# Experiences with JSON and XML Transformations

IBM Submission to W3C Workshop on Data and Services Integration
October 20-21 2011, Bedford, MA, USA

21 October 2011

John Boyer, Sandy Gao, Susan Malaika, Michael Maximilien, Rich Salz, Jerome Simeon

Feedback to: malaika@us.ibm.com

# Agenda

- Why Do We Want to Transform
- Friendliness vs Grotesqueness (Unfriendliness)
- Typical JSON and XML Mapping Issues
  - Round-Trippability
  - Friendliness
  - A Use Case
- Mapping Approaches
  - Overview of Approaches 1-4
  - Recommendations
- Ideas
- Some Published Mappings and References
- The Grand Finale

# Why Do We Want To Transform?

- Trend: Javascript/Web 2.0 dominates for:
  - Fast parsing of message content
  - Programmer convenience
- Trend: Multiple message formats (Atom, XML, JSON, etc) are becoming common
  - XML APIs and REST APIs are being extended to support JSON
- Trend: Enterprises desire validation, declarative constraints and stringency for data content, but also want the benefits of Web 2.0
  - Customer demand for:
    - Constraint and query features provided by XML
    - Programming ease provided by JSON

# Some Data and Metadata Standards

| | Relational | XML | JSON | Linked Data |
|---|---|---|---|---|
| Metadata | Data Definition Language (ISO) | XML Schema XSD (W3C) , Namespaces (W3C) | JSON Schema - IETF | RDFS (RDF Schema), Ontology (W3C and elsewhere) |
| Constraints | Integrity Constraints in table definitions (ISO) | Schematron (ISO), Relax-NG (OASIS) | - | |
| Triggers | Relational triggers | - | - | RIF (W3C) |
| Data Exchange Serializations | SQL standard (ISO) defines an XML serialization but it is not widely used – There is no agreed JSON serialization. There are RDF serializations. | XML is a syntax widely used for data exchange (W3C) | JSON is a serialized format, there are XML representations of JSON too | XML, Turtle |
| Annotations | Not part of the relational model | Many kinds of annotations are defined for XML schemas and for XML data | - | RDFa (W3C) can be used to annotate XML |
| Query & CRUD Languages | Data Manipulation Language (DML), SQL, SQL/XML (ISO) | XPath, XQuery (W3C) and others for CRUD | JAQL, JSONiq | SPARQL (W3C) |
| Query & CRUD APIs | Many for various programming languages, e.g., JDBC, ODBC | Various, includes some of the relational APIs and specific APIs, e.g., XQJ | - | SPARQL Graph Store HTTP Protocol – for CRUD (W3C) |
| Collection | Table, View, Database (ISO) | XML Collection Function (W3C) | - | RDF Graphs (W3C) |
| Transformation & Other Languages | SQL (Tables to Tables) | XSLT (XML to text, includes XML); XForms<br>SQL XMLTABLE (XML to Relational) | JavaScript | - |

Some XML communities who work on query and transformation languages are reviewing the idea of XML languages supporting JSON

Currently, the role of JSON is mainly for data exchange between JavaScript clients and servers

# Friendly XML

- Friendly XML has multiple unique paths (multiple element and attribute names) in the XML, rather than a name value pair design
- Friendliness provides
  - Easy XML consumption by programmers, authors and software
  - Straightforward transforms, queries, and indexing capabilities

Friendly XML Example

```
<location>
  <city>Armonk</city>
  <state>NY</state>
</location>
```

UnFriendly XML Example

```
<root type="object”>
<item name=“city” value=“Armonk” />
 <item name =“state” value =“NY” />
</root>
```

# Friendly JSON

- Friendly JSON does not include repeating variable names
- Data types are directly associated with each variable
- Friendliness provides easy data structure consumption by JavaScript programmers, authors

Friendly JSON Example

```
{
  "person": {
    "age": "40",
    "name": "John Smith"
  }
}
```

UnFriendly JSON Example

```
{ "book" :   { "children :
    [{ "title" :  { "attributes" :
        { "isbn": "15115115" },
          "children" :
            [ "This book is ", { "emph" : "bold" }] }
     }]  }
}
```

# Typical JSON and XML Mapping Issues

- Round-trip transformations are challenging
  - XML to JSON to XML
  - JSON to XML to JSON
- General issues
  - Different character mapping for names and values in JSON and XML
  - Mismatch in data types between JSON and XML
  - Semantic infidelity
- XML to JSON issues
  - Losing XML namespaces
  - Incorrectly mapping XML repeating elements
  - Mishandling of relative URLs
- JSON to XML issues
  - Incorrectly mapping JSON arrays
  - Associating variables with data types in JSON

