –

```
var JSONObj = new Object();
```

- Creation of an object with attribute bookname with value in string, attribute price with numeric value. Attribute is accessed by using '.' Operator –

```
var    JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
```

How to create an object in javascript using JSON ?
Save the below code as json_object_javascript.htm
```
<html>
  <head>
    <title>Creating Object JSON with JavaScript</title>
    <script language = "javascript" >
      var JSONObj = { "name" : "www.oracle.com", "year"  : 1994 };
      document.write("<h1>JSON with JavaScript example</h1>");
      document.write("<br>");
      document.write("<h3>Website Name = "+JSONObj.name+"</h3>");
      document.write("<h3>Year = "+JSONObj.year+"</h3>");
    </script>
  </head>   <body>   </body> </html>
```

Creating Array Objects :

```
<html>
  <head>
    <title>Creation of array object in javascript using JSON</title>
    <script language = "javascript" >
      document.writeln("<h2>JSON array object</h2>");
      var books = { "Pascal" : [
        { "Name"  : "Pascal Made Simple", "price" : 700 },
        { "Name"  : "Guide to Pascal", "price" : 400 } ],

        "Scala"  : [
          { "Name"  : "Scala for the Impatient", "price" : 1000 },
          { "Name"  : "Scala in Depth", "price" : 1300 }]
      }

      var i = 0
      document.writeln("<table border = '2'><tr>");
      for(i = 0;i<books.Pascal.length;i++){
        document.writeln("<td>");
        document.writeln("<table border = '1' width = 100 >");
        document.writeln("<tr><td><b>Name</b></td><td width = 50>" + books.Pascal[i].Name
+ "</td></tr>");
        document.writeln("<tr><td><b>Price</b></td><td width = 50>" + books.Pascal[i].price
+"</td></tr>");
        document.writeln("</table>");
        document.writeln("</td>");
      }
      for(i = 0;i<books.Scala.length;i++){
        document.writeln("<td>");
        document.writeln("<table border = '1' width = 100 >");
        document.writeln("<tr><td><b>Name</b></td><td width = 50>" +
books.Scala[i].Name+"</td></tr>");
        document.writeln("<tr><td><b>Price</b></td><td width = 50>" +
books.Scala[i].price+"</td></tr>");
        document.writeln("</table>");
        document.writeln("</td>");
      }
      document.writeln("</tr></table>");
    </script>
  </head>
  <body>
  </body>
</html>
```

JSON - Schema

JSON Schema is a specification for JSON based format for defining the structure of JSON data. It was written under IETF draft which expired in 2011.

JSON Schema –

- Describes your existing data format.
- Clear, human- and machine-readable documentation.
- Complete structural validation, useful for automated testing.
- Complete structural validation, validating client-submitted data.

JSON Schema Validation Libraries

Different validators are currently available for different programming languages. Currently the most complete and compliant JSON Schema validator available is JSV.

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

JSON Schema Example

Given below is a basic JSON schema, which covers a classical product catalog description –

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",

  "properties": {

    "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },

    "name": {
      "description": "Name of the product",
      "type": "string"
    },

    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
  },

  "required": ["id", "name", "price"]
}
```

The important keywords that can be used in this schema is given below–

Keywords	**Description**
$schema	The $schema keyword states that this schema is written according to the draft v4 specification.
Title	You will use this to give a title to your schema.
description	A little description of the schema.
Type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.

minimum	This is the constraint to be put on the value and represents minimum acceptable value
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
maximum	This is the constraint to be put on the value and represents maximum acceptable value.
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	The length of a string instance is defined as the maximum number of its characters.
minLength	The length of a string instance is defined as the minimum number of its characters.
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

You can check a http://json-schema.org for the complete list of keywords that can be used in defining a JSON schema. The above schema can be used to test the validity of the following JSON code.1

```
[
  {
    "id": 2,
    "name": "An ice sculpture",
    "price": 12.50,
  },

  {
    "id": 3,
    "name": "A blue mouse",
    "price": 25.50,
  }
]
```

JSON - Comparison with XML

JSON and XML are human readable formats and are language independent. They both have support for creation, reading and decoding in real world situations. We can compare JSON with XML, based on the following factors −

Verbose - XML is more verbose than JSON, so it is faster to write JSON for programmers.

Arrays Usage - XML is used to describe the structured data, which doesn't include arrays whereas JSON include arrays.

Parsing - JavaScript's eval method parses JSON. When applied to JSON, eval returns the described object.

Example

JSON

```
{
  "company": Volkswagen,
  "name": "Vento",
  "price": 800000
}
```

XML

```
<car>
  <company>Volkswagen</company>
  <name>Vento</name>
  <price>800000</price>
</car>
```

JSON with Java

Before you start with encoding and decoding JSON using Java, you need to install any of the JSON modules available. For this tutorial we have downloaded and install json.simple jar file and have added the location of json-simple-1.1.1.jar file to the environment variable CLASSPATH.

Mapping between JSON and Java entities

JSON.simple jar file maps entities from the left side to the right side while decoding or parsing, and maps entities from the right to the left while encoding.

JSON values	Java Class types
string	java.lang.String class
number	java.lang.Number class
true|false	java.lang.Boolean class
null	null
array	java.util.List class
object	java.util.Map class

On decoding, the default concrete class of java.util.List becomes org.json.simple.JSONArray and the default concrete class of java.util.Map becomes org.json.simple.JSONObject.

Encoding JSON in Java

To encode a JSON object using Java JSONObject in java we need a subclass of java.util.HashMap. No ordering is provided in HashMap. If you need the strict ordering of elements, use JSONValue.toJSONString (map) method with ordered map implementation such as java.util.LinkedHashMap.

