Use of JSON to Supplement XML

Prepared by the PESC Technical Advisory Board

5/21/2015

Problem

JavaScript Object Notation (JSON) has become a popular alternative to XML for various reasons: It is less verbose, has simpler syntax than XML and is easily generated and consumed by client side JavaScript. JSON has found extensive use in **RE**presentational **ST**ate Services (REST or RESTful Web services) as the format for sending and receiving structured data for web applications. The older XML-based web services based upon Simple Object Access Protocol (SOAP) has fallen out of favor and is used with few new applications. While some RESTful web services still provide a choice of XML or JSON, many of web services provide only JSON.

To prepare for the same trend occurring in the data exchange realm, PESC must be able to support JSON when the education community requires it. To do this, PESC must have JSON solutions available when needed rather than be forced to react to industry trends.

This document, identifies the issues in translating between the two notations, and provides a recommendation on how PESC might proceed in incorporating JSON into its standards.

Conversion Issues

There are several differences between XML and JSON that may make translation difficult:

1. JSON Objects are not equivalent to XML Complex elements. In JSON, the order of the object properties is not required to be maintained like an XML sequence. Also, JSON object property names need to be unique while XML sub-elements can be duplicated.
2. Processing instructions and comments do not have an equivalent structure in JSON.
3. JSON does not have a date type, and thus, a string representing a date in JSON may need to be recognized as a date and translated into an XML date type format.
4. JSON has less strict naming rules than XML. Direct translation of names from JSON to XML may require the creation of new element names.
5. XML has standard validation specifications while JSON does not. While there is currently activity on creating a JSON schema, XML Schema language is the best supported method to specify and validate instance documents.

Recommendation

The TAB recommends that JSON conformant exchanges be sanctioned by PESC under the following conditions: For those applications requiring JSON exchange, the exchange can be classified as PESC conformant if the JSON is derived from a PESC schema validated XML instance document using the translation rules specified in this document (or using the PESC sanctioned XSLT conversion program).

In addition, PESC will provide an XSLT stylesheet that implements the conversion rules for use by the education community. The TAB has found an open source XSLT that can be used for this purpose.

Translation Rules

The rules below are recommended for adoptions as a PESC standard:

Object	Source example	Rule	Result
Element with only text	<A>text</A>	The element name is represented as JSON property name and the	"A":"text"

		text as the JSON property value	
Namespace	<ns:A>text</ns:A)	The element name includes the namespace prefix.	"ns:A:":"text"

Element with only an attribute	<A x="att value"/>	The element is represented as a JSON object with the attribute represented as a name value pair property. To allow future potential two way conversion the attribute name is prefixed by "@".	"A":{
			"@x":"att value"
			}

Element with attribute(s) and text	<A x="att1 value" y="att2 value">text</A>	The property name "$" is used as the name of the property that has a value of the content of the element.	"A":{
			"@x":"att1 value",
			"@y":"att2 value",
			"$":"text"
			}

Null Elements	<A/>	The value of null elements is the JSON null unless there is an attribute.	"A":null

Complex Elements	<A>
	<B>text 1</B>
	<C>
	<D>text 2</D>
	</C>
	</A>
		Complex element contents will be treated as properties of the object that is named after the top level element	"A":{
			"B":"text 1",
			"C":{
			"D":"text 2"
			}
			}

Repeated Elements	<A>
	<B>text 1</B>
	<B>text 2</B>
	<C>
	<D>text 3</D>
	</C>
	<C>
	<D>text 4</D>
	</C>
	<A>
		Repeated elements use the element name as a JSON array name and the values in the array as the text or child elements of the parent element.	"A":{
			"B":["text 1",
			"text 2"],
			"C":["D":"text 3",
			"D":"text 4"]
			}

Entities	<A>"Testing"</A>	If the entity must be escaped in JSON then it is preceded by a slash. Other characters will be translated directly.	"A":"\"Testing\""

Special Characters	<A>C:\\</A>	If a character must be escaped in JSON, it will be preceded by a slash	"A":"C:\\\\"
Comments	<!--Comment-->	These will not be converted to JSON	
Processing Instructions	<?xml version="1.0" encoding="UTF-8"?>	These will not be converted to JSON	

Production Rules for translating XML to JSON

Another way of expressing these rules is to use production rules for translating XML structures into JSON:

<element> ::= "<element name>":<element value>| "$":"<element text>"
<element value> ::= <element>|"<element text>"|<complex element>|<repeated element>|null
<complex element> ::= {<attribute list><element value>}
<repeated element> ::= [<element value>,<repeated element>|<element value>]
<attribute list> ::= <attribute>,<attribute list>|<attribute>
<attribute> ::= "@<attribute name>":"<attribute value>"
<attribute value> ::="<attribute value text>"

Appendix

JSON Definition

This link provides the complete syntax for JSON:

http://json.org/

Conversion between XML and JSON:

http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html

http://badgerfish.ning.com/

http://wiki.open311.org/index.php?title=JSON_and_XML_Conversion

http://www.bramstein.com/projects/xsltjson/

http://code.google.com/p/xml2json-xslt/

http://json-lib.sourceforge.net/index.html

We also tested the Altova XML-Spy conversion:
http://www.altova.com/xmlspy.html.

The conversion lost much of the XML structure in a round trip. It made attributes into elements and lost all processing instructions and comments.

JSON conversion to and from POJO

https://json-processing-spec.java.net/

http://www.javaworld.com/article/2074650/core-java/javaone-2012--jsr-353--java-api-for-json-processing.html

http://examples.javacodegeeks.com/enterprise-java/rest/resteasy/json-example-with-resteasy-jaxb-jettison/

Other XML organizations using JSON

OASIS

The following link lists OASIS Technical Committees that either currently feature JSON and/or REST in their charters, or are discussing REST or JSON:
https://www.oasis-open.org/resources/topics/rest-json

OASIS has also recently approved version 4.0 of the Open Data Protocol (OData) and the OData JSON Format. A press release on MarketWatch is here:

http://www.marketwatch.com/story/oasis-approves-odata-40-standards-for-an-open-programmable-web-2014-03-17

Languages that support JSON parsing

- C (jason-parser)
- awk (json.awk)
- C++ (a bunch, including JSONKit, JSON++ and libjson)
- C# (JSON for .net, JSONSharp, Manatee Json)
- Javascript (JSON, kson2.js, clarinet)
- Java (JSON Tools, google-gson, Argo, SOJO, XStream, Json-lib, jjson)
- Objective C (JSONKit, NSJSONSerialization, json-framework, ObjFW)
- Perl (CPAN, perl-JSON-SL)
- PHP (native in 5.2, Services_JSON, json)
- PL/SQL (pljson, Librarie-JSON)
- Python (standard library, simplejson, pyson, ultraison)
- Ruby (built-in)
- Visual Basic (VB-JSON, PW.JSON)