# Round-trip

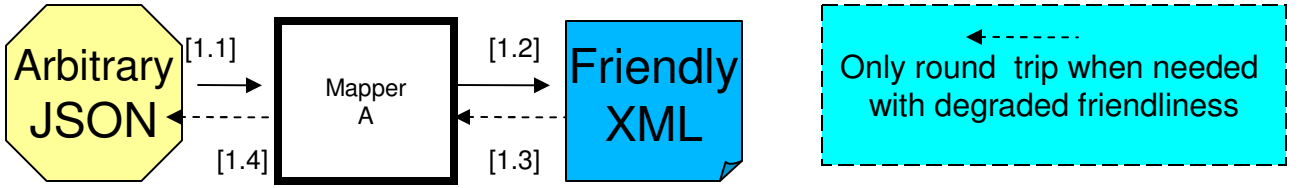
- Why is round-tripping important?
  - Data and data type information has to be preserved
  - Fidelity, to prevent data loss
  - Preservation of metadata, order
  - Maintain usability regardless of format
  - Some details are unnecessary, depending on use cases
  - May not help “Friendliness”

# Mapping Approach Overview

## Approach 1

JSON to XML

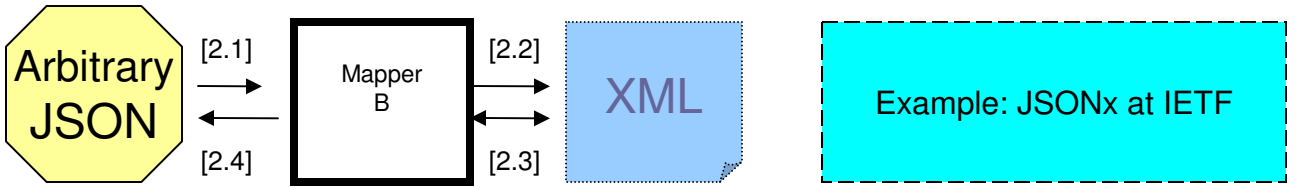
Focus on Friendliness




## Approach 2

JSON to XML

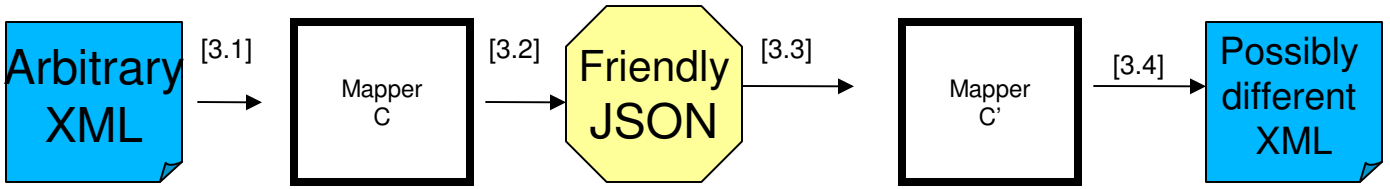
Focus on Round-Tripping




## Approach 3

XML to JSON

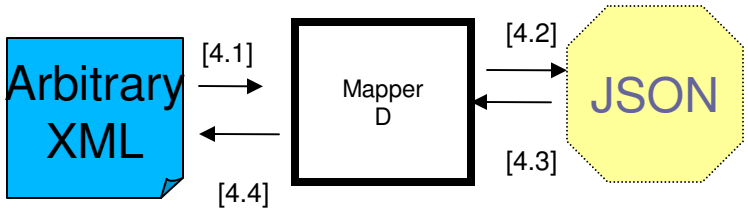
Focus on friendliness




## Approach 4

XML to JSON

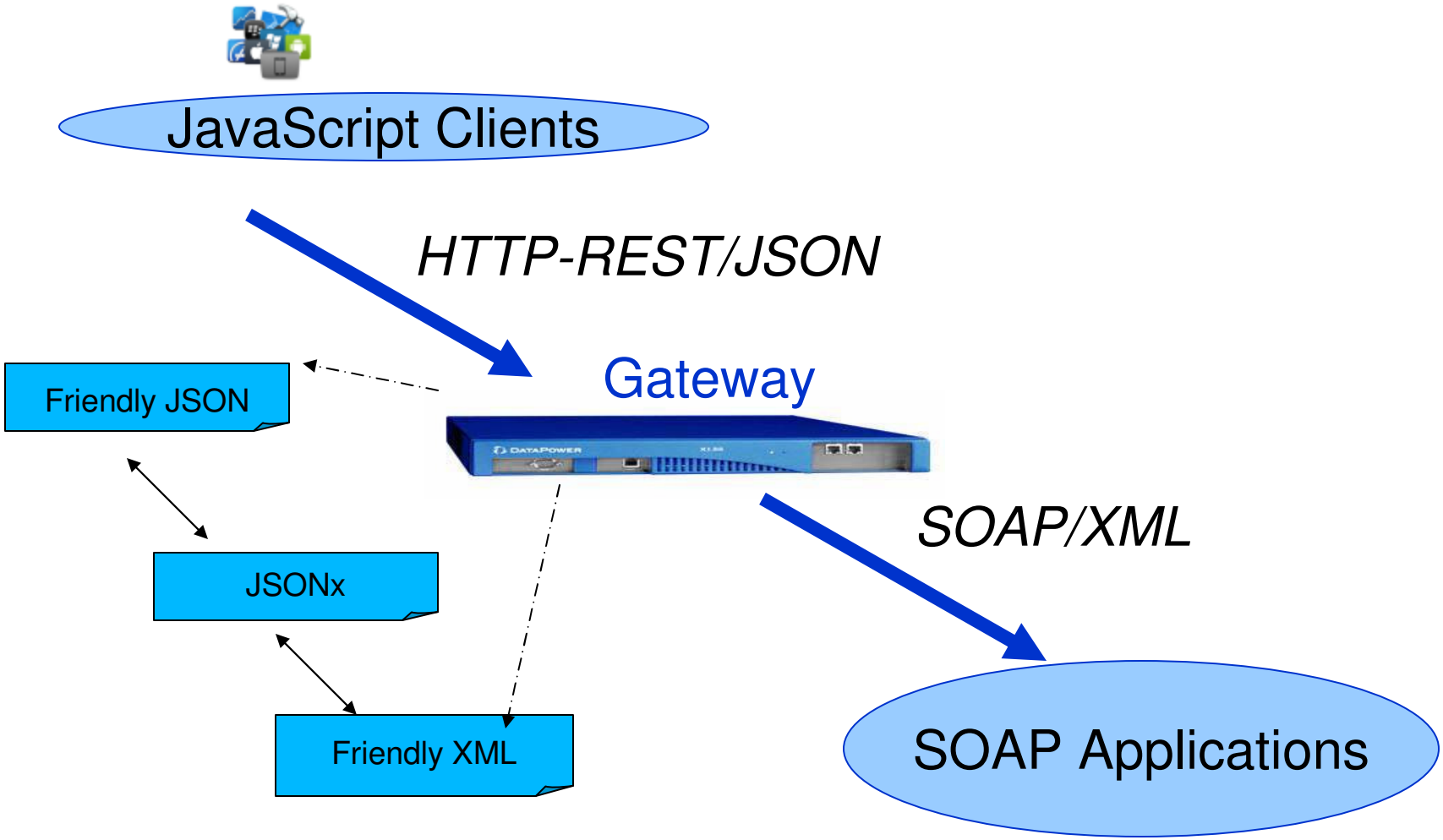
Focus on Round-Tripping



# Friendly and Round-Trippable Summary

- A JSON-to-XML mapping is *friendly* when the generated XML has meaningful element and attribute names, rather than a name value pair design.
- An XML-to-JSON mapping is *friendly* when the generated JSON has flat structure, making it easy for JavaScript programmers to consume it.
- A JSON-to-XML mapping is r*ound-trippable* when the generated XML contains all the information that was in the original JSON, without any loss.
- An XML-to-JSON mapping is *round-trippable* when the generated JSON contains all the information that was in the original XML, without any loss.

# A JSON XML Use Case



# Approach #1: JSON to Friendly XML

- Requirement 1: Friendly XML processing
- Requirement 2: Round-trippable
- Rules:
  - JSON names become XML element names
  - JSON object members and arrays become XML elements, and
  - Simple values become XML text content
  - Synthesized XML 'root' element
  - Special handling to preserve data type, kinds of emptiness, and special characters

# Approach #1: Example

In this example,

```
{
  "city": "Armonk",
  "state": "NY"
}
```

becomes

```
<root type="object">
  <city>Armonk</city>
  <state>NY</state>
</root>
```

# Approach #1: Special characters in names and empty names

A JSON name may not match an XML NCName

- Each illegal character is escaped
  - with two leading underscores,
  - a hexadecimal encoding of the character's Unicode codepoint and
  - A trailing underscore

```
{
  "var$":"value",
  "":"generic"
}
```

```
<root type="object">
 <var__24_>value</var__24_>
 <__>generic</__>
</root>
```

