Paper 1660-2016

Reading JSON in SAS® Using Groovy

John Kennedy, Mesa Digital

ABSTRACT

JavaScript Object Notation (JSON) has quickly become the de-facto standard for data transfer on the Internet, due to an increase in web data and the usage of full-stack JavaScript. JSON has become dominant in the emerging technologies of the web today, such as the Internet of Things and the mobile cloud. JSON offers a light and flexible format for data transfer, and can be processed directly from JavaScript, without the need for an external parser. SAS® has no native implementation for JSON reading; however, the addition of the Groovy Procedure in SAS 9.3 allows for execution of Java code from within SAS, allowing for JSON data to be read into a SAS dataset. This paper demonstrates the method for parsing JSON into datasets with Groovy and the XML Libname Engine, all within Base SAS.

JSON 101

JSON is a lightweight data format for transmitting data objects, centered on name-value pairs. JSON was created by Douglas Crockford, author of *Javascript: The Good Parts*, around the turn of the century to facilitate browser to server communication (Crockford). JSON's structure reminds one visually of XML, with tags marking values and also with the prominent nests featured below. JSON supports strings, numbers, arrays, booleans, nulls, and objects, separated by commas. Both JSON and XML have the same attribute-value structure, albeit with different delimiters; however, JSON supports more types, such as an explicit array structure, compared to XML, which would create an object where each attribute in the object is

COMPARING JSON TO XML

```
{
 "firstName": "John",
 "lastName": "Smith",
 "isAlive": true,
 "age": 25,
 "address": {
   "streetAddress": "21 2nd Street",
   "city": "New York",
   "state": "NY",
   "postalCode": "10021-3100"
 },
 "phoneNumbers": [
   {
     "type": "home",
     "number": "212 555-1234"
   },
   {
     "type": "office",
     "number": "646 555-4567"
   }
 ],
 "children": [ ],
 "spouse": null
}
```

Code 1.0 – test.json

```
<xml>
<firstName>John</firstName>
<lastName>Smith</lastName>
<isAlive>true</isAlive>
```

```
<age>25</age>
<address>
       <streetAddress>21 2nd Street</streetAddress>
       <city>NewYork</city>
       <state>NY</state>
       <postalCode>10021-3100</postalCode>
</address>
<spouse>null</spouse>
<phoneNumbers>
       <number>212 555-1234</number>
       <type>home</type></phoneNumbers>
<phoneNumbers>
       <number>646-555 4567</number>
       <type>office</type></phoneNumbers>
<children></children>
</xml>
```

Code 1.1 – test.xml

COMMON USES AND INCREASE IN POPULARITY OF JSON

JSON and JavaScript are linked explicitly and implicitly, and the increase in web applications and thus JavaScript use, foreshadows a rise in JSON use. As users become more dependent and demand more from the browser-server model; looking to the web for business, communications, and multimedia solutions, the use of JSON as a lightweight and efficient file format should escalate. Also, with the recent rise in popularity of JavaScript back end technology (node.js, init.js, backbone.js), more and more groups use full stack JavaScript in their web applications - having node.js on the backend (server side) and JavaScript on the frontend (browser side). Because of JSON's compatibility with JavaScript, those full-stack applications use JSON to transmit data and metadata. Examples of full stack JSON users include LinkedIn, PayPal, Flickr, and Groupon.

POSSIBLE USES OF SAS TO PARSE JSON AND THEIR FAULTS

Though JSON is rising in popularity and use, SAS has no native implementation for JSON reading. A PROC JSON does exist, but can only output JSON data, not read it. Base SAS comes out-of-the-box with many powerful string processing features, which immediately come to mind for parsing raw JSON data. For a single list of JSON, a simple input in the DATA step can read in JSON, as seen in the example below.

```
data temp;
    length attribute value $80;
    infile datalines;
      input @;
    if _infile_ ne: '{' and _infile_ ne: '}';
      input attribute: $quote. x $ value: $quote.;
     put '<' attribute '>' value '</' attribute '>';

datalines;
{
    "name" : "John",
    "position" : "intern"
}
;
run;
```
Code 1.2 – basic_json.sas

```
<name>John </name>
<position>intern </position>
```

