SESUG Paper 007-2021

Copying Data Between SAS ® and JSON Files

Bruce Gilsen, Federal Reserve Board, Washington, DC

ABSTRACT

JavaScript Object Notation (JSON) is an open standard file format and data interchange format used for some of the same purposes as XML. More information about JSON is readily available on the internet.

Starting in SAS ® 9.4, you can copy SAS data sets to JSON files with PROC JSON. Starting in SAS 9.4TS1M4, you can copy JSON files to SAS data sets with the JSON engine.

This paper provides basic information and some relatively simple examples. It also discusses ongoing research on how to copy JSON files into SAS in an automated way.

INTRODUCTION

JavaScript Object Notation (JSON) is an open standard file format and data interchange format used for some of the same purposes as XML. More information about JSON is readily available on the internet.

Starting in SAS 9.4, you can copy SAS data sets to JSON files with PROC JSON. Starting in SAS 9.4TS1M4, you can copy JSON files to SAS data sets with the JSON engine.

Copying data from SAS to JSON with PROC JSON is relatively straightforward. Copying data from JSON to SAS can be much more complicated in some cases. To the extent possible, the examples in this paper copy JSON files to SAS in an automated way. Determining how to copy additional types of JSON files into SAS in an automated way is an area of ongoing research.

Reading JSONL files into SAS and the JSONPP DATA step function, which converts a single record JSON file to a "pretty" JSON file, are also discussed.

This paper provides basic information and some examples that use small amounts of data.

COPY A SAS DATA SET TO A JSON FILE: SIMPLE EXAMPLES

First, we'll copy a SAS data set to a JSON file with a common set of options. Then, we'll copy the same data set to JSON with different option values to show how the JSON file changes.

Data set ONE has the following values.

Obs	country	city	income	date
1	usa	chicago	100	20201001
2	usa	cleveland	200	20201101
3	canada	montreal	300	20201201

It was created with the following code. Note that DATE contains SAS date values.

```
data one;
  length country $10 city $10;
  input country $ city $ income date yymmdd8.;
  format date yymmddn8.;
```

```
  datalines;
usa chicago 100 20201001
usa cleveland 200 20201101
canada montreal 300 20201201
;run;
```

EXAMPLE 1. COPY SAS DATA SET TO A JSON FILE WITH PROC JSON

This code copies SAS data set ONE to JSON file /my/home/m1xxx00/test1.json.

```
proc json out="/my/home/m1xxx00/test1.json" pretty nosastags;
  export one;
run;
```

JSON file test1.json has the following contents.

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "date": "20201001"
  },
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "date": "20201101"
  },
  {
    "country": "canada",
    "city": "montreal",
    "income": 300,
    "date": "20201201"
  }
]
```

COMMON PROC JSON STATEMENT OPTIONS

PROC JSON statement options include the following. The first two options were used in the code above.

- PRETTY (versus NOPRETTY, the default) writes the JSON file in a human-readable format with indentation and multiple lines. NOPRETTY writes the entire JSON file as one long record.

- NOSASTAGS (versus SASTAGS, the default) suppresses SAS metadata at the top of the JSON file. For example, omitting NOSASTAGS from the above code adds the following text to the top of the JSON file, along with a closing curly bracket (}) at the bottom.

```
{
  "SASJSONExport": "1.0 PRETTY",
  "SASTableData+ONE":
```

- FMTCHARACTER/NOFMTCHARACTER, FMTDATETIME/NOFMTDATETIME, and FMTNUMERIC/NOFMTNUMERIC specify whether to apply already-assigned character, date/datetime/time, or numeric formats to variables written to the JSON file. Defaults are NOFMTCHARACTER, FMTDATETIME, and NOFMTNUMERIC.

- TRIMBLANKS (the default) versus NOTRIMBLANKS specifies that trailing blanks are removed from character data written to the JSON file.

To illustrate the options, here are examples that write data set ONE to a JSON file and the resulting JSON files.

EXAMPLE 2. INCLUDE SAS METADATA AT THE TOP OF THE JSON FILE