# Approach #1: Simple Values

JSON name/value pair with a simple value

- Generate XML element from name
- Store string equivalent of the simple value as content of the XML element
- Typed as boolean, number, string, or null

```
{
  "age":43,
  "married":true,
  "address":null
}
```

```
<root type="object">
  <age type="number">43</age>
  <married type="boolean">true</married>
  <address nil="true"></address>
</root>
```

# Approach #1: Object Values

JSON name/value pair with a non-null object value,

- Generate XML element based on the name
- Generate child elements for each of the JSON name/value pairs

```
{
  "address": { "street" : "123 Main St." }
}
```

```
<root type="object">
  <address type="object">
    <street>123 Main St.</street>
  </address>
</root>
```

# Approach #1: Arrays

JSON name/value pair with an array value

- Generate a container XML element for the array
- Generate one child XML element for each value in the array

```
{
  "locations": ["Amsterdam", "London"],
  "secondary_locations": []
}
```

```
<root type="object">
  <locations type="array">
    <__>Amsterdam</__>
    <__>London</__>
  </locations>
  <secondary_locations
type="array"></secondary_locations>
</root>
```

# Approach #1: Special Characters In Values

Characters are copied from JSON except for certain conversions:

| | |
|---|---|
| < | &lt; |
| > | &gt; |
| & | &amp; |
| \udddd | &#xdddd |
| \” | “ |
| \\ | \ |
| \/ | / |
| \n | Newline (0x0A) |
| \r | &#xD |
| \t | tab(0x09) |
| Non-XML chars(\b, \f) | Omitted, or encoded as PI |

# Approach #2: JSON to XML and Back

- Requirement: Round-trippable
- Usually a name/value pair
- Rules:
  - XML elements to preserve data type
  - XML elements to preserve objects
  - XML elements to preserve arrays
  - XML attributes/text to preserve property names

# Approach #2: Example

In this example,

```
...
"batters":
    {
      "batter":
          [
            { "id": 2001, "type": "Regular" },
            { "id": 2003, "type": "Blueberry" }
          ]
    },
...
```

becomes

```
<json:object name="batters">
   <json:array name="batter">
     <json:object>
       <json:number name="id">2001</json:number>
       <json:string name="type">Regular</json:string>
     </json:object>
     <json:object>
       <json:number name="id">2003</json:number>
       <json:string name="type">Blueberry</json:string>
     </json:object>
   </json:array>
 </json:object>
```

# Approach #3: XML to Friendly JSON

- Requirement: Consumable/Friendly JSON
- Requirement: Preserve XML structure
- Rules:
  - XML element or attribute names become JSON object names,
  - XML children elements become JSON objects fields or arrays, and
  - XML text nodes become JSON simple values

# Approach #3: Multiple Child Elements With Same Name

When multiple child elements have the same name, use JSON arrays to represent those elements.

```
<people>
  <person>
    <name>John Smith</name>
    <age>40</age>
  </person>
  <person>
    <name>Jane Foster</name>
    <age>43</age>
  </person>
</people>
```

```
{  "people": {
    "person": [
     {
       "age": "40",
       "name": "John Smith"
     },
     {
       "age": "43",
       "name": "Jane Foster"
     }
    ]
   }
}
```

# Approach #3: Attributes and Element Nodes

## Preserve XML notion of attributes

```
<title isbn="15115115">This book is on XML and JSON</title>
```

```
{
  "title": {
    "@isbn": "15115115",
    "text()": "This book is on XML and JSON"
  }
}
```

# Approach #3: Text Nodes and Mixed Content

An XML element could contain both child elements and text

```
<name>John<br/>Smith</name>
```

```
{
  "name": {
    "br": {},
    "text()": "JohnSmith"
  }
}
```

# Approach #3: Handling Document Order

## Preserve document order with JSON array

```
<name>John<br/>Smith</name>
```

```
{
  "name": [
    "br": {},
    "text()": "JohnSmith"
  ]
}
```

# Approach #3: XML Namespaces

JSON does not support a name-scoping mechanism

- Confuses Round-tripping,
- Stops being friendly JSON

```
<n:root xmlns:n=\"foo\" xmlns:l=\"bar\" l:att=\"1\">
  <n:node/>
</n:root>
```

```
{  "foo": {
     "root": {
       "bar": {
         "@att": "1"
       },
       "foo": {
         "node": {}
       }
     }
   }
}
```

# Approach #3: Simple Values and Datatypes

May be necessary to keep track of the original XML type annotation

```
<year xsi:type="xs:positiveInteger">1989</year>
```

```
{ "xsi:type" : "xs:positiveInteger", value : 1989 }
```

