XML and Databases

Chapter 16: JSON

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Winter 2022/23

http://www.informatik.uni-halle.de/~brass/xml22/

Objectives

After completing this chapter, you should be able to:

- compare JSON with XML.
- write syntactically correct JSON documents.
- explain some features of XQuery 3.1 that were introduced to support JSON.

Contents

Introduction (1)

- JSON ("JavaScript Object Notation") is a standard file format for data interchange like XML.

  JSON is pronounced like "Jason" or like "JAY-sawn".

  The story of JSON is explained by its inventor, Douglas Crockford, in:

  [https://www.youtube.com/watch?v=-C-JoyNuQJs]

  He says, he did not invent it, but discovered it, because it is valid JavaScript.

  The first JSON message was sent in 2001.

- It was developed for data exchange between applications running in the browser and the server.

  In the meantime, it is used also for many other applications.

- It is tree-structured just as XML, and one can translate data in both directions quite easily.

Introduction (2)

- The current version of the JSON standard is from 2017:
  - ecma Standard ECMA-404:
    The JSON Data Interchange Syntax.
    [https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf

    The standard has 16 pages, but the real text of the standard is on 5 pages. The first edition is from October 2013:
    [https://.../uploads/ECMA-404_1st_edition_october_2013.pdf]
- The syntax diagrams are also on:
  - [https://www.json.org/json-en.html] (English)
  - [https://www.json.org/json-de.html] (German)
- This page is nicer than the standard and contains also links to implementations for many different programing languages.

Introduction (3)

JSON vs. XML:

- JSON is a bit shorter, it does not need an end tag.

  If one uses empty elements with data in attributes in XML, the advantage of JSON is much smaller, or XML might even be slightly shorter.

- For the programming language JavaScript, JSON is simpler to use than XML.

  For most other programming languages, both need a parser library.

- The JSON standard is shorter than the XML standard.

  The XML standard has 59 pages (including front matter and appendices).

  It contains DTDs, Entities, CDATA sections, comments, encoding specifications — all of which does not exist in JSON.

- JSON might be more attractive to programmers, because it uses known concepts like objects (tuples) and arrays.

Introduction (4)

- There are more alternatives, especially for configuration files.
  - For instance, `YAML` is quite well known.

    [https://en.wikipedia.org/wiki/YAML]
    [https://yaml.org/]
    [https://yaml.org/spec/1.2.2/]
  - TOML, HOCON, ...
- The NoSQL-database MongoDB stores in principle JSON documents, but it uses a binary format (BSON: "Binary JSON") and some extensions to JSON.

  E.g., it has a data type for date values which is missing in JSON. MongoDB and BSON also support different numeric types (such as integer) in addition the the standard floating point number of JSON. [https://bsonspec.org/spec.html]

Contents

Example (1)

- The student grades database in JSON.

  [https://users.informatik.uni-halle.de/˜brass/xml22/examples/ex1.json]

- On the top level, the document is an object `{...}` with three attributes/properties `"STUDENTS"`, `"EXERCISES"`, and `"RESULTS"`.

- The value of each property is an array `[...]` with objects corresponding to the table rows:

```
{ "STUDENTS": [
      {"SID": 101,
       "FIRST": "Ann",
       "LAST": "Smith",
       "EMAIL": "smith@acm.org"},
```

Example (2)

- Part 2/6 (remaining students, end of array):

```
{"SID": 102,
 "FIRST": "Michael",
 "LAST": "Jones"},
{"SID": 103,
 "FIRST": "Richard",
 "LAST": "Turner",
 "EMAIL": "richard.turner@gmx.de"},
{"SID": 104,
 "FIRST": "Maria",
 "LAST": "Brown",
 "EMAIL": "brown@hotmail.com"}
],
```

Example (3)

- Part 3/6 (exercise list):

```
"EXERCISES": [
  {"CAT": "H",
   "ENO": 1,
   "TOPIC": "Relational Algebra",
   "MAXPT": 10},
  {"CAT": "H",
   "ENO": 2,
   "TOPIC": "SQL",
   "MAXPT": 10},
  {"CAT": "M",
   "ENO": 1,
   "TOPIC": "SQL",
   "MAXPT": 14}
  ],
```

Example (4)

- Part 4/6 (results for exercises submitted by Ann Smith):

```
"RESULTS": [
  {"SID": 101,
   "CAT": "H",
   "ENO": 1,
   "POINTS": 10},
  {"SID": 101,
   "CAT": "H",
   "ENO": 2,
   "POINTS": 8},
  {"SID": 101,
   "CAT": "M",
   "ENO": 1,
   "POINTS": 12},
```

Example (5)

- Part 5/6 (list of results submitted by Michael Jones):

```
{"SID": 102,
 "CAT": "H",
 "ENO": 1,
 "POINTS": 9},
{"SID": 102,
 "CAT": "H",
 "ENO": 2,
 "POINTS": 9},
{"SID": 102,
 "CAT": "M",
 "ENO": 1,
 "POINTS": 10},
```

Example (6)

- Part 6/6 (list of results submitted by Richard Turner):

```
                {"SID": 103,
                 "CAT": "H",
                 "ENO": 1,
                 "POINTS": 5},
                {"SID": 103,
                 "CAT": "M",
                 "ENO": 1,
                 "POINTS": 7}
                ]
            }
```

These are 66 lines and 1019 characters. The XML version with data in attributes [ex1.xml] has 23 lines and 917 characters. The XML version with data as element content [ex2.xml] has 101 lines and 1698 characters.

