Converting Between XML and JSON

May 31, 2006

Stefan Goessner (/pub/au/280)

More and more web service providers seem to be interested in (http://developer.yahoo.com/common/json.html) offering (http://del.icio.us/help/json) JSON (http://labs.evdb.com/archives/2006/03/new_json_output.html) APIs (http://www.plum.com/developer.plum) beneath their XML APIs. One considerable advantage of using a JSON API is its ability to provide cross-domain requests (http://goessner.net/2006/01/remote-json.html) while bypassing the restrictive *same domain policy* of the XmlHttpRequest (http://en.wikipedia.org/wiki/Xmlhttprequest) object. On the client-side, JSON (Javascript Object Notation) comes with a native language-compliant data structure, with which it performs much better than corresponding DOM calls required for XML processing. Finally, transforming JSON structures to presentational data can be easily achieved with tools such as JSONT (http://goessner.net/articles/jsont/).

So if you're working in this space, you probably need to convert an existing XML document to a JSON structure while preserving the following:

- structure
- order
- information

In an ideal world, the resulting JSON structure can be converted back to its original XML document easily. Thus it seems worthwhile to discuss some common patterns as the foundation of a potentially bidirectional conversion process between XML and JSON. A similar discussion can be found at BadgerFish (http://badgerfish.ning.com/) and Yahoo (http://developer.yahoo.com/common/json.html#xml)– without the reversibility aspect though.

A Pragmatic Approach

A single structured XML element might come in seven flavors:

1. an empty element
2. an element with pure text content
3. an empty element with attributes
4. an element with pure text content and attributes
5. an element containing elements with different names
6. an element containing elements with identical names
7. an element containing elements and contiguous text

The following table shows the corresponding conversion patterns between XML and JSON.

Pattern	XML	JSON	Access
1	`<e/>`	`"e": null`	`o.e`
2	`<e>text</e>`	`"e": "text"`	`o.e`
3	`<e name="value" />`	`"e":{"@name": "value"}`	`o.e["@name"]`
4	`<e name="value">text</e>`	`"e": { "@name": "value", "#text": "text" }`	`o.e["@name"] o.e["#text"]`
5	`<e> <a>text</a> <b>text</b> </e>`	`"e": { "a": "text", "b": "text" }`	`o.e.a o.e.b`
6	`<e> <a>text</a> <a>text</a> </e>`	`"e": { "a": ["text", "text"] }`	`o.e.a[0] o.e.a[1]`
7	`<e> text <a>text</a> </e>`	`"e": { "#text": "text", "a": "text" }`	`o.e["#text"] o.e.a`

Please note that all patterns are considered to describe structured elements, despite the fact that the element of pattern 7 is commonly understood as a semistructured element. A pragmatic approach to convert an XML document to a JSON structure and vice versa can be based on the seven patterns above. It always assumes a normalized XML document for input and doesn't take into consideration the following:

- XML declaration
- processing instructions
- explicit handling of namespace declarations
- XML comments

Preserving order

JSON is built on two internal structures:

- A collection of name/value pairs with unique names (*associative array*)
- An ordered list of values (*array*)

An attempt to map a structured XML element...

```
<e>

  <a>some</a>

  <b>textual</b>

  <a>content</a>

</e>
```

...to the following JSON object:

```
"e": {

  "a": "some",

  "b": "textual",

  "a": "content"

}
```

yields an **invalid result**, since the name `"a"` is **not unique** in the associative array. So we need to collect all elements of identical names in an array. Using the patterns 5 and 6 above yields the following result:

```
"e": {

  "a": [ "some", "content" ],

  "b": "textual"

}
```

Now we have a structure that doesn't preserve element order. This may or may not be acceptable, depending on whether the above XML element order matters.

So, our general rules of thumb are:

A structured XML element can be converted to a **reversible** JSON structure, if

- all subelement names occur exactly once, or …
- subelements with identical names are in sequence.

and

> A structured XML element can be converted to an **irreversible** but **semantically equivalent** JSON structure, if
>
> - multiple homonymous subelements occur nonsequentially, and ...
> - element order doesn't matter.

If none of these two conditions apply, there is no pragmatic way to convert XML to JSON using the patterns above. Here, SVG and SMIL documents, which implicitly rely on element order, come to mind.

Semi-Structured XML

XML documents can contain *semi-structured elements*, which are elements with mixed content of text and child elements, usually seen in documentation markup. If the textual content is contiguous, as in:

```
<e>

  some textual

  <a>content</a>

</e>
```

we can apply pattern 7 and yield the following for this special case:

```
"e": {

  "#text": "some textual",

  "a":  "content",

}
```

But how do we convert textual content mixed up with elements? For example:

```
<e>

  some

  <a>textual</a>

  content

</e>
```

It obviously doesn't make sense in most cases to collect all text nodes in an array,

```
"e": {

  "#text": ["some", "content"],

  "a": "textual"

}
```

that doesn't preserve order or semantics.

