Paper 296-2013

Efficient extraction of JSON information in SAS® using the SCANOVER function

Murphy Choy, SIS, Singapore Management University

Kyong Jin Shim, SIS, Singapore Management University

ABSTRACT

JSON, otherwise known as JavaScript Object Notation, is a popular data interchange format which provides a human readable format. It is language independent and can be read easily in a variety of computer languages. With the rise of Twitter and other unstructured data, there has been a move to incorporate such data as a way of disseminating information. Twitter currently provides a simple API for users to extract tweets using JSON format. While SAS does not currently have a direct way of reading JSON, the SCANOVER function in SAS data step provides user with a simple and effective approach to getting JSON information into SAS datasets. In this paper, we will illustrate two examples using this technique.

INTRODUCTION

In the early days, there were many different ways to store data and allow the data to be read by different machines. Data were stored in the form of CSV and FWD text files which can be easily read by different softwares. However, there were no common standards for many of the early formats resulting in difficulties in sharing the data. Due to this, several standards were implemented which included the RIFC 4180 that was specifically designed for CSV files. The arrival of standards allowed for the progress of the inter-readability between softwares. Even with standards, there were some issues due to the use of commas in certain countries for numeric values formatting. More efficient formatting was needed. There were many other formats such as Tab-delimited files and DIF (Data interchange format). These formats were considered to be flat file format which are more designed for data bases.

With the advert of technologies, there were further moves to develop data formats which are interchangeable between different OS and softwares. The most important one that was developed was the XML format. The XML format has several features which make it very attractive to users. The format is light-weight, simple, generalizable and can be adopted by web applications. Originally envisioned as a document format, it rapidly evolved into a data format favored by web based applications. XML also introduced the ability to incorporate hierarchical structures. Even with its widespread use, XML has been criticised on several features. To overcome this, JSON has been proposed as an alternative format.

JSON

JSON which stands for JavaScript Object Notation is a lightweight data-interchange format that is easy for humans to read and write as well as easy for machines to parse and generate. It is developed on a subset of the JavaScript Programming Language. JSON is a format that is completely language independent but uses conventions that are familiar to the C-group of languages.

JSON is built on two structures:

- A collection of name/value pairs which is realized as an object, record, dictionary, hash table, keyed list, or associative array in other languages.
- An ordered list of values realized as an array or sequence.

These are universal data structures which all modern programming languages support them in one form or another. It also makes sense that a data format that is interchangeable with programming languages also be based on these structures.

JSON's data basic types are:

- Number
- String

- Boolean
- Array
- Object
- null

Non-significant white space may be added freely around the "structural characters" such the brackets, colon and comma. They do not affect the structure of the data.

In JSON, an object begins with left brace and ends with right brace. Each name is followed by colon and the name/value pairs are separated by comma. An example is shown below.

```
{
   "firstName": "John",
   "lastName" : "Smith",
   "age"      : 25
}
```

An array is an ordered collection of values which begins with left bracket and ends with right bracket and are separated by comma.

```
 "phoneNumber":
   [
      {
       "type"  : "home",
       "number": "212 555-1234"
      },
      {
       "type"  : "fax",
       "number": "646 555-4567"
      }
   ]
```

JSON's data structure is similar to XML. However, unlike XMLs, JSON schemas are more uncommon in comparison to XML schemas being used. Due to manual and human nterventions needed to construct the schemas for JSON, JSON schemas are relatively less useful for data read ins. This created a problem for SAS which does not provide a default read in mechanism with PROC Import, filename XML engine and data step. To overcome this deficiency, the author proposes the use of SCANOVER options in SAS data steps for extracting the information from JSON

SCANOVER FUNCTION IN DATA STEP

The SCANOVER function in SAS is a very interesting and specialized function in SAS data step which looks for the location of a particular string of words and extract those information. The amount of information to be extracted must be specified before the extraction works. Below is an example extracted from SAS (SAS, 2010).

```
filename phonebk host-specific-path;
data _null_;
  file phonebk;
  input line $80.;
  put line;
  datalines;
    Jenny's Phone Book
    Jim Johanson phone: 619-555-9340
       Jim wants a scarf for the holidays.
    Jane Jovalley phone: (213) 555-4820
       Jane started growing cabbage in her garden.
       Her dog's name is Juniper.
    J.R. Hauptman phone: (49)12 34-56 78-90
       J.R. is my brother.
   ;
run;
```

In the text which contains information, we are trying to extract the phone numbers which is preceded by phone: tags. To extract those information, one can use the following.

```
data _null_;
   infile phonebk truncover scanover;
   input @'phone:' phone $32.;
   put phone=;
run;
```

Notice the use of the @'phone:' construct. The construct points to the location of the data which is complemented by the subsequent input statement variable construct for the length of the data to be extracted. The combination allows the extraction of the information. Using the same concept, we will hereby demonstrate how to use programming construct in JSON.