Example (7)

- Formatting after “pretty printing” with https://jsonlint.com/:

```
{
    "STUDENTS": [{
            "SID": 101,
            "FIRST": "Ann",
            "LAST": "Smith",
            "EMAIL": "smith@acm.org"
        },
        {
            "SID": 102,
            "FIRST": "Michael",
            "LAST": "Jones"
        },
        ...
    ],
```

Example (8)

- This is a direct translation of the relational database structure:
  - The database is an object consisting of relations.
  - The relations are represented as arrays of tuples.
  - The tuples are objects containing the data values.

    With the columns as names of the properties/attributes.
- Of course, as in XML, other nestings are possible:
  - One could nest an array with the results for a particular student as a property of the student object.
  - One could nest an array with the results for a particular exercise as a property of the exercise object.

JSON Syntax (1)

- JSON has only six data types:
  - Strings (in double quotes): `"abc"`
  - Numbers (floating point numbers): `-1.23e4`
  - Boolean values: `true` and `false`
  - The null value: `null`
  - Objects (containing name/value-pairs): `{"a":1, "b":2}`
  - Arrays: `[1, 2, 3]`
- A JSON value is a value of one of these six data types.

  Often, a JSON document contains an object on the top level, but this is not required, any JSON value (with optional whitespace before and after it) can be a JSON document. The old RFC 4627 required an object or array on top level, and there are security concerns with arrays (leaving only objects).

JSON Syntax (2)

- Strings are enclosed in double quotes: `"..."`.

  Double quotes inside the string must be escaped as e.g. in Java: `\"`.

  Control characters are excluded. Thus strings must end on the same line.

- The following escape sequences are understood:
  - `\"`: Quotation mark.
  - `\\`: Backslash (“reverse solidus”).
  - `\/`: Slash (“solidus”).
  - `\n`: Linefeed.
  - `\r`: Carriage return.
  - `\t`: Tabulator (“horizontal tab”).
  - `\b`: Backspace.
  - `\f`: Formfeed.
  - `\u`$h_1h_2h_3h_4$: UTF-16 unicode value (four hexadecimal digits).

JSON Syntax (3)

- The escape sequence `\/` can help to mask HTML end tags like `</script>` in string constants.

  The browser might assume that a quote was missing. C and Java have no \/.

- White space (space, tab, linefeed, carriage return) is allowed between each two tokens.

- The names in name/value pairs (inside objects `{...}`) must be written as string constants.

  I.e. the `"..."` are required. (In JavaScript, the quotes are not required, but only if the name is no reserved word. By requiring the quotes, the JSON specification does not need to include a list of reserved words or explain what are valid letters in Unicode to be used in identifiers. The name can be any string, including characters that are not letters or digits.)

JSON Syntax (4)

- Empty Arrays and empty objects are possible.
- A comma after the last array element is forbidden.

  C and Java would allow this, because an automatic generation does not need to treat the last element specially, and the later addition of further elements is simplified.
- An arbitrary nesting is possible, e.g. one can also nest objects in objects.

  The values in name/value-pairs can be of any of the six data types. The same is true for array elements.
- One syntax check for JSON documents is:

  [https://jsonlint.com/]

Contents

XQuery 3.1 Support for JSON

-

References

- [https://www.json.org/json-en.html]
  [https://www.json.org/json-de.html]
- ecma Standard ECMA-404: The JSON Data Interchange Syntax.
  [https://www.ecma-international.org/wp-content/uploads/
  ECMA-404_2nd_edition_december_2017.pdf
- SELFHTML: JSON
  [https://wiki.selfhtml.org/wiki/JSON]
- [https://www.w3schools.com/js/js_json_intro.asp]
- [https://en.wikipedia.org/wiki/JSON]
  [https://de.wikipedia.org/wiki/JavaScript_Object_Notation]
- Douglas Crockford: The JSON Saga.
  [https://www.youtube.com/watch?v=-C-JoyNuQJs]
- Seva Safris: A Deep Look at JSON vs. XML, Part 1: The History of Each Standard.
  [https://www.toptal.com/web/json-vs-xml-part-1]
  Part 2: The Strengths and Weaknesses of Both.
  [https://www.toptal.com/web/json-vs-xml-part-2]
- You've been Haacked: JSON Hijacking.
  [https://haacked.com/archive/2009/06/25/json-hijacking.aspx/]