Towards Efficient and Unified XML/JSON Conversion - A New Conversion

Šandrih, Branislava; Tošić, Dušan; and Filipović, Vladimir

Abstract: ***It is important to choose the most appropriate format for data representation that will result in best performance and meet all needed requests. Nowadays, XML and JSON are the two most commonly used formats for data representation and/or interchange. When switching from one format for data representation to another, existing converters can be used. In this paper, we describe a converter from XML to JSON format and vice versa that we developed. Subsequently, we compare our converter with an existing, publicly available converter. We found that there are some differences and difficulties in conversion when applying different converters. Some results of our comparison are presented here.***

Index Terms: ***JSON to XML, XML to JSON, data format converter***

1. Introduction

For the past two decades, XML has been the most
used format for structured data representation and interchange. It is defined by the W3C's XML 1.0 Specification in 1996 (see [1]), and can represent various data: text, images, mathematical expressions, calendars, music sheets, etc. As a consequence, vast amount of related tools have been developed. Afterwards, a different new data representation format was proposed. This notation was called JSON, and it is tightly related to object representation in JavaScript. This representation turned out to be more compact. Since every JSON object is a valid JavaScript object, objects' deserialization is not needed during processing.

One of the possible drawbacks of JSON is lack of a standardized schema. At the moment, there is only a draft available online (see [2]). On the other hand, there are two ways to describe XML document's content, known as native Document Type Definition (DTD) and derived XML Schema

Definition (XSD). Easy translation from one schema to another is not possible, which may be one of the reasons of non-existence of a standardized schema. More information on proposed ways of translation can be found in [3] and [4].

Apart from this constraint, it is possible to translate from one format for data representation to another. In this paper we describe a converter we proposed and implement, and some features that we developed as the way of implementation. This converter is then compared with one of the existing online converters. Although there are several cases when conversions are not strictly feasible, it is guaranteed that no data will be lost.

The paper is organized as follows. Section 2 is a review of related work. Brief descriptions and comparisons between two formats for data representations are also given. Section 3 contains detailed information on the proposed converter and its implementation. Section 4 contains analysis of the featured converter in comparison to the existing online converter. In Section 5, results of performance comparison are presented and discussed. Section 6 contains some final remarks and recommendations.

2. Related Work

An Extensible Markup Language (XML) document forms a hierarchical structure that starts with the root element. The fundamental design considerations of XML include simplicity and human readability. Every XML document must be "well-formed", i.e. be in accordance with strict rules, defined by a XML schema. It does not contain predefined tag sets and each valid tag is defined by either the user or through a different automated scheme. As a consequence of this strict structure definition, content can be easily validated. In contrast, XML documents are difficult to parse and may be considered too redundant. In fact, every XML element has an opening and closing tag, which is repetitive and requires longer document reading time.

In JavaScript Object Notation (JSON) format, data is presented as a set of key-value pairs, with curly brackets representing objects and angular brackets representing arrays. Because of simple syntax and structure, JSON cannot store all data types and it is therefore not suitable for data validation. Syntax and some examples are fully described in [5] and [6]. Some of the strengths of JSON over XML listed in [6] and [7] include: good support from JavaScript libraries, simple API for many programming languages, completely programmed technique for deserializing and serializing JavaScript objects with very little coding, speedy deserialization and web-browsers' support. Some of the strengths of XML over JSON are: strong support for name spaces, support by most of the development tools, XML schema and DTD which can be used to define grammar rules, etc.

XML might be an ideal choice for transferring documents with a lot of different data types and elements. Meanwhile JSON is better suited for dynamic web applications and simple data transmissions (see [8]). Relational databases can also import and manipulate with both XML and JSON data. Different comparisons and measurements under varying transmission scenarios have been conducted in [9]. Some applications of JSON format for data representation, in context of web applications, are given in [10], [11] and [12]. Common conclusion is: JSON is faster and uses fewer resources than XML. For instance, results presented in [13] show that JSON is superior to XML as a data loading tool for AJAX applications. Yet, because of its rich features, XML will always have its place in the transfer and validation of documents.

