Experiences with JSON and XML Transformations

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

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

Background

While JSON and XML are both used to represent structured data they have disjoint sets of tools and supporting language libraries are different; as such, software engineers find it increasingly necessary to convert between the two formats to take advantage of the available tooling, and to make data available in its "most natural" form to applications.

In this article we introduce the concepts of friendly JSON and friendly XML, and we consider the effects of round-trip capability when defining mappings. We then describe various mapping approaches between JSON and XML as well as a short comparison between them.

The Mapping Approaches

When mapping JSON and XML, these topics are of interest:

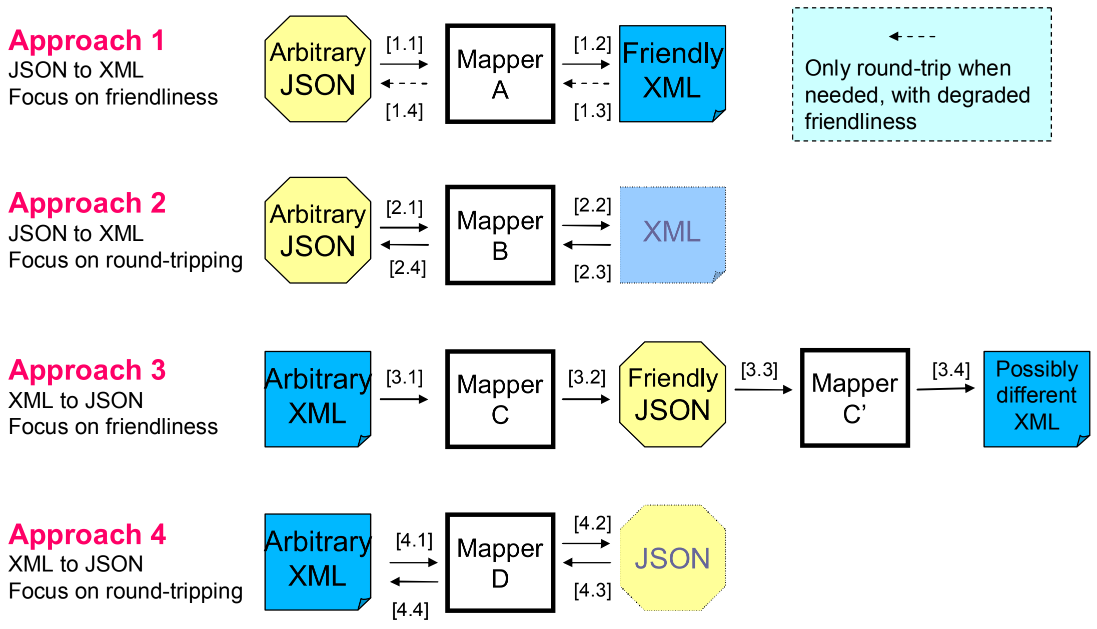
- Round-tripping the messages (XML to JSON to XML; or JSON to XML to JSON)
- Character encodings
- Mapping names and characters
- Mapping data types
- Mapping XML namespaces
- Mapping XML repeating elements
- Mapping JSON arrays
- Associating variables with data types in JSON

In order to characterize mapping approaches, we consider two major aspects: friendliness and round-trippability.

- 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.

We consider four mapping approaches.

Mapping Approach Overview



Approach #1: Arbitrary JSON to XML (Friendly, Round-trippable)

This mapping approach converts JSON to XML in a way that preserves the naming and structure in the JSON to the maximum extent possible. The JSON to XML mapping [1] under development by the W3C XForms group falls into this category.

The first priority is to make the JSON data easily consumable by XML applications, for example easily referenceable by XPath location paths that are similar to JavaScript references to the JavaScript objects that correspond to the JSON. Concretely, JSON object members and array items are mapped to XML elements and simple values are mapped to textual content of the containing element.

```
JSON:

{
  "city": "Armonk",
  "state": "NY",
  "population": 4080
}


XML:

<root type="object">
  <city type="string">Armonk</city>
  <state type="string">NY</state>
  <population type="number">4080
  </population>
</root>
```

The second priority is to "gracefully degrade" the friendliness of the conversion as needed in order to preserve all information in the JSON. This also has the side effect of enabling round-trip conversion of the XML back into JSON:

- JSON data types (string, number, boolean) are captured using additional “type” attributes
- JSON “null” values are represented using special “nil” attributes

- JSON arrays are indicated using “type” attributes, and array items become sub-elements of the array element
- When JSON names contain characters not allowed by XML NCName, they are escaped to form a valid NCName. The original JSON names can optionally be stored in special “name” attributes.

