Xquery 3.1
Json

Arantza Irastorza Goñi

Informazioaren Kudeaketa Aurreratua

Gradua Ingeniaritza Informatikoan

Esp. Software Ingeniaritza

Lengoaiak eta Sistema Informatikoak saila

2017 urtarrila

eman ta zabal zazu

informatika fakultatea

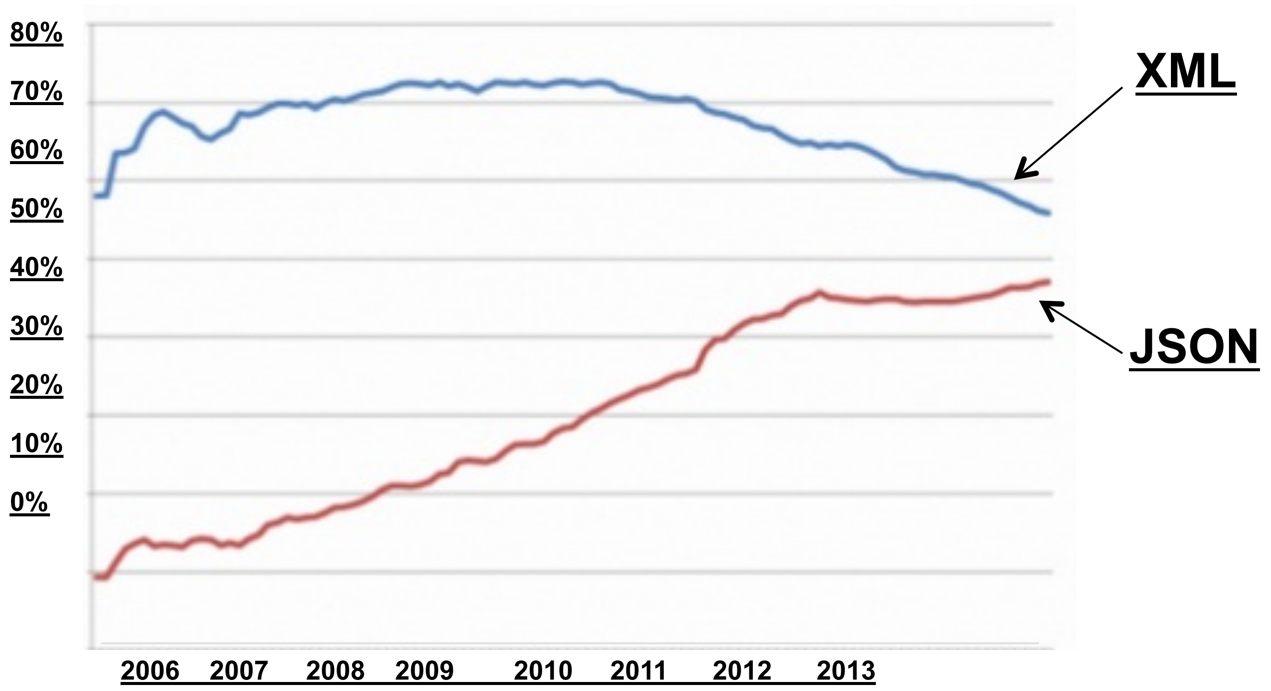
facultad de informática

Contents

- JSON
  - XML vs. JSON
- Xquery 3.1
  - New in the data model: **map** and **array**
  - New functions: ***json-doc()*** and ***parse-json()***

    https://www.w3.org/TR/xpath-functions-31/

Trends in XML and JSON usage



Based on directory of 11,000 web APIs listed at Programmable Web, December 2013

JSON: JavaScript Object Notation

- A lightweight data-interchange format
  - Easy for humans to read and write
  - Easy for machines to parse and generate

```
<Book>
   <Title>Parsing Techniques</Title>
   <Authors>
      <Author>Dick Grune</Author>
      <Author>Ceriel J.H. Jacobs</
Author>
   </Authors>
   <Date>2007</Date>
   <Publisher>Springer</Publisher>
</Book>
```

```
{
  "Book":
    {
      "Title": "Parsing Techniques",
      "Authors":
               [ "Dick Grune",
                 "Ceriel J.H. Jacobs" ],
      "Date": "2007",
      "Publisher": "Springer"
    }
}
```