The actuality of both XML and JSON formats has as a consequence a fact that a lot of users are interested in comparison and conversion between these formats. In references [7], [8] and [9], the comprehensive analysis and comparison of XML and JSON web technologies are presented. Various libraries of object serialization (in XML, JSON and binary formats), from qualitative and quantitative aspects, are compared in [14]. Using each library, a common example is serialized to a file. The size of the serialized file and the processing time are measured during the execution to compare all object serialization libraries. The author concluded that some libraries show the performance penalty, but it is clear that there is no unique best solution.

Moreover, in paper [15] data transmission overhead time and the deserialization efficiency of JSON, XML and FSV (fixed sized file) are quantitatively compared. Obtained experimental data show that the data transmission efficiency of JSON is higher than other data transmission formats, which gives direction in choosing data transmission format for lightweight applications.

There are many publicly available publications that deal with transformations from and to JSON format. In paper [12] the authors introduce a data mapping template, based on context-free grammar, which is used for mapping JSON data to Java data. An implementation model, based on two pushdown automata that recognizes previously defined templates, is also introduced.

Significant body of work is dedicated to the consecutive problem of translation from XML schema to JSON schema (see: [3] and [4]). In [16], a recursive algorithm, based on the multi-tree data structure of XML and JSON objects, is proposed. Translation between these two types of data serializing forms and the efficiency of this algorithm is checked by translation experiments. Another approach, described in [17], aims toward creation of the transformation schema for annotation driven JSON/XML bidirectional transformations. There are many resources on the Internet (for example: [18], [19], [20], [21], [22] and [23]) that are directly related to conversion from XML to JSON format and vice versa.

3. Proposed Converter and Its Implementation

There are rising number of services that, beneath existing XML APIs, offer JSON APIs as well.
Beside smaller representation of data, JSON has more advantages. For instance, it is able to provide cross-domain requests and thus successfully deals with the same origin policy. JSON comes with a native language-compliant data structure, and therefore does not need to make corresponding DOM calls required for XML processing. Vast amount of applications are changing their data representation format. It is important that data represented in the new manner remain unchanged, i.e. to preserve structure, order and information. For this reason, high quality converters are required. Similarly, quantity of this data is most often very large. Therefore, converters need to be as efficient as possible.

We implemented a converter that is available at http://poincare.matf.bg.ac.rs/~branislavas/app/. It was implemented in JavaScript. JavaScript was chosen because of its superset relation to JSON objects. Conversion from XML format for data representation to its JSON representation consists of two steps: 1) building JavaScript object and 2) deserializing it. Throughout the step of building JavaScript object, for the input XML DOM tree of the document, its child nodes' types are being examined. Node can be: element, text or CDATA section. Comments and other types of nodes are not supported, since they could not be represented in JSON. In case if node is an element, its attributes are firstly gathered as fields of corresponding JavaScript object. Afterwards, simple fixture of this element's children is being examined, i.e. number of CDATA sections, text and other contained elements is counted. Depending on the number of the text content and CDATA sections, objects are built in slightly different manners. If an element is not a leaf, i.e. the element has its children, the same function is being recursively called for the subtree with this child element as root. Pseudo code of this function is shown in Figure 3.1.

```
input: XML DOM tree
output: JavaScript object
_____________________________________________crea
te empty JavaScript object

collect attributes of the root node
  store each value named after that attribute's name

examine nature of the children nodes
  count text children
  count cdata children
  count element children

if node has no attributes and no children
   it should be null
else
  if node has element children
   recursively apply previous steps on each child element
  else if node has text children
    save content as "#text" field of output object
  else if node has cdata children
    save content as "#cdata" field of output object

return created JavaScript object
```

Figure 3.1: Building JavaScript object from XML DOM root element

Step of deserialization could be done with built-in JavaScript function *JSON.stringify()*. Instead, function that does this in accordance with the rest of converter's logic was written. Firstly, not all browsers support this function. Secondly, it does not indent its content. Finally, CDATA sections are not JSON-specific, and they are handled as rather special class of tags; eventually, they are manipulated with special attention to the final end.

