Introduction to JSON

Roger Lacroix
roger.lacroix@capitalware.com
http://www.capitalware.com

What is JSON?

- JSON: ***J***ava***S***cript ***O***bject ***N***otation.
- JSON is a simple, text-based way to store and transmit structured data. By using a simple syntax, the user can easily store anything from a single number through to strings, arrays, and objects using nothing but a string of plain text. You can also nest arrays and objects, allowing you to create complex data structures.

What is JSON?

- JSON is a syntax for storing and exchanging data.
- JSON is text (no binary data).
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent.

Why use JSON?

- Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.
- For IBM MQ, setting the message's Format attribute to 'MQSTR' (String) then when the receiving application performs an 'MQGET with Convert', MQ will convert the text from one CCSID to another CCSID. i.e. ASCII to EBCDIC

JSON is NOT

- Overly Complex
- A 'document' format
- A markup language
- A programming language

JSON Syntax

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects i.e. { }
- Square brackets hold arrays i.e. [ ]

JSON Object Syntax

- Unordered sets of name/value pairs
- Starts with **{** (left brace)
- Ends with **}** (right brace)
- Each name is followed by **:** (colon) and then its value
- Name/value pairs are separated by **,** (comma)

JSON Data - A Name and a Value

- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:
- Example

```
{ "name" : "John" }
```

JSON Arrays

- An ordered collection of values
- Starts with [(left bracket)
- Ends with] (right bracket)
- Name/value pairs are separated by , (comma)

JSON Array Example

```
{
   "name" : "John",
   "cars" : [  "Ford", "BMW", "Fiat" ]
 }
```

JSON Data Types

JSON Values

- In JSON, values must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - null
- JSON values ***cannot*** be one of the following data types:
  - a function
  - a date
  - undefined

JSON Strings

- Strings in JSON must be written in double quotes.
- Example

```
{ "name" : "John" }
```

JSON Numbers

- Numbers in JSON must be an integer or a floating point.
- Example

```
{ "age" : 30 }
```

JSON Objects

- Values in JSON can be objects.
- Example

```
{
"employee" : {  "name" : "John", "age" : 30, "city" : "New York" }
}
```

JSON Arrays

- Values in JSON can be arrays.
- Example

```
{
"employees" : [  "John", "Anna", "Peter" ]
}
```

JSON Booleans

- Values in JSON can be true or false.
- Example

```
{ "sale" : true }
```

JSON null

- Values in JSON can be null.
- Example

```
{ "middlename" : null }
```

Nested JSON Objects

- Values in a JSON object can be another JSON object.
- Example

```
myObj = {
                "name":"John",
                "age":30,
                "cars" : {
                            "car1" : "Ford",
                            "car2" : "BMW",
                            "car3" : "Fiat"
                         }
 }
x = myObj.cars["cars2"];
```

Arrays in JSON Objects

- Arrays can be values of an object property:
- Example

```
myObj = {
            "name" : "John",
            "age" : 30,
           "cars" : [  "Ford", "BMW", "Fiat" ]
}
x = myObj.cars[0];
```