XML to JSON

- Threre are web converters

  http://www.freeformatter.com/xml-to-json-converter.html

```
<Book id="MCD">
    <Title>Modern Compiler Design</Title>
    <Author>Dick Grune</Author>
    <Publisher>Springer</Publisher>
</Book>
```

```
{
   "@id": "MCD",
   "Title": "Modern Compiler Design",
   "Author": "Dick Grune",
   "Publisher": "Springer"
}
```

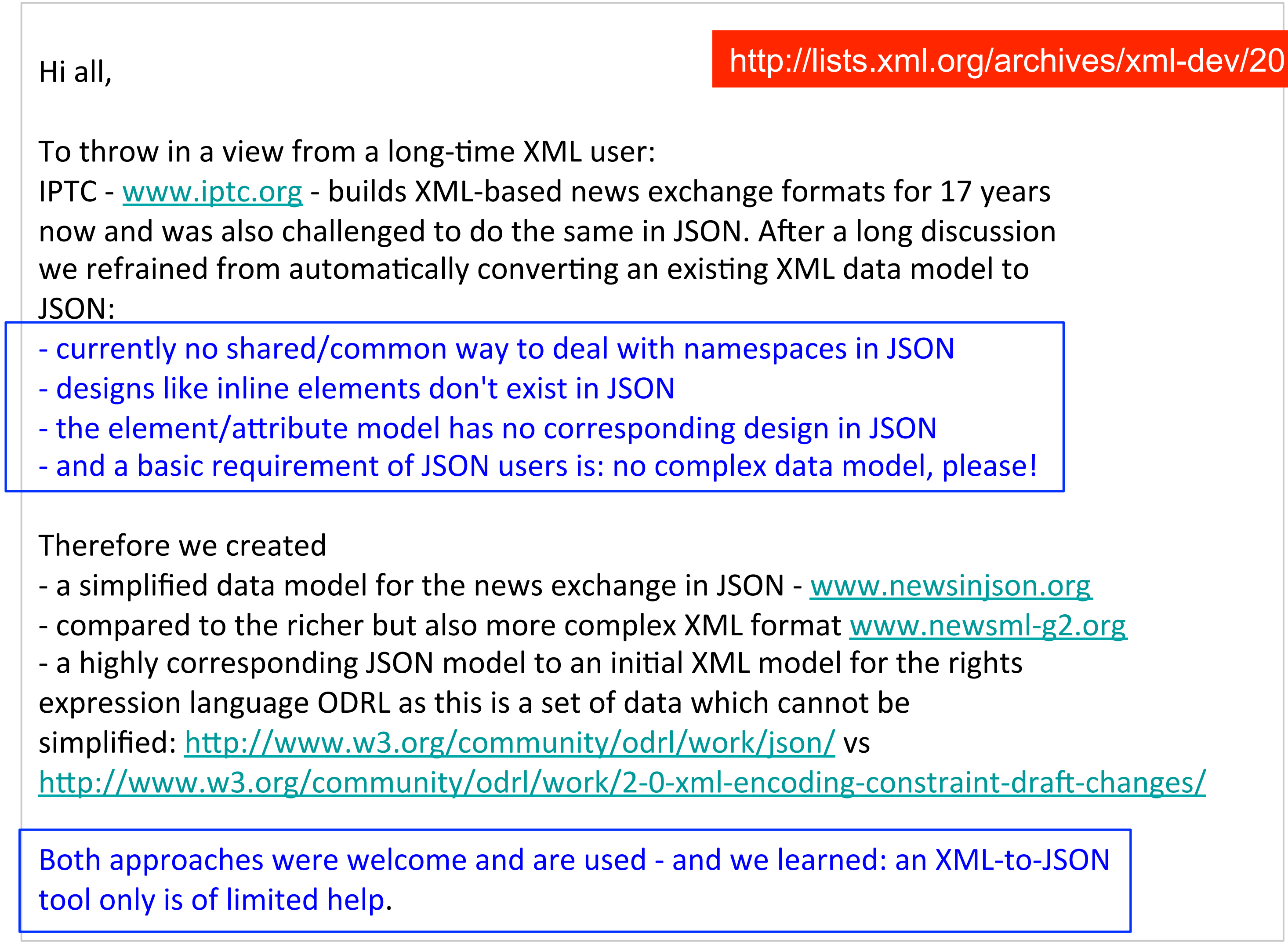
- But …

XML to JSON

- No attributes in JSON
- No namespaces in JSON
  - JSON is a reaction against the complexity of XML and namespaces is one of the biggest culprits, in terms of complexity
- No equivalent of XSLT in JSON
  - JSON people believe that text documents should be processed by a general purpose language such as Java or JavaScript, not a domain-specific language such as XSLT

XML to JSON

http://lists.xml.org/archives/xml-dev/201412/msg00022.html



Hi all,

To throw in a view from a long-time XML user:
IPTC - www.iptc.org - builds XML-based news exchange formats for 17 years now and was also challenged to do the same in JSON. After a long discussion we refrained from automatically converting an existing XML data model to JSON:

- currently no shared/common way to deal with namespaces in JSON
- designs like inline elements don't exist in JSON
- the element/attribute model has no corresponding design in JSON
- and a basic requirement of JSON users is: no complex data model, please!

Therefore we created
- a simplified data model for the news exchange in JSON - www.newsinjson.org
- compared to the richer but also more complex XML format www.newsml-g2.org
- a highly corresponding JSON model to an initial XML model for the rights expression language ODRL as this is a set of data which cannot be simplified: http://www.w3.org/community/odrl/work/json/ vs
http://www.w3.org/community/odrl/work/2-0-xml-encoding-constraint-draft-changes/

Both approaches were welcome and are used - and we learned: an XML-to-JSON tool only is of limited help.

informatika fakultatea facultad de informática

XML vs. JSON

