JSON Best Practices

Ali-Akbor

**JavaScript Object Notation (JSON)**

* JSON is a lightweight data-interchange format that is completely language independent.
* It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data
* The official Internet media type for JSON is application/json.
* It was designed for human-readable data interchange.
* The filename extension is .json.

**What JSON looks like**

We will now look how a typical JSON looks like. The following code snippet is a valid (you will see in a later chapter what is syntactically valid JSON) JSON representing

{

"Title": "The Cuckoo's Calling",

"Author": "Robert Galbraith",

"Genre": "classic crime novel",

"Detail": {

"Publisher": "Little Brown",

"Publication\_Year": 2013,

"ISBN-13": 9781408704004,

"Language": "English",

"Pages": 494

},

"Price": [

{

"type": "Hardcover",

"price": 16.65

},

{

"type": "Kidle Edition",

"price": 7.03

}

]

}

### **JSON Syntax**

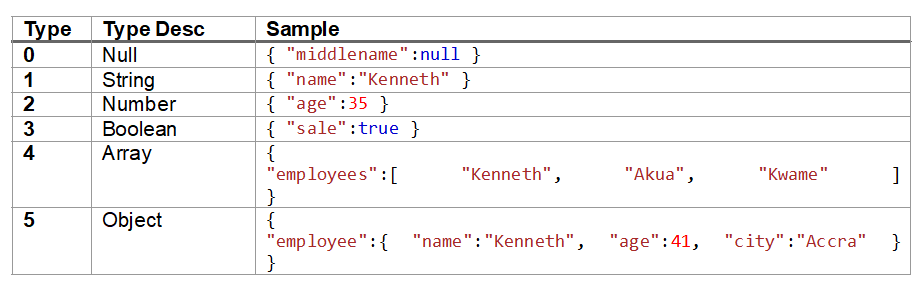
**Always enclose the key, value pair within double quotes.** Most JSON parsers don’t like to parse JSON objects with single quotes.

**{** "name": "Katherine Johnson" **}**

**Never use hyphens in your key fields.** Use underscores  ( \_ ), all lower case, or camel case.

**{** "first\_name":"Katherine"**,** "last\_name":"Johnson" **}**

**JSON - Data Type**



**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.

**Characteristics of JSON**

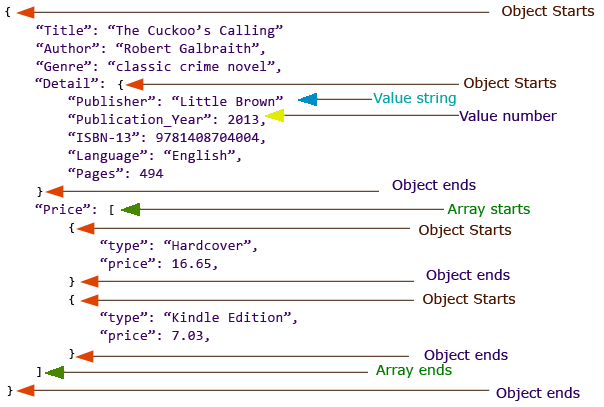
* JSON is easy to read and write.
* It is a lightweight text-based interchange format.
* JSON is language independent.

## Basic Constructs

* There four basic and built-in data types in JSON. They are strings, numbers, booleans (i.e true and false) and null. Besides, there are two data types which are structured - objects and arrays.
* Objects are wrapped within '{' and '}'. Arrays are enclosed by '[' and ']'. Objects are a list of label-value pairs. Arrays are list of values.
* Both objects and arrays can be nested.
* strings, numbers, booleans (i.e true and false) and null can be used as values.

**Understanding JSON Structure**

The following image and then text following will be useful to get you started with how JSON data is constructed.



**Always create a Root element.**

Creation of Root element is optional, but it helps when you are generating complicated JSON.

**JSON with root element**{  
"menu": [  
 {  
 "id": "1",  
 "name":"File",  
 "value": "F",  
 "popup": {  
 "menuitem": [  
 {

"name":"New",

"value": "1N",

"onclick":"newDoc()"

},  
 {

"name":"Open",

"value":"1O",

"onclick":"openDoc()"

},  
 {

"name":"Close",

"value":"1C",

"onclick":"closeDoc()"

}  
 ]  
 }  
 },  
 {  
 "id": "2",  
 "name":"Edit",  
 "value": "E",  
 "popup": {  
 "menuitem": [  
 {

"name":"Undo",

"value": "2U",

"onclick":"undo()"

},  
 {

"name":"Copy",

"value": "2C",

"onclick":"copy()"