```
proc json out="/my/home/m1xxx00/test2.json" pretty;
  export one;
run;
```

```
{
  "SASJSONExport": "1.0 PRETTY",
  "SASTableData+TWO": [
    {
      "country": "usa",
      "city": "chicago",
      "income": 100,
      "date": "20201001"
    },
    {
      "country": "usa",
      "city": "cleveland",
      "income": 200,
      "date": "20201101"
    },
    {
```

```
      "country": "canada",
      "city": "montreal",
      "income": 300,
      "date": "20201201"
    }
  ]
}
```

EXAMPLE 3. WRITE THE JSON FILE AS ONE LONG RECORD

```
proc json out="/my/home/m1xxx00/test3.json" nosastags;
  export one;
run;
```

```
[{"country":"usa","city":"chicago","income":100,"date":"20201001"},{"country"
:"usa","city":"cleveland","income":200,"date":"20201101"},{"country":"canada"
,"city":"montreal","income":300,"date":"20201201"}]
```

EXAMPLE 4. DON'T USE ASSOCIATED SAS DATE FORMATS WHEN WRITING SAS DATE VALUES AND DON'T REMOVE TRAILING BLANKS FROM CHARACTER VARIABLES

In this case, the values of DATE are not formatted and character variables have trailing blanks.

```
proc json out="/my/home/m1xxx00/test4.json" pretty nosastags nofmtdt
  notrimblanks;
  export one;
run;
```

```
[
  {
    "country": "usa       ",
    "city": "chicago   ",
    "income": 100,
    "date": 22189
  },
  {
    "country": "usa       ",
    "city": "cleveland ",
    "income": 200,
    "date": 22220
  },
  {
```

```
    "country": "canada    ",
    "city": "montreal  ",
    "income": 300,
    "date": 22250
  }
]
```

The simple examples above could suffice for basic usage. The PROC JSON documentation contains much more detailed PROC JSON information, including how to control the containers in a JSON file and organize data in a nested fashion, and includes examples of more complex output operations.

COPY A JSON FILE INTO SAS

NESTED VERSUS NON-NESTED JSON FILES

A simply organized JSON file like test1.json created in the previous section does not have nested levels. The layout corresponds somewhat to the rectangular nature of a SAS data set.

Here is the first object of JSON file test2.json, used in Example 2 below. It is nested because the INCOMETAX and SALESTAX keys are nested inside the TAX key. This layout does not directly correspond to the rectangular nature of a SAS data set.

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "tax": {
              "incometax": 10,
              "salestax": 5
     }
  },
```

Reading a non-nested JSON file into SAS is much simpler than reading a nested JSON file. When a JSON file is read, ordinal variables (variables whose names start with ORDINAL_) provide a relationship between the generated data sets. For non-nested JSON files, the ordinal variables can be dropped, but for nested JSON files, they can sometimes be used to merge the generated data sets and get a meaningful result.

READING JSON FILES INTO SAS: AUTOMATING THE PROCESS

One objective in writing this paper was to come up with ways to automate the process of reading JSON files into SAS. At present, this is an area of ongoing research. The examples reflect the results to date and may be updated in future versions of this paper. Suggestions for how to easily read JSON files by readers of this paper would be greatly appreciated.

- Examples 1-3 show JSON files that can be read in a relatively automated fashion.
- Example 4 shows a JSON file that requires one change to be read in a relatively automated way.
- Example 5 shows a JSON file that appears to be most easily read in a different manner but could be somewhat automated.
- Example 6 shows a JSON file where it appears that having specific knowledge about the JSON file is needed to get the desired result, making it hard to automate.

The examples all use a small amount of data, but much larger files with the same characteristics should exhibit the same behavior as these small files.

EXAMPLE 1. READ A NON-NESTED JSON FILE INTO SAS

Read JSON file test1.json, created above, into SAS. It has the following contents.