- **JSON** is the best tool for sharing data
  - Data is stored in arrays and records
    - XML stores data in trees (DOM)
  - Data transfers are much easier when the data is stored in a structure that is familiar to object-oriented languages
  - This makes it very easy to import data from a JSON file into Perl, Ruby, Javascript or Python

XML vs. JSON

- **XML** is the right tool for sharing documents
  - It holds any data type and can be used to transport full documents with formatting information included
    - XML is best used when transporting something like a patient chart or text document with markup included
  - XML's strength is extensibility and the avoidance of namespace clashes

Contents

- JSON
  - XML vs. JSON
- Xquery 3.1
  - New in the data model: **map** and **array**
  - New functions: ***json-doc()*** and ***parse-json()***

https://www.w3.org/TR/xpath-functions-31/

XQuery 3.1

- Support for **consuming** JSON data
- Support for maps and arrays in the Data model
  - Allowing for a more natural representation of JSON data in XQuery expressions

informatika fakultatea
facultad de informática

XQuery 3.1. Installation steps

- Oxygen XML Editor: **18.1**
- **Saxon 9.7 XSLT and XQuery Transformer**
  - Install the add-on for **Saxon-EE 9.7 (External)**. Go to menu > Help > "Install new add-ons", pick the "default" update site (https://www.oxygenxml.com/InstData/Addons/default/updateSite.xml) and install **Saxon 9.7 XSLT and XQuery Transformer**.
  - Restart Oxygen
    - Note: Debugging XSLT/XQuery transformations based on this engine is NOT supported.
- **XQuery Transformation**
  - After restarting Oxygen, edit the transformation scenario (or create a new one) and select from the Transformer combo **Saxon-EE XQuery 9.7.0.x (External)**. Press the **"Advanced options"** button (cogwheel), and select your Saxon configuration file. Associate the scenario with the XQuery file

```
<?xml version="1.0"?>
<configuration edition="EE" xmlns="http://saxon.sf.net/ns/configuration">
    <xquery version="3.1"/>
</configuration>
```