Approach #2: Arbitrary JSON to XML (Round-trippable)

This approach usually uses name value pair approach, for example JSONx [2], which was contributed by IBM to IETF.

```
JSON:
{
  "city": "Armonk",
  "state": "NY",
  "population": 4080
}


XML:
<object>
  <string name="city">Armonk</string>
  <string name="state">NY</string>
  <number name="population">4080</number>
</object>
```

Round-tripping JSON requires that data and data type information are preserved as encodings are applied. For some encodings, data loss can occur because JSON structures do not directly map to the target encoding.

- A JSON structure can be comprised of a variety of data types. In JSON, a numeric value of 547 is distinct from a string value of "547". The boolean value true is distinct from the string "true". So to be able to round-trip to and from JSON with full fidelity, the mapping needs to store metadata describing the JSON type of each simple value.
- Additionally, in JSON a property that contains an array is distinct from one that contains an object. There is no such distinction in XML documents. So you need metadata that describes whether an XML element represents a JSON object or a JSON array.
- And finally, the character set that's valid for a JSON property name isn't a subset of the character set that's valid for an XML element name; so you can't just directly map a JSON property name to an element name. The only option is to store it as data on the element so you can preserve the name.

JSONx satisfies the above round-tripping requirements.

Approach #3: Arbitrary XML to JSON (Friendly, not Round-trippable)

This mapping approach converts XML to JSON in a way that attempts to preserve the naming and structure in the XML document to the maximum extent possible. JAQL, for example, has a function to convert XML to JSON [3].

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

The first priority is to make the XML data easily consumable by JSON applications, for example easily referenceable by object and array access operations that are similar to XPath navigation on the original document.

Round-trippability is notably more difficult than in the JSON to XML case, and existing approaches that attempt at producing friendly JSON are not round-trippable.

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

Approach #4: Arbitrary XML to JSON (Round-trippable)

Mapping XML to JSON in a round-trippable way raises specific challenges. Notably, several aspects of the XML data model do not have a JSON counterpart. Some of the specific differences are:

- XML supports multiple child elements with the same name. In JSON it is unusual to allow object fields with the same name.
- XML supports different kinds of nodes, including notably element and attributes nodes which are distinct.
- XML supports namespaces and qualified names that do not have an equivalent in JSON.
- XML supports document order and allows mixed content.
- XML supports a richer set of data types than JSON (e.g., data/time, binary types).

```
XML:
<book>
  <title isbn="15115115">
    This book is
    <emph>bold</emph>
  </title>
</book>

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

It is still possible to define a generic, round-trippable mapping from XML to JSON. Here is a set of rules that can be used to provide such a mapping:

(1) All elements are mapped as JSON objects with its qualified name as a field.
(2) Attributes of that element are included as an attribute object inside that element.
(3) Children of that element are included as an array that allows preservation of the original order of the element's children.

Comparison of Mapping Approaches

Mapping approaches #1 and #2 are suitable for scenarios where data exists in JSON and needs to be mapped to XML. Mapping approaches #3 and #4 are suitable for scenarios where data exists in XML and needs to be mapped to JSON. Note that it is extremely difficult, if not impossible, to have a single mapping approach that can handle both arbitrary XML and arbitrary JSON.

Approach #1 should be used when the highest priority is to make the resulting XML easy to consume by existing XML tools and programming models; approach #2 should be used when the priority is round-trippable conversions, where the intermediary XML is not accessed directly by XML applications.

Similarly, approach #3 is more suitable when the highest priority is to make the resulting JSON easily consumable, for example by JavaScript of User Interface developers; use approach #4 when the priority is not JSON consumption, but the ability of round-trippable conversions.

Conclusion

As applications become more distributed, in particular as enterprise (or back-end) systems need to be accessed by growing numbers of types of clients, (often mobile, often with limited resources,) the need to map between the "natural" data formats of these two environments is becoming increasingly important.

We have identified several of the characteristics important to these mappings. Depending on the requirements of the particular use-case, one of the approaches we describe would be a better fit then the alternatives. The industry can probably benefit from more study of this area; developers can certainly benefit from guidelines and/or specifications from the World Wild Web Consortium.

References

[1] http://www.w3.org/MarkUp/Forms/wiki/Json
[2] http://tools.ietf.org/html/draft-rsalz-jsonx-00
[3] http://code.google.com/p/jaql/wiki/Builtin_functions#xmlToJson%28%29