Paper SAS380-2017

The JSON LIBNAME Engine: Real-World Applications Using Open APIs

Michael Drutar and Eric Thies, SAS Institute Inc.

ABSTRACT

JSON is quickly becoming the industry standard for data interchanges, especially in supporting REST APIs. But until now, importing JSON content into SAS® software and leveraging it in SAS has required significant custom code. Developing that code can be laborious, requiring transcoding, manual text parsing, and creating handlers for unexpected structure changes. Fortunately, the new JSON LIBNAME engine (in the fourth maintenance release for SAS® 9.4 and later) delivers a robust, efficient method for importing JSON content into SAS data structures. This paper demonstrates several real-world examples of the JSON LIBNAME engine using open data APIs. The first example contrasts the traditional custom code and the JSON LIBNAME engine approach using big data from the United Nations Comtrade Database. The two approaches are compared in terms of complexity of code, time to execute, and the resulting data structures. The same method is applied to data from Google Maps and the US Census Bureau's APIs. Finally, to demonstrate the ability of the JSON LIBNAME engine to handle unexpected changes to a JSON data structure, we use the SAS JSON procedure to write a JSON file and then simulate changes to that structure to show how one JSON LIBNAME engine process can easily adjust the import to handle those changes.

INTRODUCTION

When the JSON data format first appeared, the SAS programmer's approach to extracting its data was a tedious and laborious task. The steps to do so usually included the following:

1. Download the JSON file from the Internet.
2. Import the JSON file as a text file.
3. Visually inspect the JSON file to determine what information needs to be extracted.
4. Write text parsing code to extract the needed data from the file.

Although the four steps above are certainly achievable, they can be time-consuming. Moreover, the four steps can be unreliable. If a data structure changes within the JSON file, the process could fail.

The good news is that the new JSON LIBNAME engine simplifies these steps. In order to demonstrate the advantages that the new LIBNAME engine offers, a demonstration will be provided that shows how the four steps described above can be used to import a JSON file. Later, the JSON LIBNAME engine will be used to import the same JSON file. The data source for this demonstration will be publicly available from the United Nations Comtrade Database API at https://comtrade.un.org/data/doc/api/. Specifically, a JSON file that contains the data for the commodity classification Harmonized System is at https://comtrade.un.org/data/cache/classificationHS.json.

PREVIOUS METHOD OF READING JSON FILES INTO SAS

The first step in the process is to download the JSON data file. This is best achieved by using PROC HTTP, which is documented at http://support.sas.com/documentation/cdl/en/proc/69850/HTML/default/viewer.htm#n0bdg5vmrpyi7jn1pbgbje2atoov.htm.

First, you need to use the FILENAME statement to declare a filename to indicate where the downloaded JSON file will be placed. In the example below we declare the file named rept and associate it with the file **C:\temp\classificationHS.JSON:**

```
filename rept "C:\UNComtrade\classificationHS.JSON";
```

After the FILENAME statement has been submitted, the file itself can be downloaded via PROC HTTP.

```
proc http
      method = "get"
      url =
"https://comtrade.un.org/data/cache/classificationHS.json"
      out = rept;
```

```
run;
enddata;
run;
quit;
```

After this code is run, the file named classificationHS.JSON is created in the target folder **C:\UNComtrade**.

Step 2 of the process is to import the downloaded file as a text file. Although there are several methods of importing text files into SAS, the example below leverages the INFILE statement within a DATA step. One advantage of doing so is the ability to use the FILENAME statement that was already declared in Step 1. The code for Step 2 is below:

```
data work.one;
  infile rept pad;
  input @1 char1 $4000.;
run;
```

After running this code, the data set named WORK.ONE is created by reading the file that is associated with the file named rept (in this case **C:\temp\classificationHS.JSON**) with a character length of 4000. This is one of the disadvantages of this old method of extracting JSON data. The programmer has assumed that every line of text in the JSON file is less than 4000 characters long. Although the current JSON file might conform to that length, the United Nations Comtrade team might update the source JSON file with new data strings that are longer than 4000. If this happens, it could cause truncations within the imported string char1.