So the best pragmatic solution is to treat mixed semi-structured content in JSON the same way as XML treats CDATA sections -- as *unknown markup*.

```
"e": "some <a>textual</a> content"
```

Another rule is that XML elements with

- mixed content of text and element nodes and
- CDATA sections

are converted to a reversible JSON string containing the complete XML markup according to pattern 2 or 4.

Examples

Now let's look at two examples using the insight we've gained thus far. Microformats (http://microformats.org/) are well suited because they are an open standard and short enough for a brief discussion.

XOXO (http://microformats.org/wiki/xoxo), as a simple XHTML-based outline format, is one of several microformats. The slightly modified sample from the Draft Specification (http://microformats.org/wiki/xoxo#Simple_XOXO_Fragment) reads:

```
<ol class="xoxo">

  <li>Subject 1

    <ol>

        <li>subpoint a</li>

        <li>subpoint b</li>

    </ol>

  </li>

  <li><span>Subject 2</span>

    <ol compact="compact">

        <li>subpoint c</li>

        <li>subpoint d</li>

    </ol>

  </li>

</ol>
```

Now we apply the patterns above to convert this XML document fragment to a JSON structure.

1. The outer list with two list items is converted using pattern 6.
2. The first list item contains a single textual content *"Subject 1"* and an inner list element. So, it can be treated according to pattern 7.
3. The first inner list is converted with pattern 6 again.
4. Pattern 5 is applied to the second item of the outer list.
5. The second inner list is converted using a combination of patterns 3 and 6.

Here is the resulting JSON structure, which is reversible without losing any information.

```
"ol": {

  "li": [

    {

      "#text": "Subject 1",

      "ol": {

        "li": ["subpoint a", "subpoint b"]

      }

    },

    {

      "span": "Subject 2",

      "ol": {

        "@compact": "compact",

        "li": ["subpoint c", "subpoint d"]

      }

    }

  ]

}
```

hCalendar (http://microformats.org/wiki/hcalendar) is another microformat based on the iCalendar (http://www.ietf.org/rfc/rfc2445.txt) standard. We'll just ignore the fact that the iCalendar format could be more easily converted to JSON, and will look at an hCalendar event example (http://microformats.org/wiki/hcalendar#Example), which is also slightly modified so that it is a structured, rather than mixed, semi-structured document fragment.

```
<span class="vevent">

  <a class="url" href="http://www.web2con.com/">

    <span class="summary">Web 2.0 Conference</span>

    <abbr class="dtstart" title="2005-10-05">October 5</abbr>

    <abbr class="dtend" title="2005-10-08">7</abbr>

    <span class="location">Argent Hotel, San Francisco, CA</span>

  </a>

</span>
```

Here, patterns 2, 3, 4, 5 and 6 are used to generate the following JSON structure:

```
"span": {
  "a": {
    "@class": "url",
    "@href": "http://www.web2con.com/",
    "span": [
      { "@class": "summery", "#text": "Web 2.0 Conference" },
      { "@class": "location", "#text": "Argent Hotel, San Francisco,
    },
    "abbr": [
      { "@class": "dtstart", "title": "2005-10-05", "#text": "October
      { "@class": "dtend", "title": "2005-10-08", "#text": "7" }
    }
  }
}
```

This example demonstrates a conversion that does not preserve the original element order. Even if this may not change semantics here, we can do the following:

1. state that a conversion isn't sufficiently possible.
2. tolerate the result if order doesn't matter.
3. try to make our XML document more JSON-friendly.

In many cases the last point may be not acceptable, at least when the XML document is based on existing standards. But in other cases, it may be worth the effort to consider some subtle XML changes, which can make XML and JSON play nicely together. Changing the `<abbr>` elements to `<span>` elements in the *hCalendar* example would be an improvement.

XML is a document-centric format, while JSON is a format for structured data. This fundamental difference may be irrelevant, as XML is also capable of describing structured data. If XML is used to describe highly structured documents, these may play very well together with JSON.

Problems may arise, if XML documents do the following:

- implicitly rely on element order
- contain a lot of semi-structured data

As proof of this concept, I have implemented two Javascript functions (http://goessner.net/download/prj/jsonxml/) -- `xml2json and json2xml` -- based on the six patterns above, which can be used for the following:

- client-side conversion
    - a parsed XML document via DOM to a JSON structure
    - a JSON structure to a (textual) XML document
- implementing converters in other server side languages

Future XML document design may be influenced by these or similar patterns in order to get the best of both the XML and JSON worlds.