In the process of converting input JSON to the proper XML representation, steps are converter-independent. Generic JSON string is parsed as the following. Open tag (<) is being written followed by attributes-values pairs; in case this element has children, closing tag (>) is being added and the function is called recursively with children elements as arguments. Text and CDATA sections are written as-is. After recursive call, tag of the parent element closes. If element has no children, it is immediately being closed (/>)

4. Comparative Analysis

This section is dedicated to comparative analysis of two XML/JSON converters. First of these converters (in the following text, converter A), was developed by the authors of this paper (see [24]). The other converter was selected as a top ranking result on Google search engine, retrieved with "XML JSON online converter" query (in the following text, converter B, see [25]). As can be seen in [14], the unique best converter does not exist.

A comprehensive overview is specified within Table 4.1. The first seven cases have been isolated as basic structural descriptions. The following eight cases are rather converter specific or the conversion is somewhat uncertain and ambiguous, i.e. even human who understands both representation would be uncertain what the correct conversion would be. Since multiple solutions are possible, we try to provide one that loses least information (mostly none) and which would be the easiest to parse afterwards.

Cases' enumerations are given in the first column. The second column, *XML*, contains starting XML collection-of-tags to be converted. The third column, *JSON = A(XML)*, represents the first converter's output after conversion of the XML given in the previous column, while *JSON = B(XML)* represents the second converter's output for the same input. This output in JSON is then converted back to XML, and the consequent XML is given in *XML = A(JSON)*$^{-1}$ (i.e. *XML = B(JSON)*$^{-1}$). The column titled *F* indicates whether the conversion given in the concrete row is: feasible (y), partially-feasible (p) or non-feasible (n). If the obtained XML matches the starting one, conversion is feasible. In case when XML was not successfully restored, i.e. data is lost, conversion is non-feasible. When all data contained within original XML is also contained in the final one, but the order of elements was corrupted, the conversion is considered to be partially feasible.

4.1. Basic Cases

The first seven cases are chosen to cover cases of: empty elements, elements with text only, elements with attributes only, elements with both text and attributes, elements that contain other elements of different type, elements that contain other elements of the same type and combined elements. As it can be seen from the first seven table rows, both converters perform the same on elemental cases. The only difference is the character indicating an attribute. In some sense, converter A has more php-specific notation, using '*$*' symbol, while converter B uses '*-*'. The eighth case covers the presence of CDATA section. CDATA (Character Data) is used to indicate that the content in between these strings includes data that *could* be interpreted as XML markup, but should not be. First converter completely preserves CDATA element with its content, storing the complete tag as an element's value. On the contrary, the second converter extracts the content from the tag and stores the content as the value of the element in JSON format. Since there is no indication that this content belongs to the CDATA section, content is restored as plain text. Therefore, this conversion was unsuccessful.

4.2. Special Cases

Row nine covers the case when an element contains more than one CDATA sections. Converter A stores the content in the same manner as in the previous case, and again restores the whole content successfully. Similarly, converter B is unable to restore the data. The tenth case combines CDATA with other data. The problem with this case is preserving the order. Converter A aggregates all textual information into one *#text* element and that way preserves the data. Yet, when trying to restore the original XML, it cannot be certain about the order of textual content contained in the element. This is solved by joining all text on the same level. Although the converter was not able to recreate the order, it did not lose any of the content, so this conversion is considered to be partially feasible. On the other hand, converter B again extracts the CDATA content and joins it with other content. Since there is no indication of CDATA elements' presence, the content is lost. In the case of an empty element having an attribute without value, the second converter outputs something that is not recognized as a valid JSON representation. Consequently, this result could not be parsed, so this conversion is concluded to be totally unsuccessful. Moreover, as can be seen in the eleventh row, the first converter successfully converts to valid JSON and fully restores the starting XML.

The case #12 represents the similar case to the tenth. Instead of CDATA element, a regular element is combined with text content before and after the element. At this point, both converters perform similarly. They did not manage to recreate the order, but the loss of data did not happen in this case. This can be labeled as partially feasible conversion. The case #13 combines textual content, regular element and CDATA section. Converter A completely restores the starting XML, without corruption of order or loss of data. Yet, the converter B ignores element specified after CDATA section, and therefore loses child-element and all its content. The case #14 combines two child-elements with CDATA section. When CDATA section exists, the converter B ignores everything except eventually the text, so both child-elements do not exist in the restored XML. The only existing element is CDATA section, not in its original manner, but as a regular element named *#cdata-section*. It can be seen that, in case when other elements are present, converter B recognizes CDATA as a special tag. Again, converter A manages to produce starting XML. The last case combines textual content with empty child-elements. It might be expected that this case equals to case twelve. For converter A, it turns out that this case is redundant. Yet, in case of converter B, empty element is absent in JSON representation. Therefore, restoring starting XML could not be accomplished, and this conversion is unsuccessful.