The next step is to visually inspect the one data set within a SAS data viewer. From there, records that should be imported and those that should not be imported can be identified. The image below shows the records that should be retained (enclosed in a red box).

	char1
9	"text": "ALL - All HS commodities",
10	"parent": "#"
11	},
12	{"id": "TOTAL", "text": "TOTAL - Total of all HS commodities", "parent": "#" },
13	{"id": "AG2", "text": "AG2 - All 2-digit HS commodities", "parent": "#" },
14	{"id": "AG4", "text": "AG4 - All 4-digit HS commodities", "parent": "#" },
15	{"id": "AG6", "text": "AG6 - All 6-digit HS commodities", "parent": "#" },
16	{"id": "01", "text": "01 - Live animals", "parent": "TOTAL" },
17	{"id": "0101", "text": "0101 - Live horses, asses, mules and hinnies", "parent": "01" },
18	{"id": "010110", "text": "010110 - Live horses/asses/mules/hinnies: pure-bred breeding animals", "parent": "0101" },
19	{"id": "010111", "text": "010111 - Horses, live pure-bred breeding", "parent": "0101" },
20	{"id": "010119", "text": "010119 - Horses, live except pure-bred breeding", "parent": "0101" },
21	{"id": "010120", "text": "010120 - Asses, mules and hinnies, live", "parent": "0101" },

Figure 1. WORK.ONE Data View

There are various methods that could be used to uniquely identify and flag the records that we want to keep. For this example, records that contain the string ”text“,will be retained.

From these retained records, the next step is to parse the text and extract the needed data from the string char1. For this paper, the SCAN function will be used to look for certain occurrences of double quotation marks. From a visual inspection, you can see that the needed text is between the eighth and ninth double quotation marks. For this process, the SUBSTR function will be used with the SCAN function to locate the positions for the SUBSTR parameters. The code to do this is below:

```
data two;
set one;
```

```
/*KEEP ONLY NEEDED RECORDS*/
where index(char1,'text"') > 0;

/*FIND THE START OF THE NEEDED STRING*/
  call scan(char1,8,pos,len,'"');
  start_string=pos;

/*FIND THE END OF THE NEEDED STRING*/
  call scan(char1,9,pos,len,'"');
  end_string=pos-1;

/*USE THE FOUND 'START_STRING' AND 'END_STRING' VALUES TO FEED THE
PARAMETERS FOR THE SUBSTR FUNCTION*/
 commodity_description= substr(char1,start_string,end_string-
start_string);

/*DROP UNNEEDED VARIABLES*/
  drop len pos;

run;
```

The resulting data is displayed below:

Total rows: 7656 Total columns: 4

	char1	start_str...	end_string	commodity_description
30	{"id":"0105","text":"0105 - Live	26	81	0105 - Live poultry, domestic fowls, ducks, geese, etc.
31	{"id":"010511","text":"010511	28	69	010511 - Fowls, live domestic < 185 grams
32	{"id":"010512","text":"010512	28	44	010512 - Turkeys
33	{"id":"010513","text":"010513	28	210	010513 - Live animals // Live poultry, that is to say, fowls of the sp
34	{"id":"010514","text":"010514	28	210	010514 - Live animals // Live poultry, that is to say, fowls of the sp
35	{"id":"010515","text":"010515	28	217	010515 - Live animals // Live poultry, that is to say, fowls of the sp

Figure 2. WORK.ONE Data View

At this point the SAS programmer has successfully extracted the necessary data from the JSON file and placed it under the column named **commodity_description**. Although this method that is being used to parse the text might appear to work just fine, it's not very reliable. Principally, the assumption is that the text between the eighth and ninth double quotation marks will always be the expected content to be extracted. The above code might not work correctly if the data structure of the JSON file changes. For example, what if the JSON file is updated with a new column that is placed between the string named id and the string named text? This new field might cause the SUBSTR function to extract the wrong text from the JSON file. So the code above, especially Step 4, is essentially hardcoded.