Output 1.0 – basic_json.sas

AN INTRODUCTION TO GROOVY

Groovy, initially developed by James Strachan in 2003, is a language built on top of the Java platform, compatible with Java and Java modules, such as the JSON jar files used later. Groovy Procedure is the SAS application for accessing the Groovy virtual machine, allowing access of Groovy and implementation of Groovy code from within a SAS program without a need for special configuration.

HELLO WORLD

```
proc groovy;
       submit;
              println('hello world')
       endsubmit;
```

```
hello world!
```

Output 2.0 – hello_groovy.sas

Code 2.0 – hello_groovy.sas

Groovy in SAS is sandwiched between a submit and an endsubmit, unique to a few procedures such as Groovy and Proc Lua. Although only one line of Groovy exists in hello_groovy.sas, the Java influence is clear from the 'println()' function. Also note the lack of a 'run' statement after the procedure. The Groovy code will execute after the 'endsubmit', no need for a run.

READING JSON WITH PROC GROOVY

INSTALLING THE JSON MODULES

In this text, the Java modules for JSON created and supported by the JSON organization will be used. Although a pure Groovy JSON module exists, the compatibility between Groovy and Java allows for the use of the Java modules, which have much more documentation and a much larger user community. The modules currently reside at Douglas Crockford's Github page (link in the Resources section)

After the modules have been downloaded, the Proc Groovy statement needs to find the new modules, as shown below.

```
       proc groovy classpath="java-json.jar";
              /* two ways to point to jar file */
                            submit;
                                   add classpath="java-json.jar"
```

Code 3.0 (fragment)

READING THE JSON AND CONVERTING TO XML

*Code 1.0 (test.json) will be used as input

```
       proc groovy;
       /* showing proc groovy where my json java files are */
              add classpath = "java-json.jar";
              submit; /* starting groovy */


                     import org.json.JSONObject;
```

```
                    import org.json.XML;

                    /* loading json from file */
                    def json_string = new File("test.json").text;

                    /* converting json to xml */
                    def json = new JSONObject(json_string);
                    def xml = XML.toString(json);

                    /* saving to file */
                    def out = new PrintWriter("test.xml");

                    /* adding xml headers */
                    out.println("<xml>", xml, "</xml>");
                    out.close();

                    endsubmit;
        quit;
```

Code 3.1 – xml_generator.sas

The code is quite simple thanks to the org.json modules. Twice as many lines are handle file input/output rather than JSON/XML conversion.

For creating a dataset from the JSON, creating a new XML file and then reading it from SAS is necessary, as it circumvents going through the Macro layer and passing the data directly to SAS through Java. After the Groovy step, a dataset can easily be created from the JSON data using the SAS XML Engine.

CREATING A DATASET FROM XML

```
        libname x xml "test.xml";
        libname dat "test.dat";

        /* creating dataset of phone numbers for person */
        data dat.phoneNumbers;
                set x.phoneNumbers;
        run;

        proc print data=dat.phoneNumbers;

        run;
```

Code 3.2 – dataset_from_xml.sas

CONCLUSION

SAS has no native, explicit JSON reader. The need for a JSON reader is amplified by the rapid increase in the use of JSON as a file format with the increased use of web applications. Groovy Procedure can be used along with JSON Java modules to read JSON data into SAS datasets by converting the JSON to XML.

ACKNOWLEDGEMENTS

Thanks to Jeff Bailey, Darren Key, Rick Langston, Salman Maher and Bill Oliver

RESOURCES

- **Code Samples (github)**
- **JSON Java modules**
- **SAS XML Libname Documentation**
- **SAS Proc Groovy Documentation**

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John Kennedy
john@mesa.digital
mesa.digital