config.xml file

Data model: map

- It is a collection of key/value pairs
  - Within a map, each key is unique and the order of the entries has no particular significance
- Can be constructed by
  - a map constructor : **map**{ }
  - a built-in function
  - parsing a JSON
- Namespace with built-in functions for querying and manipulating

  http://www.w3.org/2005/xpath-functions/map

Data model: map. Examples

```
xquery version "3.1";
declare namespace
          output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "json";
map {
     "ACC": "Accounting",
     "SAL": "Sales",
     "MAR": "Marketing"
}
```

```
map {
     "ACC": map {
                   "name": "Accounting",
                   "code": 300 },
     "SAL": map {
                   "name": "Sales",
                   "code": 310 },
     "MAR": map {
                   "name": "Marketing",
                   "code": 320 }
}
```

Data model: map. Query

```
xquery version "3.1";
declare namespace
                output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare namespace map = "http://www.w3.org/2005/xpath-functions/map";
declare option output:method "json";
let $deptnames := map {
                "ACC": “Accounting",
                “SAL": “Sales",
                “MAR": "Marketing"
                }
return
  map {
    "dept1" : map:get($deptnames, "ACC"),
    "dept2" : $deptnames(“SAL"),
    “dept3” : $deptnames?(“MAR”)
  }
```

$deptnames?MAR

Result

```
{
    "dept1“ :  “Accounting",
    "dept2“ :  “Sales“,
    “dept3” :   “Marketing”
}
```

?: “lookup” operator
Multiple keys can be provided

Data model: map. **for-each()**

```
xquery version "3.1";
declare namespace
          output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare namespace map = "http://www.w3.org/2005/xpath-functions/map";
declare option output:method "text";
let $deptnames :=
          map { "ACC": "Accounting", "SAL": "Sales", "MAR": "Marketing" }
let $func := function($k, $v) {concat('Key: ', $k, ', value: ', $v)}
return
          map:for-each($deptnames, $func)
```

iterates over all the entries in a map, and each entry has two components (key, value)
It returns a **sequence**

must be a function that accepts two arguments

Result

```
Key: ACC, value: Accountin
Key: MAR, value: Marketinc
Key: SAL, value: Sales
```

Functions that operate on maps

- **map:size**: returns the number of entries in a map. For example, map:size($deptnames) returns the integer 3.
- **map:contains**: tests whether a map contains an entry with a particular key. For example, map:contains($deptnames, "ACC") returns true, and map:contains($deptnames, "FOO") returns false.
- **map:keys**: returns a sequence of all the keys in the map, in no particular order. For example, map:keys($deptnames) returns the sequence of three strings ("ACC", “SAL", “MAR").

https://www.w3.org/TR/xpath-functions-31/

Data model: array

- It is an ordered list of values
  - The values are called members,
    - They can be retrieved based on their position number
- Can be constructed by
  - an array constructor : array{ } or [ ]
  - a built-in function
  - parsing a JSON
- Namespace with built-in functions for querying and manipulating

http://www.w3.org/2005/xpath-functions/array

Data model: array. Examples

```
xquery version "3.1";
let $arrayone := [1, 32, "a", "7"]
return
      $arrayone(3)
```

Result

```
a
```

```
xquery version "3.1";
let $arraytwo := array{ 1, "a", 34, 43 }
return
   <item>
     <c>{$arrayfour(3)}</c>
     <c>{$arrayfour(4)}</c>
   </item>
```