JSON LIBNAME ENGINE

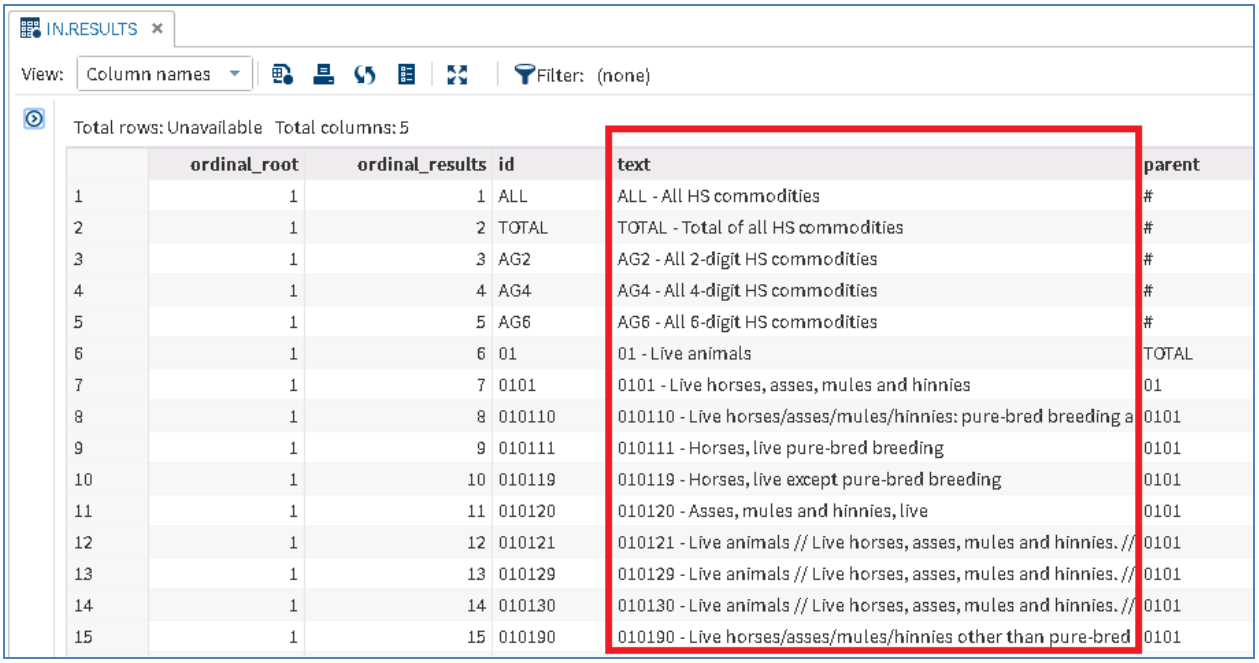
The new JSON LIBNAME engine is one of the most exciting new features of SAS. Its main functionality is to associate a SAS libref with a JSON document. Using the JSON LIBNAME engine, SAS users can access JSON files (either locally or on the web) via the method with which they access most other data sources: a LIBNAME statement. For details about the JSON LIBNAME engine, see the documentation at http://support.sas.com/documentation/cdl/en/lestmtsref/69738/HTML/default/viewer.htm#n1jfdetszx99ban1rl4zll6tej7j.htm.

The JSON LIBNAME engine simplifies the process that was described in the previous section to only a few lines of code:

```
filename in url "https://comtrade.un.org/data/cache/classificationHS.json"
debug;
libname in json;
```

The code above completely replaces all the steps that were described in the previous section. First, the FILENAME statement is similar to the FILENAME statement from the original example but it includes the URL option. See the documentation at http://support.sas.com/documentation/cdl/en/lestmtsref/69738/HTML/default/viewer.htm#p103pi2vrzn6qhn1e8alrs01jrb7.htm. This is one of the best features of the JSON LIBNAME engine, which is to read a JSON file directly from the Internet. Therefore, it is no longer necessary to download the online file to local disk space.