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "date": "20201001"
  },
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "date": "20201101"
  },
  {
    "country": "canada",
    "city": "montreal",
    "income": 300,
    "date": "20201201"
  }
]
```

First, execute the following steps.

- Specify the SAS library where SAS files will be copied (library XXX).
- Remove any existing SAS files with PROC DATASETS (not necessary if the library is empty).
- Access the JSON file test1.json with a LIBNAME statement.
- Use PROC COPY to copy the JSON file contents to multiple SAS data sets in library XXX.
- List the data sets in XXX with PROC DATASETS (output not shown to conserve space). Data sets ALLDATA and ROOT are created.
- ALLDATA is always created and contains all the JSON data in one data set. One or more other SAS data sets containing components of the JSON data are also created.

```
libname xxx '/my/home/m1xxx00/example31';                /* SAS library */
```

```
proc datasets library=xxx kill;                /* Remove prior SAS files */
run;quit;

libname ex31 json '/my/home/m1xxx00/test1.json';           /* JSON file */
proc copy in=ex31 out=xxx;
run;
proc datasets lib=xxx;
run;quit;
```

Data sets ALLDATA and ROOT have the following values.

ALLDATA

Obs	P	P1	V	Value
1	1	country	1	usa
2	1	city	1	chicago
3	1	income	1	100
4	1	date	1	20201001
5	1	country	1	usa
6	1	city	1	cleveland
7	1	income	1	200
8	1	date	1	20201101
9	1	country	1	canada
10	1	city	1	montreal
11	1	income	1	300
12	1	date	1	20201201

ROOT

Obs	ordinal_root	country	city	income	date
1	1	usa	chicago	100	20201001
2	2	usa	cleveland	200	20201101
3	3	canada	montreal	300	20201201

The JSON file is not nested, so we can just drop the ORDINAL_ variables from data set ROOT to create a final data set.

```
data xxx.root;
  set xxx.root (drop=ordinal_:);
run;
```

Data set ROOT now has the following values.

Obs	country	city	income	date
1	usa	chicago	100	20201001
2	usa	cleveland	200	20201101
3	canada	montreal	300	20201201

EXAMPLE 2. READ A NESTED JSON FILE WITH TWO LEVELS INTO SAS

JSON file test2.json is nested; the INCOMETAX and SALESTAX keys are nested inside the TAX key.

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "tax": {
              "incometax": 10,
              "salestax": 5
     }
  },
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "tax": {
              "incometax": 20,
              "salestax": 10
     }
  },
  {
    "country": "canada",
    "city": "montreal",
    "income": 300,
    "tax": {
              "incometax": 30,
              "salestax": 15
     }
  }
]
```

To copy test2.json into a SAS data set, first execute the following steps.

- Specify the SAS library where SAS files will be copied (library XXX).
- Remove any existing SAS files with PROC DATASETS. This ensures that the list of data set names is generated correctly by PROC SQL below.
- Access the JSON file test2.json with a LIBNAME statement.
  - AUTOMAP=CREATE generates a JSON map and writes it to the location specified by the MAP= option.
  - MAP= specifies the location of the JSON map file.
  - ORDINALCOUNT=ALL specifies that all possible ordinal variables, which provide a relationship between the generated data sets, are generated. ORDINALCOUNT's default value is 2, but we need all possible variables to merge the generated data sets.
- Use PROC COPY to copy the JSON file contents to multiple SAS data sets in library XXX.
- List the data sets in XXX with PROC DATASETS (output not shown to conserve space). Data sets ALLDATA, ROOT, and TAX are created

```
libname xxx '/my/home/m1xxx00/example32';                /* SAS library */
proc datasets library=xxx kill;               /* Remove prior SAS files */
run;quit;
libname ex32 json '/my/home/m1xxx00/test2.json'
        map='user32.map' automap=create ordinalcount=all;  /* JSON file */
proc copy in=ex32 out=xxx;
run;
proc datasets lib=xxx;
run;quit;
```

Data sets ALLDATA, ROOT, and TAX have the following values.