Result

```
<item>
    <c>34</c>
    <c>43</c>
</item>
```

[1, (32, 19)] ➡

1	32, 19

array { 1, (32, 19) } ➡

1	32	19

Data model: array. Query

```
declare namespace
          array = "http://www.w3.org/2005/xpath-functions/array";
let $arrayints := [10, 20, 30, 40, 50]
let $nestedarray := [10, [20, 30, 40], 50]
return
<item>
   <int>{array:get($arrayints, 2)}</int>
   <int>{$arrayints(3)}</int>
   <int>{$nestedarray(2)(3)}</int>
   <int>{$arrayints?(2)}</int>
   <int>{$arrayints?(2, 3, 4)}</int>
   <int>{$arrayints?*}</int>
</item>
```

Retrieves a sequence containing every member in the array

Result

```
<item>
   <int>20</int>
   <int>30</int>
   <int>40</int>
   <int>20</int>
   <int>20 30 40</int>
   <int>10 20 30 40 50</int>
</item>
```

Data model: array. Query

```
declare namespace
            output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "json";
let $arrayints := [10, 20, 30, 40, 50]
return
     $arrayints
```

Array of integer values

Result

```
[
  10,
  20,
  30,
  40,
  50
]
```

Retrieves a sequence containing every member in the array

```
let $arrayints := [10, 20, 30, 40, 50]
let $sequenceints := $arrayints?*
return
     $sequenceints
```

ERROR: JSON output method cannot handle a sequence of more than one item

Data model: array. **for-each()**

```
declare namespace
          output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare namespace array= "http://www.w3.org/2005/xpath-functions/array";
declare option output:method "text";
let $arrayints := [10, 20, 30, 40, 50]
let $func := function($v) { $v*2 }
return
   <item>
          { array:for-each($arrayints, $func) }
   </item>
```

iterates over all the entries in an array, and each entry has one value.
It returns a **sequence**

must be a function that accepts one argument

Result

```
<item>
     20 40 60 80 100
</item>
```

Functions that operate on arrays

- **array:size**: Returns the number of members in an array. For example, array:size([10, 20, 30]) returns the integer 3.
- **array:head**: Returns the first member of an array. For example, array:head([10, 20, 30]) returns the integer 10.
- **array:append**: Adds one member to the end of the array. For example, array:append([10, 20, 30], 40) returns the array [10, 20, 30, 40].
- **array:filter**: Applies a function to each member in an array and returns an array containing those members for which the function returns true. For example, array:filter([10, 20, 30], function($n) {$n > 15}) returns [20, 30] because those are the members of [10, 20, 30] that are greater than 15.
- **array:flatten**: Turns arrays into sequences of items, recursively flattening any arrays that are within arrays. For example, array:flatten([["a", "b", "c"], ["d", "e", "f"]]) returns a sequence of six strings: "a", "b", "c", "d", "e", "f".
- **array:join**: Merges the members in multiple arrays into a single array, retaining the order. For example, array:join(([10, 20, 30], ["a", "b", "c"])) returns an array with six members: [10, 20, 30, "a", "b", "c"].

https://www.w3.org/TR/xpath-functions-31/

Consuming JSON: *json-doc()*

- It parses an external resource containing JSON and typically returns a map or array

```
xquery version "3.1";
declare namespace
          output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "json";
let $var := json-doc('product.json')
return $var
```

Result:

```
{
 "number“ :  557,
 "name“ :  "Fleece Pullover",
 "colorChoices“ :   [
                 "navy",
                 "black"
  ],
 "other“ :  null,
 "is-current“ :  true
}
```

The content of **product.json**

Consuming JSON: ***parse-json()***

- It parses a string supplied in JSON format and typically returns a map or array

```
xquery version "3.1";                                          a map
let $var := parse-json( ' { "name": “john" ,
                              "age": 52 ,
                              "department": "accounting" } ' )
return  <name> { $var?name } </name>
```

Result

```
<name>john</name>
```

Consuming JSON: ***parse-json()***

a map containing a nested map

```
let $var := parse-json( ' { "employee":
                                   {  "name": “john" ,
                                      "age": 52,
                                      "department": "accounting"
                                   }
                              } ' )
return <name> { $var?employee?name } </name>
```

Result

```
<name>john</name>
```