The next step of leveraging the JSON LIBNAME engine is to simply assign the libref. Executing the LIBNAME statement will associate the SAS libref named in with the JSON file. When run, the contents of the library that is referenced by the libref named in can be viewed. In this example, three data sets are created: ALLDATA, RESULTS, and ROOT. The IN.RESULTS data set contains the information that was extracted in the previous example:



	ordinal_root	ordinal_results	id	text	parent
1	1	1	ALL	ALL - All HS commodities	#
2	1	2	TOTAL	TOTAL - Total of all HS commodities	#
3	1	3	AG2	AG2 - All 2-digit HS commodities	#
4	1	4	AG4	AG4 - All 4-digit HS commodities	#
5	1	5	AG6	AG6 - All 6-digit HS commodities	#
6	1	6	01	01 - Live animals	TOTAL
7	1	7	0101	0101 - Live horses, asses, mules and hinnies	01
8	1	8	010110	010110 - Live horses/asses/mules/hinnies: pure-bred breeding a	0101
9	1	9	010111	010111 - Horses, live pure-bred breeding	0101
10	1	10	010119	010119 - Horses, live except pure-bred breeding	0101
11	1	11	010120	010120 - Asses, mules and hinnies, live	0101
12	1	12	010121	010121 - Live animals // Live horses, asses, mules and hinnies. //	0101
13	1	13	010129	010129 - Live animals // Live horses, asses, mules and hinnies. //	0101
14	1	14	010130	010130 - Live animals // Live horses, asses, mules and hinnies. //	0101
15	1	15	010190	010190 - Live horses/asses/mules/hinnies other than pure-bred	0101

Figure 3. IN.RESULTS Data View

CHANGING JSON DATA STRUCTURES

So far, the JSON LIBNAME engine has simplified our manual process of downloading and parsing code. However, the real value of the JSON LIBNAME engine is the ability to successfully read a changing JSON data structure.

For the purposes of this paper, this will be demonstrated by simulating a data structure change to the JSON file that we have been using as an example. First, a new variable named ID_text is added to the IN.RESULTS table that was created in the previous example. Then, PROC JSON will be used to create as output the SAS data set (from the previous example) to a JSON file. For details about PROC JSON, see https://support.sas.com/documentation/cdl/en/proc/69850/HTML/default/viewer.htm#p0ie4bw6967jg6n1iu629d40f0by.htm. The code to do this is below:

```
proc sql;
  create table results_new_columns
    as select ID, monotonic() as ID_text, Text
      from in.results
  ;
quit;

proc json out=" C:\UNComtrade\OUTPUT_JSON_new_columns.json" nosastags pretty
;
 write values "data";
 write open array;
   export results_new_columns;
```

```
      write close;
run;
```

As before, we will use both methods of extracting the data on this new JSON file.

METHOD 1: PARSING THE JSON FILE

```
filename rept "C:\UNComtrade\OUTPUT_JSON_new_columns.json";
data work.one;
  infile rept pad;
  input @1 char1 $4000.;
run;

data two; set one;

/*KEEP ONLY NEEDED RECORDS*/
where index(char1,'text"') > 0;

/*FIND THE START OF THE NEEDED STRING*/
  call scan(char1,3,pos,len,'"');
  start_string=pos;

/*FIND THE END OF THE NEEDED STRING*/
  call scan(char1,4,pos,len,'"');
  end_string=pos-1;

/*USE THE FOUND 'START_STRING' AND 'END_STRING' VALUES TO FEED THE PARAMETERS
FOR THE SUBSTR FUNCTION*/
 commodity_description= substr(char1,start_string,end_string-start_string);

/*DROP UNNEEDED VARIBLES*/
  drop len pos;

run;
```

Here is the output of the parsing method:

Total rows: 15312 Total columns: 4

	char1	start_string	end_string	commodity_description
1	"ID_text": 1,	0	-1	
2	"text": "ALL - All HS commodities"	10	34	ALL - All HS commodities
3	"ID_text": 2,	0	-1	
4	"text": "TOTAL - Total of all HS comm	10	45	TOTAL - Total of all HS commodities
5	"ID_text": 3,	0	-1	
6	"text": "AG2 - All 2-digit HS commod	10	42	AG2 - All 2-digit HS commodities
7	"ID_text": 4,	0	-1	