ALLDATA

Obs	P	P1	P2	V	Value
1	1	country		1	usa
2	1	city		1	chicago
3	1	income		1	100
4	1	tax		0	
5	2	tax	incometax	1	10
6	2	tax	salestax	1	5
7	1	country		1	usa
8	1	city		1	cleveland
9	1	income		1	200
10	1	tax		0	
11	2	tax	incometax	1	20
12	2	tax	salestax	1	10
13	1	country		1	canada
14	1	city		1	montreal

15	1	income		1	300
16	1	tax		0	
17	2	tax	incometax	1	30
18	2	tax	salestax	1	15

ROOT

Obs	ordinal_root	country	city	income
1	1	usa	chicago	100
2	2	usa	cleveland	200
3	3	canada	montreal	300

TAX

Obs	ordinal_root	ordinal_tax	incometax	salestax
1	1	1	10	5
2	2	2	20	10
3	3	3	30	15

Now execute the following steps.

- Read a DICTIONARY table to create macro variable ALL_BUT_ALLDATA containing a space-separated list of all SAS data sets in library XXX except ALLDATA, with each data set name preceded by xxx. In this example, ALL_BUT_ALLDATA's value is as follows:
  xxx.ROOT xxx.TAX
- Merge all data sets in library XXX except ALLDATA by ORDINAL_ROOT.

```
proc sql noprint ;
   select cats("xxx.", memname) into :all_but_alldata separated by " "
   from dictionary.tables
   where libname = "XXX" and memname ne "ALLDATA"
   ;
quit ;
data xxx.finaldata2;
  merge &all_but_alldata;
  by ordinal_root;
  drop ordinal_:;
run;
```

Data set FINALDATA2 has the following values.

Obs	country	city	income	incometax	salestax
1	usa	chicago	100	10	5
2	usa	cleveland	200	20	10

```
3     canada     montreal       300          30          15
```

EXAMPLE 3. READ A NESTED JSON FILE WITH THREE LEVELS INTO SAS

JSON file test3.json is nested with three levels. It's not required that all possible values be present. MONTREAL doesn't have estimated income tax or a second car tax payment (perhaps the car was sold mid-year).

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "tax": {
              "incometax": {
                            "incometax_estimated": 3,
                            "incometax_withheld": 7
                          },
              "salestax": 5,
              "cartax": {
                            "cartax_h1": 2,
                            "cartax_h2": 2
                          }
     }
  },
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "tax": {
              "incometax": {
                            "incometax_estimated": 6,
                            "incometax_withheld": 14
                          },
              "salestax": 10,
              "cartax": {
                            "cartax_h1": 3,
                            "cartax_h2": 3
                          }
     }
```

```
  },
  {
    "country": "canada",
    "city": "montreal",
    "income": 300,
    "tax": {
              "incometax": {
                            "incometax_withheld": 30
                           },
              "salestax": 15,
              "cartax": {
                            "cartax_h1": 2
                           }
     }
  }
]
```

To copy test3json into a SAS data set, first repeat the previous example's steps that specify the SAS library and JSON file, copy the JSON file contents to multiple SAS data sets, and list the data sets. See the previous example for an explanation of the code.

```
libname xxx '/my/home/m1xxx00/example33';                /* SAS library */
proc datasets library=xxx kill;                /* Remove prior SAS files */
run;quit;
libname ex33 json '/my/home/m1xxx00/test3.json'
        map='user33.map' automap=create ordinalcount=all;  /* JSON file */
proc copy in=ex33 out=xxx;
run;
proc datasets lib=xxx;
run;quit;
```

The SAS library contains data sets ALLDATA, ROOT, TAX, TAX_CARTAX, and TAX_INCOMETAX; all but ALLDATA are displayed below. Missing values for INCOMETAX_ESTIMATED and CARTAX_H2 in the third observation of the respective data sets reflect values not present in the JSON file.

ROOT

Obs	ordinal_root	country	city	income
1	1	usa	chicago	100
2	2	usa	cleveland	200
3	3	canada	montreal	300

TAX

Obs	ordinal_root	ordinal_tax	salestax
1	1	1	5
2	2	2	10
3	3	3	15

TAX_INCOMETAX

Obs	ordinal_ root	ordinal_ tax	ordinal_ incometax	incometax_ estimated	incometax_ withheld
1	1	1	1	3	7
2	2	2	2	6	14
3	3	3	3	.	30

TAX_CARTAX

Obs	ordinal_ root	ordinal_ tax	ordinal_ cartax	cartax_ h1	cartax_ h2
1	1	1	1	2	2
2	2	2	2	3	3
3	3	3	3	2	.

Now repeat the previous example's steps that create a macro variable containing a space-separated list of all SAS data sets in library XXX except ALLDATA with each data set name preceded by xxx, and merge those data sets by ORDINAL_ROOT. The macro variable's value is as follows:

xxx.ROOT xxx.TAX xxx.TAX_CARTAX xxx.TAX_INCOMETAX