5. Performance Comparison

This section is dedicated to performance comparison of the two converters. Discussion about formats themselves and their efficiency can be read in detail in [9]. The choice of data interchange format has significant consequences on data transferring rates and performance. For systems that, for various reasons, have to change data representation format, it is important that this conversion takes the least time possible. This is even more important if these systems use APIs for both formats and have to switch between formats occasionally.

Performance comparison was conducted in the next manner. XML document was firstly generated, containing 2166 child elements inside root element, with exactly four leaf-children each. This number of child elements was doubled three times afterwards. Conversion comparison was performed on Intel i3 CPU @2.20 GHz, 6GB RAM, Win 8.1 64bit operating system, with 28 Mbps download and 2 Mbps upload speed. Google Chrome 51.0.2704.63 was used as Internet browser and testing environment.

In Figure 5.1 and Figure 5.2 *x*-axis represents number of objects used for conversion and *y*-axis represents duration of the conversion. When converting from XML to JSON, as can be seen in Figure 5.1, complexity was linear for both converters. The converter A was showing slightly worse performance (e.g., in the last case, 310 ms in favor of the second converter), when the number of child-objects was increasing.

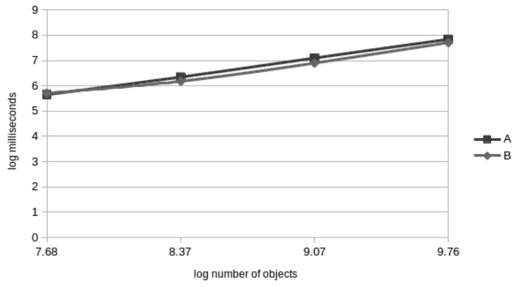

Figure 5.1: Time (in milliseconds) needed for converting x number of XML objects to JSON format

As can be noted from Figure 5.2, unexpected results appeared in the reverse conversion. Resulted JSON output was converted back to original XML. Time needed for the converter A stayed below 1s in all four cases. Time needed for the converter B grew exponentially. In the last case, converting 17328 objects took 2.2s for the converter B, and only 924 ms for the converter A. We believe that this is due to parsing an XML DOM tree in the XML to JSON conversion. This is not required at all, since JSON is a JavaScript object itself.

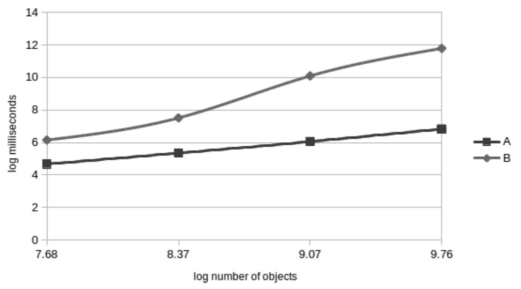

Figure 5.2: Time (in milliseconds) needed for converting x number of JSON objects to XML format

Because of this property, this consequent JavaScript should be directly observed, and not the XML DOM tree that was firstly generated by the Internet browser.

6. Conclusion

Both XML and JSON, as data storage and transmission formats, have their specific assets. When transferring documents with wide variety of data types and elements, XML might be a better option. In spite of its expressiveness, XML tags are redundant, which makes it “heavy” for transfer. On the contrary, JSON is simpler format and avoids redundancy, but supports less features (e.g. lack of validation, comments and name space support). Choice of format itself can depend on other available technologies, experience, nature of data, and even personal preferences and experience.