Figure 4. Output of the Parsing Method

METHOD 2: JSON LIBNAME ENGINE

```
filename rept "C:\UNComtrade\OUTPUT_JSON_new_columns.json";
libname rept json;
```

Here is the output of the JSON LIBNAME engine:

Total rows: Unavailable Total columns: 5 Rows 1-100

	ordinal_r...	ordinal_d...	id	ID_text	text
8	1	8	010110	8	010110 - Live horses/asses/mules/hinnies: pure-bred breeding animals
9	1	9	010111	9	010111 - Horses, live pure-bred breeding
10	1	10	010119	10	010119 - Horses, live except pure-bred breeding
11	1	11	010120	11	010120 - Asses, mules and hinnies, live
12	1	12	010121	12	010121 - Live animals // Live horses, asses, mules and hinnies. // - Horses : // -- Pure-b
13	1	13	010129	13	010129 - Live animals // Live horses, asses, mules and hinnies. // - Horses : // -- Other
14	1	14	010130	14	010130 - Live animals // Live horses, asses, mules and hinnies. // -Asses

Figure 5. Output from the JSON LIBNAME ENGINE

Notice how the parsing method is not returning the correct results. Because the new column is titled **ID_text,** the parsing code thinks that records containing the new column are valid for processing. As a result, incorrect records are being returned to our output data set. However, the JSON LIBNAME engine successfully imports the additional column along with the original columns. Therefore, the JSON LIBNAME engine provides a simpler approach to extracting data from JSON data structures, and is also a superior approach to using a JSON file as a reliable data source.

ADDITIONAL EXAMPLES

Perhaps one of the most exciting aspects of the JSON LIBNAME engine is the potential for new data sources for the SAS programmer. As JSON is quickly becoming the industry standard for data interchanges (especially in supporting REST APIs), the JSON LIBNAME engine empowers the SAS programmer to start leveraging data from JSON sources as new APIs become available. Below are two examples that leverage the JSON LIBNAME engine with JSON data via APIs that are publicly available.

EXAMPLE 1: REVERSE GEOCODING

One of the most popular public APIs today is Google Maps. It provides a wide variety of fantastic services that return information in JSON format. For documentation about the API (including usage limits), see https://developers.google.com/maps/documentation/. For example, for this paper, we will use the Google Maps API for reverse geocoding. The user can submit a set of longitude and latitude coordinates as input, and receive as output an address (or series of addresses) that are closest to those coordinates. Since SAS Global Form is being held at Walt Disney World this year, the coordinates for the Cinderella Castle are provided as the input for this example: 28.4195°N 81.5812°W.

According to the documentation, the web request for these coordinates would be:

https://maps.googleapis.com/maps/api/geocode/json?latlng=28.4195,-81.5812

The response from the API that is viewed in a browser looks like this:

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "Cinderella Castle",
               "short_name" : "Cinderella Castle",
               "types" : [ "premise" ]
            },
            {
               "long_name" : "Orlando",
               "short_name" : "Orlando",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Orange County",
               "short_name" : "Orange County",
               "types" : [ "administrative_area_level_2", "political" ]
            },
            {
               "long_name" : "Florida",
               "short_name" : "FL",
               "types" : [ "administrative_area_level_1", "political" ]
            },
            {
               "long_name" : "United States",
               "short_name" : "US",
               "types" : [ "country", "political" ]
            },
            {
```


Figure 6. Returned Output from the API Call

Next, the JSON LIBNAME engine is used to read the preceding JSON file:

```
filename in url
'https://maps.googleapis.com/maps/api/geocode/json?latlng=28.4195,-81.5812';
libname in json;
data results_out;
set in.results;
run;
```

The data set named work.results_out that is generated as output is shown below:

Total rows: 8 Total columns: 4

	ordinal_...	ordinal_r... ▲	formatted_address	place_id
1	1	1	Cinderella Castle, Orlando, FL 32836, USA	ChIJA1P2o5iA54gRb6Sqp8-jhnU
2	1	2	Fantasyland, Orlando, FL 32836, USA	ChIJ8SgvpJiA54gRatZvDzPv9Vc
3	1	3	Bay Lake, FL, USA	ChIJE2Qhhjh-3YgRssMv4oAliTY
4	1	4	Orlando, FL 32836, USA	ChIJ_0fG9gaA54gR2anJ4lSJJ4k
5	1	5	Orange County, FL, USA	ChIJLUsuoVaY54gRS_Ma3ZvbwUQ
6	1	6	Orlando Metropolitan Area, FL, USA	ChIJnZnf_xd954gRqQz4C7QedMg
7	1	7	Florida, USA	ChIJvypWkWV2wYgR0E7HW9MTLvc
8	1	8	United States	ChIJCzYy5IS16lQRQrfeQ5K5Oxw

Figure 7. Data View of the Data Set Named work.results_out

The JSON LIBNAME engine returns several data sets from this call to the API:

Figure 8. View of the Data Sets within the SAS Library

Each of these data sets can be very useful. The time and effort to manually parse the API's returned JSON content to create each of these data sets would have been grueling. Moreover, Google might start to add more information to the JSON output for this call. This new information might be very useful. The JSON LIBNAME engine would not only successfully import the new data provided, but the SAS programmer could put it to use immediately.

EXAMPLE 2: US CENSUS

The JSON LIBNAME engine can also help to find information about other JSON files within an API itself. An example of this is provided by the US Census API located at http://www.census.gov/data/developers/updates/new-discovery-tool.html. An interesting feature of this API is that there are several descriptor JSON files on the site. These JSON files can be used to discover all data sets within 2010.

For example, if we use the JSON LIBNAME engine to read one of the descriptor data sets, then the output will contain a data set named DATASET in the library named descr:

```
filename descr url "http://api.census.gov/data/2010.json";
libname descr json;
```

Total rows: 6 Total columns: 20 Rows

	c_vintage	c_geographyLink	c_examplesLink
1	2010	http://api.census.gov/data/2010/acs5/geography.json	http://api.census.gov/data/2010/acs5/examples.json
2	2010	http://api.census.gov/data/2010/cbp/geography.json	http://api.census.gov/data/2010/cbp/examples.json
3	2010	http://api.census.gov/data/2010/flows/geography.json	http://api.census.gov/data/2010/flows/examples.json
4	2010	http://api.census.gov/data/2010/nonemp/geography.json	http://api.census.gov/data/2010/nonemp/examples.json
5	2010	http://api.census.gov/data/2010/sf1/geography.json	http://api.census.gov/data/2010/sf1/examples.json
6	2010	http://api.census.gov/data/2010/surname/geography.json	http://api.census.gov/data/2010/surname/examples.json

Figure 9. View of the Data Set DATASET within the SAS DESCR Library

From the data set DESCR.DATASET, the variable c_examplesLink can be seen. The column offers several web

addresses to various example JSON files within the API. A simple macro can now be written to leverage the JSON LIBNAME engine and to extract data from each of these examples:

```
%macro get_examples(f_name,url);
filename &f_name. clear;
libname &f_name. clear;
filename &f_name. url "&url";
libname &f_name. json;
%mend get_examples;