# Approach #4: XML to JSON and Back

- Requirement: Round-trippable
- Rules:
  - Elements are mapped as JSON objects with its qualified name as a field
  - Attributes of XML elements are included as a JSON “attributes” object
  - Element children are included as an array to preserve the original order of the element's children

# Approach #4: Example

In this example,

```
...
<book>
  <title isbn=\"15115115\">This book is <emph>bold</emph></title>
</book>
...
```

becomes

```
{ "book" :
   { "children :
      [ { "title" : { "attributes" : { "isbn": "15115115" },
                      "children" : [ "This book is ",
                                     { "emph" : "bold" } ] } } ] } }
```

# Mapping Approach Recommendations

- Use Approach 1 when your highest priority is to make the resulting XML easy to consume and then optionally round-trippable
- Use Approach 2 when your priority is round-trippable conversions
- Use Approach 3 when your highest priority to make the resulting JSON easily consumable and then round-trippable
- Use Approach 4 when your priority is not JSON consumption, but still capable of round-trip conversions

# Ideas and Discussion

- Provide structured guidance to the W3C XSLT and other groups that are looking at integrating JSON and XML ?
- Provide guidance for architects and developers who create APIs and formats which apply to both JSON and XML ?
- Review the OData JSON proposals ? (Contains JSON representation for ATOM )

# Another Idea - Consider creating a W3C REST Community?

Topics could include:

- (1) Identify gaps e.g.,
  - WADL or similar?  URL Query?
  - Expressing REST API results (e.g., RDF; JSON; XML; ATOM) and paging ?
  - JSON Schema http://tools.ietf.org/html/draft-zyp-json-schema-03 ?
- (2) Create Best Practices e.g.,
  - REST CRUD (use of POST as alternative) ;
  - POST CREATE only or TUNNEL anything ;
  - Partial update via POST instead of PATCH
  - Retrieve subset a resource from POST
  - Resource Links / ATOM - HREF and REL ;
    - <atom:link rel='...' href='...'/> ;
    - <machine rel='...' href='...'/>
  - URI opaqueness
  - MIME Types
    - Generic MIME -> Generic Tooling ;
    - Specific MIME class -> Specific Tooling
    - Important for JSON - no guarantee there will be "Root Element (outer element) ; if MIME TYPE JSON/XML don't know anything

# Examples from CWMG
## (Cloud Management Work Group at DMTF)

- Modeling collections : Update collections
  - Allowable operations on a class of resources :
    - <operation url='xxx' rel='xxx'/>
    - <operation url="/foo" rel="stop'/>
- Async processing
  - What does client do after 202 ?
- CWMG Reference:
  - http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.0a.pdf
- CWMG Primer:
  - http://dmtf.org/sites/default/files/standards/documents/DSP2027_1.0.0a.pdf

# Examples from URL Queries

- Queries in URLs have become common
- It is usual to offer the query responses in one or more of XML, ATOM or JSON.
- Here are some examples:
  - Twitter Search API : http://search.twitter.com/api/
    - Example - http://search.twitter.com/search.atom?lang=en&q=&metopera&rpp=15 - Return all English language tweets that contain references to user metopera in chunks of 15 tweets per page - as atom
  - FaceBook Graph API (part of the Facebook Graph Protocol) : http://developers.facebook.com/docs/reference/api/
    - Example - https://graph.facebook.com/me/friends?limit=3 - Return all my friends in chunks of 3 - as JSON
  - OData : http://www.odata.org/developers/protocols/uri-conventions#QueryStringOptions
    - Example - http://services.odata.org/OData/OData.svc/Products?$skip=2&$top=2&$orderby=Rating - Return the 3rd and 4th products when sorted by rating - as atom
    - The OData interface has been adopted by Microsoft and by SAP

# Some Published Mappings

- JSONx: http://datatracker.ietf.org/doc/draft-rsalz-jsonx/
- JAQL: http://code.google.com/p/jaql/wiki/Builtin_functions#xmlToJson()
- XForms: http://www.w3.org/MarkUp/Forms/wiki/Json
- Jettison: http://jettison.codehaus.org/User%27s+Guide#User%27sGuide-Conventions]
- Badgerfish: http://www.sklar.com/badgerfish
- JSON4J: http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.json.help/docs/GettingStarted.html
- Zorba: http://www.zorba-xquery.com/doc/zorba-1.4.0/zorba/xqdoc/xhtml/www.zorba-xquery.com_modules_json.html

# References

- XML: http://www.w3.org/XML/
- JSON : http://www.json.org/
- Convert Atom documents to JSON : http://www.ibm.com/developerworks/library/x-atom2json/index.html
- OData Protocol JSON Format: http://www.odata.org/developers/protocols/json-format

# The Grande Finale