APPLICATION IN READING REGULAR JSON

Regular JSON files are files which have repetitive constructs and fields. These files are the best files for the SCANOVER approach as the regularity allows for proper read in of data. An example of a regular well-formatted JSON file is shown below (Twitter, 2012) which is extracted from Twitter.

```
{
 "completed_in":0.031,
 "max_id":122078461840982016,
 "max_id_str":"122078461840982016",
 "next_page":"?page=2&max_id=122078461840982016&q=blue%20angels&rpp=5",
 "page":1,
 "query":"blue+angels",
 "refresh_url":"?since_id=122078461840982016&q=blue%20angels",
 "results":[
  {
   "created_at":"Thu, 06 Oct 2011 19:36:17 +0000",
   "entities":{
    "urls":[
     {
      "url":"http://t.co/L9JXJ2ee",
      "expanded_url":"http://bit.ly/q9fyz9",
      "display_url":"bit.ly/q9fyz9",
      "indices":[
       37,
       57
      ]
     }
    ]
   },
   "from_user":"SFist",
   "from_user_id":14093707,
   "from_user_id_str":"14093707",
   "geo":null,
   "id":122032448266698752,
   "id_str":"122032448266698752",
   "iso_language_code":"en",
   "metadata":{
```

```
33       "recent_retweets":3,
34       "result_type":"popular"
35     },
36     "profile_image_url":"http://a3.twimg.com/profile_images/51584619/SFist07_normal.jpg",
37     "source":"&lt;a href=&quot;http://Twitter.com/tweetbutton&quot; rel=&quot;nofollow&quot;&gt;Tweet
   Button&lt;/a&gt;",
38     "text":"Reminder: Blue Angels practice today http://t.co/L9JXJ2ee",
39     "to_user_id":null,
40     "to_user_id_str":null
41   },
42   {
43     "created_at":"Thu, 06 Oct 2011 19:41:12 +0000",
44     "entities":{
45
46     },
47     "from_user":"masters212",
48     "from_user_id":2242041,
49     "from_user_id_str":"2242041",
50     "geo":null,
51     "id":122033683212419072,
52     "id_str":"122033683212419072",
53     "iso_language_code":"en",
54     "metadata":{
55       "recent_retweets":1,
56       "result_type":"popular"
57     },
58     "profile_image_url":"http://a3.twimg.com/profile_images/488532540/rachel25final_normal.jpg",
59     "source":"&lt;a href=&quot;http://Twitter.com/&quot;&gt;web&lt;/a&gt;",
60     "text":"Starting to hear Blue Angels... Not such angels with all of the noise and carbon pollution.",
61     "to_user_id":null,
62     "to_user_id_str":null
```


From the file, the JSON file has regular sections which are similar to the one in the example. Given such regularity in the JSON file, one can use the codes below to extract the information.

```
filename data 'c:\temp\obama.json';

data datatest;
    infile data lrecl = 32000 truncover scanover;
    input @'"text": "' text $255. @'"from_user": "' from_user $255. @'"created_at":
    "' created_at $255. ;
    text = substr(text,1,index(text,'",')-2);
    from_user = substr( from_user,1,index( from_user,'",')-2);
    created_at = substr(created_at,1,index(created_at,'",')-2);
run;
```

The codes work well and produce a regular table that can then be analyzed. However, regular JSON files are uncommon. The approach above can then be modified to handle irregular JSON.

APPLICATION IN READING IRREGULAR JSON

Irregular JSON files are those that have fields which are inconsistent between different files. The differences can be with respect to the structure which includes the hierarchy of the file as well as the fields. Below is an example extracted from an online source.

{"total": 1,
"page": 1,
"results":
[{"name": "Speckly",
"permalink": "speckly",
"namespace": "company",
"overview": "Speckly's mission is to make the process of finding media delivered through [BitTorrent](http://www.crunchbase.com/company/bittorrent) easy for everyone. Part of that equation is making BitTorrent searches relevant, fast, and easy to understand. ",
"image":
{"available_sizes":
[[[150,
61],
"assets/images/resized/0002/2217/22217v1-max-150x150.png"],
[[156,
64],
"assets/images/resized/0002/2217/22217v1-max-250x250.png"],
[[156,
64],
"assets/images/resized/0002/2217/22217v1-max-450x450.png"]],
"attribution": null}}]}

We can see that most of the information is not repeated. This makes it very difficult for extraction of information. Using the same principles demonstrated before, we can always extract fields which are unique and critical to us. To that end, one can use the macro below to achieve this.

```
%macro flexdata(data,comm,output);

data &output;
    infile &data lrecl = 255 truncover scanover;
    input &comm;
    &comm = substr(&comm,1,index(&comm,'",')-2);
run;

%mend;
```

Essentially, the macro allows us to extract specific words and information tagged from the JSON file. If the information is unique to that entity, then the information extracted can be tied to the entity and a regular table of information can be built using singular word extraction and linked them up. This allows us to build a database for analysis.

CONCLUSION

SAS datastep with SCANOVER function provides excellent utilities for us to work on JSON data.

REFERENCES

SAS(R) 9.2 Language Reference: Dictionary, Fourth Edition, 2011

Twitter API References. Get Search Documentation, 2012.

CONTACT INFORMATION

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

Name: Murphy Choy
E-mail: goladin@gmail.com

Name: Kyong Jin Shim
E-mail: kjshim@smu.edu.sg