Nested Arrays in JSON Objects

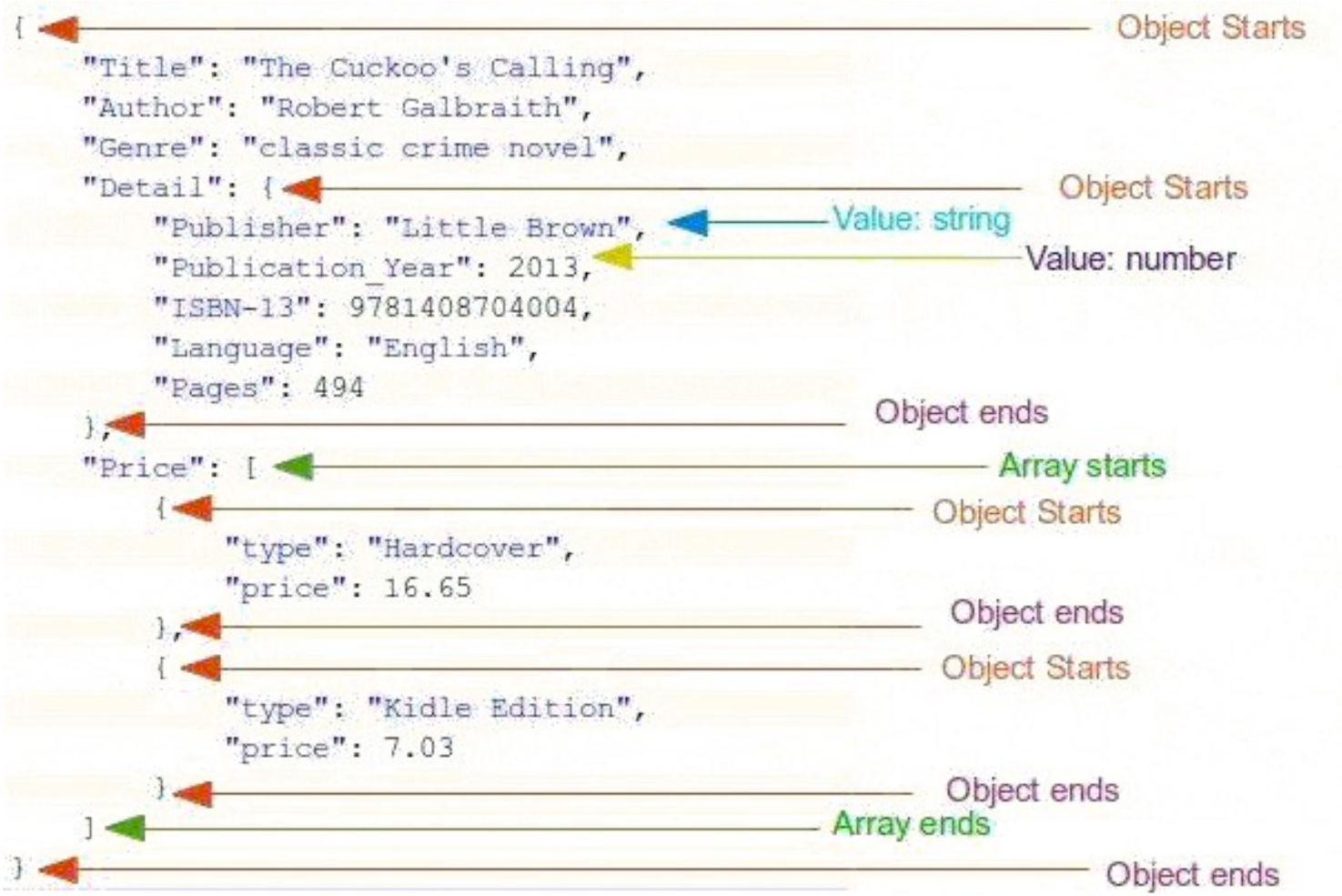
- Values in an array can also be another array, or even another JSON object:
- Example

```
myObj = {
            "name" : "John",
            "age" : 30,
            "cars" : [
                        { "name" : "Ford",  "models":[ "Fiesta", "Focus", "Mustang" ] },
                        { "name" : "BMW", "models" : [ "320", "X3", "X5" ] },
                        { "name" : "Fiat", "models" : [ "500", "Panda" ] }
           ]
}
```

- To access arrays inside arrays, use a for-in loop for each array:

```
for (i in myObj.cars) {
    x += "<h1>" + myObj.cars[i].name  + "</h1>";
    for (j in myObj.cars[i].models) {
        x += myObj.cars[i].models[j];
    }
}
```

JSON Data Example



JSON vs XML

- Both JSON and XML can be used as a data format when sending/receiving data between servers.
- The following JSON and XML examples both defines an employees object, with an array of 3 employees:

JSON vs XML

```
{"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 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

JSON vs XML

- 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

JSON vs XML

- Since XML is widely used as data interchange format, we will try to draw a comparison between them. The purpose of the comparison is not to determine which is better but rather we will try to understand which one is suitable for storing specific kind of data.

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.
- JSON lacks namespaces
- There are several specifications to define schema(metadata) for XML, for example DTD and XSD. JSON schema is available but is not as widely used as XML schemas.

JSON vs XML

- XML and JSON can be used with most of the programming languages. The problem comes when matching XML tags used by one system may be different in another system, hence, XML data will require transformation. i.e. <FirstName> vs <first_name>. In the case of JSON, 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 it is not widely used.
- XML has Xquery specification for querying XML data. JSON does 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.

JSON vs XML

- Favor XML over JSON when any of these is true:
  - You need message validation
  - You're using XSLT
  - Your messages include a lot of marked-up text
  - You need to inter-operate with environments that don't support JSON
- Favor JSON over XML when all of these are true:
  - Messages don't need to be validated
  - You're not transforming messages
  - Your messages are mostly data, not marked-up text
  - The messaging endpoints have good JSON tools

Java and JSON

- JEE (Java Enterprise Edition) includes a JSON parser.

https://docs.oracle.com/javaee/7/tutorial/jsonp001.htm

- JSE (Java Standard Edition) does **not** include a JSON parser.
- I suggest you use the Google-Gson parser for JSE projects.

https://github.com/google/gson

https://en.wikipedia.org/wiki/Gson

https://github.com/google/gson/blob/master/UserGuide.md

Questions & Answers