```
proc sql noprint ;
   select cats("xxx.", memname) into :all_but_alldata separated by " "
   from dictionary.tables
   where libname = "XXX" and memname ne "ALLDATA"
   ;
quit ;
data xxx.finaldata3;
  merge &all_but_alldata;
  by ordinal_root;
  drop ordinal_:;
run;
```

Data set FINALDATA3 has the following values.

Obs	country	city	income	salestax	cartax_ h1	cartax_ h2	incometax_ estimated	incometax_ withheld
1	usa	chicago	100	5	2	2	3	7
2	usa	cleveland	200	10	3	3	6	14
3	canada	montreal	300	15	2	.	.	30

EXAMPLE 4. READ A JSON FILE WITH NAME COLLISIONS AND RETAINED VALUES

The *SAS® 9.4 Global Statements: Reference* (*LIBNAME Statement: JSON Engine* chapter, *Creating and Editing a JSON MAP Data* section) contains the following JSON file, which we'll call test4.json.

```
[
   {
    "type": "Full",
    "info" : [
       { "name" : "Eric" , "age" : 21, "phone" : [
          { "type" : "cell", "number" : "540-555-2377" },
          { "type" : "home", "number" : "540-555-0120" }
          ]
       },
       { "name" : "John",  "age" : 22, "phone" : [
          { "type" : "cell", "number" : "919-555-6665" },
          { "type" : "home", "number" : "336-555-0140" }
          ]
       }
     ]
   },
   {
    "type": "Part",
    "info" : [
       { "name" : "Bjorn" , "age" : 27, "phone" : [
          { "type" : "cell",   "number" : "720-555-8377" },
          { "type" : "burner", "number" : "720-555-2877" },
          { "type" : "home",   "number" : "720-555-0194" }
          ]
       }
     ]
   }
]
```

Let's repeat the previous example's steps that specify the SAS library and JSON file, copy the JSON file contents to multiple SAS data sets, and list the data sets.