When switching to another format for data representation is required, mentioned converters can be used. It is not absolutely clear how to preserve data content and its order. Some proposals have been given. It can be seen that both converters suggest different conversions in several cases. One can choose from these two depending on personal preferences and reasoning. Some of suggestions could be the following. If CDATA sections appear among data, and these sections’ content is important, converter A might be more adequate choice. One should have on mind that data loss is possible with converter B. Similarly, when converting from

TABLE 4.1: RESULTS OF CONVERTING FROM XML TO JSON AND VICE VERSA, USING TWO DIFFERENT CONVERTERS

#	**XML**	**JSON = A(XML)**	**XML = A(JSON)$^{-1}$**	**F**	**JSON = B(XML)**	**XML = B(JSON)$^{-1}$**	**F**
1	<element />	{ "element": null }	<element />	Y	{ "element": null }	<element />	Y
2	<element> text </ element>	{ "element": “text” }	<element> text </ element>	Y	{ "element": “text” }	<element> text </ element>	Y
3	<element attr="value" />	{ "element":{ "$attr":"value" } }	<element attr="value" />	Y	{ "element":{ "-attr":"value" } }	<element attr="value" />	Y
4	<element attr="value"> text </ element>	{ "element":{ "$attr":"value", “#text”: “text” } }	<element attr="value">text</ element>	Y	{ "element":{ "-attr":"value", “#text”: “text” } }	<element attr="value"> text </ element>	Y
5	<element> <a>text</a> <b>text</b> </ element>	{ "element":{ "a":“text”", “b”: “text” } }	<element> <a>text</a> <b>text</b> </ element>	Y	{ "element":{ "a":“text”", “b”: “text” } }	<element> <a>text</a> <b>text</b> </ element>	Y
6	<element> <a>text</a> <a>text</a> </ element>	{ element: { "a":[“text”, “text”] } }	<element> <a>text</a> <a>text</a> </ element>	Y	{ element: { "a":[“text”, “text”] } }	<element> <a>text</a> <a>text</a> </ element>	Y
7	<element> text <a>text</a> </ element>	{ "element":{ "#text":“text”", “a”: “text” } }	<element> text <a>text</a> </ element>	Y	{ "element":{ "#text":“text”", “a”: “text” } }	<element> text <a>text</a> </ element>	Y
8	<element> <![CDATA[content]]> </element>	{ "element": "<![CDATA[content]]>" }	<element> <![CDATA[content]]> </element>	Y	{ "element": " content " }	<element> content </element>	N
9	<element> <![CDATA[content1]]> <![CDATA content2]]> </element>	{ "element": "<![CDATA[content1]]> <![CDATA[content2]]>” }	<element> <![CDATA[content1]]> <![CDATA[content2]]> </element>	Y	{ "element": " content1 content2" }	<element> content1 content2</element>	N
10	<element> text1 <![CDATA[content]]> text2 </element>	{ "element":{ "#text":"text1 text2", "#cdata":"content " } }	<element> text1 text2 <![CDATA[content]]> </element>	P	{ "element": "text1 content text2" }	<element> text1 content text2 </element>	N
11	<element attr=””/>	{ "element": {"$attr":"" }}	<element attr=””/>	Y	{ {”element”: { }} }	Invalid JSON	N
12	<element> text1 <a>content</a> text2 </element>	{ "element":{ "#text":" text1 text2", "a":"content" } }	<element> text1 text2 <a>content</a> </element>	P	{ "element": { "#text": ["text1 ", “text2”], "a": "content" } }	<element> text1 text2 <a>content</a> </element>	P
13	<element> text <![CDATA[content]]> <a /> </element>	{ "element":{ "#text":" text", "#cdata":"content", “a”:null } }	<element> text <![CDATA[content]]> <a /> </element>	Y	{ "element": { "#text": "text", "#cdata-section": " content" } }	<element> text <#cdata-section> content </#cdata-section> </element>	N
14	<element> <a /> <![CDATA[content]]> <b /> </element>	{ "element":{ "a":null, "#cdata":"content", "b":null } }	<element> <a /> <![CDATA[content]]> <b /> </element>	Y	{ "element": { "#cdata-section": "content" } }	<element> <#cdata-section> content </#cdata-section> </element>	N
15	<element> x <c/> y </element>	{ "element":{ "#text":"x y ", "c":null } }	<element> x y <c/> </element>	P	{ "element": { "#text": ["x","y"] } }	<element>xy</element>	N

JSON to XML, especially for large files, time grows exponentially with converter B. Invertibility might also be important criterion. As seen from the previous, converter A gave feasible outputs more often.