```
import org.json.simple.JSONObject;
class JsonEncodeDemo {
  public static void main(String[] args){
    JSONObject obj = new JSONObject();
    obj.put("name", "foo");
    obj.put("num", new Integer(100));
    obj.put("balance", new Double(1000.21));
    obj.put("is_vip", new Boolean(true));

    System.out.print(obj);
  }
}
```

On compiling and executing the above program the following result will be generated –

```
{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}
```

Example that shows a JSON object streaming using Java JSONObject –

```
import org.json.simple.JSONObject;
class JsonEncodeDemo {
  public static void main(String[] args){
    JSONObject obj = new JSONObject();
    obj.put("name","foo");
    obj.put("num",new Integer(100));
    obj.put("balance",new Double(1000.21));
    obj.put("is_vip",new Boolean(true));

    StringWriter sw = new StringWriter();
    obj.writeJSONString(sw);

    String  jsonText = sw.toString();
    System.out.print(jsonText);
  }
}
```

On compiling and executing the above program, the following result is generated −

```
{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}
```

Decoding JSON in Java

Let us make use of JSONObject andJSONArray where JSONObject is a java.util.Map and JSONArray is a java.util.List, so that we can access them with standard operations of Map or List.

```
import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.parser.ParseException;
import org.json.simple.parser.JSONParser;

class JsonDecodeDemo {
  public static void main(String[] args){
    JSONParser parser = new JSONParser();
    String s = "[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}]";

    try{
      Object obj = parser.parse(s);
      JSONArray array = (JSONArray)obj;

      System.out.println("The 2nd element of array");
      System.out.println(array.get(1));
      System.out.println();

      JSONObject obj2 = (JSONObject)array.get(1);
      System.out.println("Field \"1\"");
      System.out.println(obj2.get("1"));

      s = "{}";
      obj = parser.parse(s);
      System.out.println(obj);

      s = "[5,]";
      obj = parser.parse(s);
      System.out.println(obj);

      s = "[5,,2]";
      obj = parser.parse(s);
      System.out.println(obj);
    }catch(ParseException pe){

      System.out.println("position: " + pe.getPosition());
      System.out.println(pe);
      }
   }
}
```

On compiling and executing the above program, the following result will be generated –

```
The 2nd element of array
{"1":{"2":{"3":{"4":[5,{"6":7}]}}}}

Field "1"
{"2":{"3":{"4":[5,{"6":7}]}}}
{}
[5]
[5,2]
```

JSON with Ajax

Developers uses JSON to pass AJAX updates between the client and the server. Websites updating live sports scores can be considered as an example of AJAX. If these scores have to be updated on the website, then they must be stored on the server so that the webpage can retrieve the score when it is required. This is where we can make use of JSON formatted data.

Any data that is updated using AJAX can be stored using the JSON format on the web server. AJAX is used so that javascript can retrieve these JSON files when necessary, parse them, and perform one of the following operations –

- Store the parsed values in the variables for further processing before displaying them on the webpage.
- It directly assigns the data to the DOM elements in the webpage, so that they are displayed on the website.

Example
Strore the following code as test.html under ...\webapps\jsons\
Here the loading function loadJSON() is used asynchronously to upload JSON data.

```
<html>
   <head>
      <meta content = "text/html; charset = ISO-8859-1" http-equiv = "content-type">
      <script type = "application/javascript">
         function loadJSON(){
            var data_file = "data.json";
            var http_request = new XMLHttpRequest();
            try{
               // Opera 8.0+, Firefox, Chrome, Safari
               http_request = new XMLHttpRequest();
            }catch (e){
               // Internet Explorer Browsers
               try{
                  http_request = new ActiveXObject("Msxml2.XMLHTTP");
               }catch (e) {

                  try{
                     http_request = new ActiveXObject("Microsoft.XMLHTTP");
                  }catch (e){
                     // Something went wrong
                     alert("Your browser broke!");
                     return false;
                  }
```

```
          }
        }

        http_request.onreadystatechange = function(){
          if (http_request.readyState == 4  ){
            // Javascript function JSON.parse to parse JSON data
            var jsonObj = JSON.parse(http_request.responseText);

            // jsonObj variable now contains the data structure and can be accessed as jsonObj.name
and jsonObj.country.
            document.getElementById("Name").innerHTML =  jsonObj.name;
            document.getElementById("Country").innerHTML =  jsonObj.country;
          }
        }

        http_request.open("GET", data_file, true);
        http_request.send();
      }
    </script>
    <title>JSON</title>
  </head>

  <body>
    <h1>Cricketer Details</h1>
    <table class = "src">
      <tr><th>Name</th><th>Country</th></tr>
      <tr><td><div id = "Name">Baga</div></td>
      <td><div id = "Country">Nepal</div></td></tr>

    </table>

    <div class = "central">
      <button type = "button" onclick = "loadJSON()">Update Details </button>
    </div>
  </body>
</html>
```

The input file data.json is kept in jsons folder with a name data.json

```
{"name": "Sachin", "country": "India"}
```

The above HTML code will generate the following screen, where you can check AJAX in action –

```
Cricketer Details
Name   Country
Baga   Nepal
```

When we click on update Detail button we will get :

```
Cricketer Details
Name   Country
Sachin India
```