```
libname xxx '/my/home/m1xxx00/example34';               /* SAS library */
proc datasets library=xxx kill;              /* Remove prior SAS files */
run;quit;
libname ex34 json '/my/home/m1xxx00/test4.json'
        map='user34.map' automap=create ordinalcount=all;  /* JSON file */
proc copy in=ex34 out=xxx;
run;
proc datasets lib=xxx;
run;quit;
```

Data sets ALLDATA, ROOT, INFO, and INFO_PHONE are created, as follows.

ALLDATA

Obs	P	P1	P2	P3	V	Value
1	1	type			1	Full
2	1	info			0	
3	2	info	name		1	Eric
4	2	info	age		1	21
5	2	info	phone		0	
6	3	info	phone	type	1	cell
7	3	info	phone	number	1	540-555-2377
8	2	info	phone		0	
9	3	info	phone	type	1	home
10	3	info	phone	number	1	540-555-0120
11	1	info			0	
12	2	info	name		1	John
13	2	info	age		1	22
14	2	info	phone		0	
15	3	info	phone	type	1	cell
16	3	info	phone	number	1	919-555-6665
17	2	info	phone		0	
18	3	info	phone	type	1	home
19	3	info	phone	number	1	336-555-0140
20	1	type			1	Part
21	1	info			0	
22	2	info	name		1	Bjorn
23	2	info	age		1	27

24	2	info	phone		0	
25	3	info	phone	type	1	cell
26	3	info	phone	number	1	720-555-8377
27	2	info	phone		0	
28	3	info	phone	type	1	burner
29	3	info	phone	number	1	720-555-2877
30	2	info	phone		0	
31	3	info	phone	type	1	home
32	3	info	phone	number	1	720-555-0194

ROOT

Obs	ordinal_root	type
1	1	Full
2	2	Part

INFO

Obs	ordinal_root	ordinal_info	name	age
1	1	1	Eric	21
2	1	2	John	22
3	2	3	Bjorn	27

INFO_PHONE

Obs	ordinal_root	ordinal_info	ordinal_phone	type	number
1	1	1	1	cell	540-555-2377
2	1	1	2	home	540-555-0120
3	1	2	3	cell	919-555-6665
4	1	2	4	home	336-555-0140
5	2	3	5	cell	720-555-8377
6	2	3	6	burner	720-555-2877
7	2	3	7	home	720-555-0194

Two things distinguish this JSON file from the previous examples.

- There are two different TYPE properties: TYPE of employee (Full or Part) and TYPE of phone (cell, burner, home). This kind of name collision would interfere with merging values from the component data sets, as done in the previous examples.
- The same values are used in multiple observations in the SAS data set. For example, in the first data object, there are two full-time employees, Eric and John, who each have two phones. In the final SAS data set below, the TYPE (renamed to TYPEEMP, below) is “Full” in the first four observations, and NAME is “Eric” and AGE is “21” in the first two observations.

The approach taken in the SAS *LIBNAME Statement: JSON Engine* documentation is to manually edit the JSON map file and make multiple changes. We'll take the following approach.

- Manually rename the TYPE property for type of employee to TYPEEMP in the JSON file.
- First merge INFO and INFO_PHONE by ORDINAL_INFO, then merge that data set with ROOT by ORDINAL_ROOT.

After repeating the code above for the modified JSON file (not shown), submit the following code.

```
data xxx.finaldata4;
  merge xxx.info xxx.info_phone;
  by ordinal_info;
run;
data xxx.finaldata4;
  merge xxx.finaldata4 xxx.root;
  by ordinal_root;
  drop ordinal_:;
run;
```

Data set FINALDATA4 after the first merge has the following values.

Obs	ordinal_ root	ordinal_ info	name	age	ordinal_ phone	type	number
1	1	1	Eric	21	1	cell	540-555-2377
2	1	1	Eric	21	2	home	540-555-0120
3	1	2	John	22	3	cell	919-555-6665
4	1	2	John	22	4	home	336-555-0140
5	2	3	Bjorn	27	5	cell	720-555-8377
6	2	3	Bjorn	27	6	burner	720-555-2877
7	2	3	Bjorn	27	7	home	720-555-0194

Data set FINALDATA4 after the second merge has the following values.

Obs	name	age	type	number	typeemp
1	Eric	21	cell	540-555-2377	Full
2	Eric	21	home	540-555-0120	Full
3	John	22	cell	919-555-6665	Full
4	John	22	home	336-555-0140	Full
5	Bjorn	27	cell	720-555-8377	Part
6	Bjorn	27	burner	720-555-2877	Part
7	Bjorn	27	home	720-555-0194	Part

To test for name collisions, we can check if any variable names are in more than one "P variable" (P1, P2, and P3 in this case) in ALLDATA. The "P variables" contain the properties from the JSON file in a way that reflects the way they are nested.

The JSON libname engine creates several macro variables. One of them, of the form *libref*_JADPNUM, where *libref* is the libref used to access the JSON file, contains the number of "P variables". It's named ex34_JADPNUM in this case.