We tried to offer the solution that would result in preserving most of the information. We managed to save the content, but in some cases the order could not be restored. In future research, we intend to improve the algorithm, so that it will be able to store and apply ordering on elements.

REFERENCES

[1] W3C XML Specification, https://www.w3.org/TR/REC-xml/ (URL).

[2] Internet draft of JSON scheme, http://json-schema.org (URL).

[3] Nogatz F. and Fruhwirth T, "From XML Schema to JSON Schema: Translation with CHR", Proceedings of the 11th International Workshop on Constraint Handling Rules, 2014, pp. 1-8.

[4] Droettboom M., et al., "Understanding JSON Schema Release 1.0",http://spacetelescope.github.io/understanding-json-schema/UnderstandingJSONSchema.pdf (URL), 2015.

[5] Ecma International, "Standard ECMA-404 -- The JSON Data Interchange Format", Ecma International, https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (URL), 2013.

[6] Severance C, "Discovering JavaScript Object Notation", Computer, Issue No.04 (vol.45), pp: 6-8, 2012.

[7] Haq Z.U., Khan G.F. and Hussain T, "A Comprehensive analysis of XML and JSON web technologies", New Developments in Circuits, Systems, Signal Processing, Communications and Computers, 2013, pp. 102-109.

[8] Šimec A., Magličić M, "Comparison of JSON and XML format for data representations", Proc. of: Central European Conference on Information and Intelligent Systems, 2014, pp. 272-275.

[9] Nurseitov N., Paulson M., Reynolds R. and Izurieta C, "Comparison of JSON and XML Data Interchange Formats: A Case Study", Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE 2009, November 4-6, 2009, San Francisco, California, USA, pp. 157-162.

[10] Chasseur, Li, Patel, "Enabling JSON Document Stores in Relational Systems", in WebDB 2013, http://pages.cs.wisc.edu/~chasseur/pubs/argo-long.pdf (URL), 2013.

[11] Keith A:, "RDF/JSON, "A Specification for serialising RDF in JSON", in SFSW 2008, 2008.

[12] Peng, Cao. Xu, "Using JSON for Data Exchanging in Web Service Applications", Journal of Computational Information Systems 7: 16 (2011) pp. 5883-5890.

[13] Boci L., "Comparison between JSON and XML in Applications Based on AJAX", International Conference on Computer Science & Service System (CSSS 2012), pp. 1174 1177, 2012.

[14] Maeada K., "Performance evaluation of object serialization libraries in XML, JSON and binary formats", Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP 2012), pp. 177-182, 2012.

[15] Gao J. and Duan H., "Research on data transmission efficiency of JSON", Computer Engineering and Design 2011-07, 2011.

[16] Wang G., "Improving Data Transmission in Web Applications via the Translation between XML and JSON", Third International Conference on Communications and Mobile Computing (CMC 2011), pp. 182-185, 2011.

[17] Lee, D. A., "JXON: an Architecture for Schema and Annotation Driven JSON/XML Bidirectional Transformations", Proceedings of Balisage: The Markup Conference, Balisage Series on Markup Technologies, vol. 7, doi:10.4242/BalisageVol7.Lee01 2011.

[18] Converters from freeformatter site, http://www.freeformatter.com/xml-to-json-converter.html (URL).

[19] Converter from Code beautify, http://codebeautify.org/xmltojson (URL).

[20] Converter from HTTP Utility, http://www.httputility.net/xml-to-json-to-xml.aspx (URL).

[21] Converter from Browserling, https://www.browserling.com/tools/xml-to-json (URL).

[22] Converter from Online tools, http://online-toolz.com/tools/xml-json-convertor.php (URL).

[23] XML to JSON and vice-versa converter, as well as converter from JSON to different data formats, http://convertjson.com/json-to-xml.htm (URL).

[24] Converter developed by the authors of the paper, http://poincare.matf.bg.ac.rs/~branislavas/app/ (URL).

[25] Utilities-Online converter, http://www.utilities -online.info/xmltojson/ (URL).