%get_examples(cbp,http://api.census.gov/data/2010/cbp/examples.json)
%get_examples(flows,http://api.census.gov/data/2010/flows/examples.json)
%get_examples(nonemp,http://api.census.gov/data/2010/nonemp/examples.json)
%get_examples(sf1,http://api.census.gov/data/2010/sf1/examples.json)
%get_examples(surname,http://api.census.gov/data/2010/surname/examples.json)
```

The results of the execution of the preceding code are: each example JSON file is assigned to a LIBNAME statement and a library is assigned to that JSON structure that contains the data:

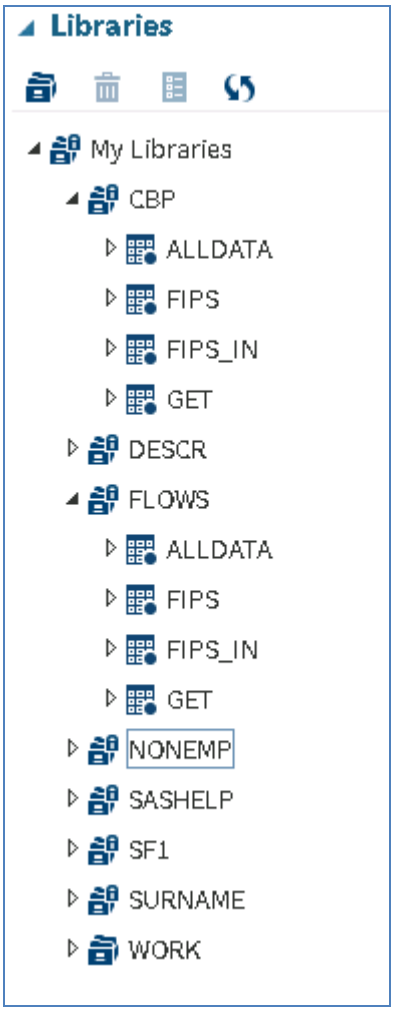


Figure 10. View of the Libraries within a SAS Session

The 13 lines of SAS code in this example, which include the JSON LIBNAME engine) have replaced what would have been potentially hours of text parsing along with what could have been hundreds of lines of SAS code. Moreover, these 13 lines of code could be enhanced more to have the macro dynamically called. So if new examples are added to the file named 2010.json, they would be automatically added to the macro execution. The robust JSON LIBNAME

engine empowers the SAS programmer to do all these things.

CONCLUSION

Although at one time the best method for extracting JSON data was via custom parsing code, the JSON LIBNAME engine makes this no longer necessary. Leveraging the LIBNAME engine not only opens up new sources of data to SAS programmers, but it also enables them to access that data much more reliably. As new JSON data sources become available via public and private APIs, the JSON LIBNAME engine places the SAS programmer in the best possible position to consume and leverage the data.

REFERENCES

SAS Institute Inc. 2017. SAS 9.4 Base SAS Procedures Guide. “HTTP Procedure” Cary, NC: SAS Institute Inc. Available http://support.sas.com/documentation/cdl/en/proc/69850/HTML/default/viewer.htm#n0bdg5vmrpyi7jn1pbgbje2atoov.htm.

SAS Institute Inc. 2017. SAS 9.4 Statements: Reference. “LIBNAME Statement, JSON Engine” Cary, NC: SAS Institute Inc. Available http://support.sas.com/documentation/cdl/en/lestmtsref/69738/HTML/default/viewer.htm#n1jfdetszx99ban1rl4zll6tej7j.htm.

SAS Institute Inc. 2017. SAS 9.4 Statements: Reference. “FILENAME Statement, URL Access Method” Cary, NC: SAS Institute Inc. Available http://support.sas.com/documentation/cdl/en/lestmtsref/69738/HTML/default/viewer.htm#p103pi2vrzn6qhn1e8alrs01jrb7.htm.

SAS Institute Inc. 2017. SAS 9.4 Procedures Guide. “JSON Procedure” Cary, NC: SAS Institute Inc. Available https://support.sas.com/documentation/cdl/en/proc/69850/HTML/default/viewer.htm#p0ie4bw6967jg6n1iu629d40f0by.htm.

The United Nations Comtrade Database. “The UN Comtrade data extraction API” New York, NY. Available https://comtrade.un.org/data/doc/api/.

Google Maps APIs. “Documentation” Mountain View, CA. Available https://developers.google.com/maps/documentation/.

United States Census Bureau. “New Discovery Tool” Washington, DC. Availablehttp://www.census.gov/data/developers/updates/new-discovery-tool.html.

CONTACT INFORMATION

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

Michael Drutar
Michael.Drutar@sas.com

Eric Thies
Eric.Thies@sas.com