},

{

"name":"Cut",

"value": "2T",

"onclick": "cut()"

}  
 ]  
 }  
 }  
 ]  
}

**JSON without root element**[  
 {  
 "id": "1",  
 "name":"File",  
 "value": "F",  
 "popup": {  
 "menuitem": [  
 {

"name":"New",

"value": "1N",

"onclick": "newDoc()"

},  
 {

"name":"Open",

"value": "1O",

"onclick": "openDoc()"

},  
 {

"name":"Close",

"value": "1C",

"onclick": "closeDoc()"

}  
 ]  
 }  
 },  
 {  
 "id": "2",  
 "name":"Edit",  
 "value": "E",  
 "popup": {  
 "menuitem": [  
 {

"name":"Undo",

"value": "2U",

"onclick": "undo()"

},

{

"name":"Copy",

"value": "2C",

"onclick": "copy()"

},  
 {

"name":"Cut",

"value": "2T",

"onclick": "cut()"

}  
 ]  
 }  
   
   
 }  
]

## Comparison with Relational Database

Since JSON is used to host/represent data, we will discuss how it is different from the traditional Relational Database model used in RDBMS systems like MySQL, SQL Server etc. This may be useful for you to choose JSON over RDBMS or RDBMS over JSON depending upon the type and structure of data you want to deal with. Let's start with a comparison against certain features:

* **Structure** : In the relational database, these are tables, which are responsible for storing data in form of rows and columns. JSON uses objects and arrays - objects are label-value pairs and arrays are the list of values. They can be nested recursively.
* **Metadata** : In a relational database, it is a schema, which is used for storing data about the structure and type of the data to be stored and schemas are predefined, i.e. they are created at the time of creation of database and tables before you can store data. JSON also may use schema, to have a definition of the structure and type of data to represented, but it is not predefined. Most of the time it is self-describing, even if it uses a schema, it comes with much more flexibility than a schema used in relational database. But it would be judgmental to say that it is an advantage of JSON over Relational Database. Having a pre-defined schema may have several benefits depending upon the data to be dealt with.
* **Retrieving data** : Relational databases use Structured Query Language, an expressive and very powerful language, based on relational algebra to fetch data from the database. JSON does not have any widely used or accepted language to query the data stored. JAQL and JSONiq are many of the query languages which mostly are work in progress to query data from JSON.
* **Sorting** : SQL does the job in case of Relational Database. In the case of JSON, since arrays often used, in programs, arrays can be sorted.
* **Application** : There are many open-source as well as commercial Relational Database systems are available - like MySQL, POstgreSQL, SQL Server, Oracle, DB2 etc. JSON is mostly applied to programming languages. But, there is also NoSQL systems. NoSQL systems use JSON format to store data. Some of the NoSQL systems use JSON format are - MongoDB, CoucheDB etc.
* **Learning curve**: JSON is a clear winner here. Since the basic data types and structure used here are similar to those used in many programming languages, it is obvious that if you are coming from a programming background, you will pick things up in JSON pretty fast. RDBMS is a separate field of study on the other hand. But definitely, the time you invest in learning Relational database return you several opportunities and benefits.

## JSON vs XML

* XML is more expressive than JSON. XML sometimes also suffers from using tags repeatedly, where as JSON is much more concise.
* XML is more complex than JSON.
* There are several specifications to define schema(metadata) for XML, for example DTD and XSD. JSON schema is there for doing the same for JSON, but it is not as widely used as XML schemas.
* XML can be used with most of the programming languages as JSON. But the point is, when you are working with XML, then you have you are actually trying match two systems those data structures are different. In the case of JSON though, since objects and arrays are basic data structures used, it is easy to work with them in programs.
* For selecting specific parts of an XML document, there is standard specification called XPath. This is widely used. In JSON, we have JSONPath to do the same, but not widely in use.
* XML has Xquery specification for querying XML data. JSON though have JAQL, JSONiq etc, but they are not in use widely.
* XML has XSLT specification which may be used to apply a style to an XML document. JSON does not have any such thing.
* <?xml version="1.0" encoding="UTF-8" ?>
* **<inventors>**
* **<inventor>**
* **<name>**Katherine Johnson**</name>**
* **<age>**101**</age>**
* **<city>**Newport News**</city>**
* **</inventor>**
* **<inventor>**
* **<name>**Dorothy Vaughan**</name>**
* **<age>**98**</age>**
* **<city>**Hampton**</city>**
* **</inventor>**
* **<inventor>**
* **<name>**Henry Ford**</name>**
* **<age>**83**</age>**
* **<city>**Detroit**</city>**
* **</inventor>**
* **</inventors>**