```
%macro collisions;
  %local i p_current p_allnames;
    /* Unique variable names from each P variable are
       concatenated in macro variable P_ALLNAMES */
  %let p_allnames=;
  %do i=1 %to &json14a_JADPNUM;
     proc sql noprint;
       select distinct P&i
       into :p_current separated by ' '
       from xxx.alldata;
     quit;
     %let p_allnames= &p_allnames &p_current;
  %end;
  data _null_;
    array names(1000) $32 v1-v1000;                  /* Variable names */
    length duplicates $200;                   /* Duplicate names if any */
      /* Copy variable names to array elements */
    p_allnames="&p_allnames";
    i=1;
    names(i) = scan(p_allnames, 1, " ");
    do while (names(i) ne " ");
     i+1;
     names(i) = scan(p_allnames, i, " ");
    end;
      /* Sort names.  Blank sorts lowest so only the last I-1 array
         elements contain names.  If any adjacent array elements are
         equal, there are duplicates. */
    call sortc(of names(*));
    do j= 1000-i+1 to 1000;
      if names(j)=names(j-1) then do;
         if index(duplicates,strip(names(j)))=0
            then duplicates=catx(' ',duplicates,names(j));
      end;
    end;
    if duplicates="" then put "No name collisions in P variables";
```

```
                    else put "Name collisions in P variables: " duplicates;
  run;
%mend collisions;
%collisions;
```

The macro finds duplicate TYPE names in the original ALLDATA data set but no duplicates in the ALLDATA data set generated from the modified JSON file.

EXAMPLE 5. READING THE ALLDATA FILE

Consider the following JSON file, test5.json.

```
{
    "quiz": {
        "sport": {
            "q1": {
                "question": "# of NY Knicks titles?",
                "option": [
                    "one",
                    "two",
                    "three",
                    "four"
                ],
                "answer": "two"
            }
        },
        "math": {
            "q1": {
                "question": "5 + 7 = ?",
                "option": [
                    "10",
                    "11",
                    "12",
                    "13"
                ],
                "answer": "12"
            },
            "q2": {
                "question": "12 - 8 = ?",
                "option": [
```

```
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "answer": "4"
            }
        }
    }
```

The desired SAS data set is as follows.

Obs	answer	option1	option2	option3	option4	question
1	two	one	two	three	four	# of NY Knicks titles?
2	12	10	11	12	13	5 + 7 = ?
3	4	1	2	3	4	12 - 8 = ?

Let's execute the same code as the previous example to copy the JSON file to multiple SAS data sets, and show the resulting SAS data sets.

```
libname xxx '/my/home/m1xxx00/example35';                /* SAS library */
proc datasets library=xxx kill;               /* Remove prior SAS files */
run;quit;
libname ex35 json '/my/home/m1xxx00/test5.json'
        map='user35.map' automap=create ordinalcount=all;  /* JSON file */
proc copy in=ex35 out=xxx;
run;
proc datasets lib=xxx;
run;quit;
```

SPORT_Q1

Obs	ordinal_root	ordinal_quiz	ordinal_sport	ordinal_q1	question	answer
1	1	1	1	1	# of NY Knicks titles?	two

Q1_OPTION

Obs	ordinal_root	ordinal_quiz	ordinal_sport	ordinal_q1	ordinal_option	option1	option2	option3	option4
1	1	1	1	1	1	one	two	three	four

MATH_Q1

Obs	ordinal_ root	ordinal_ quiz	ordinal_ math	ordinal_ q1	question	answer
1	1	1	1	1	5 + 7 = ?	12

Q1_OPTION2

Obs	ordinal_ root	ordinal_ quiz	ordinal_ math	ordinal_ q1	ordinal_ option	option1	option2	option3	option4
1	1	1	1	1	1	10	11	12	13

MATH_Q2

Obs	ordinal_ root	ordinal_ quiz	ordinal_ math	ordinal_ q2	question	answer
1	1	1	1	1	12 - 8 = ?	4

Q2_OPTION

Obs	ordinal_ root	ordinal_ quiz	ordinal_ math	ordinal_ q2	ordinal_ option	option1	option2	option3	option4
1	1	1	1	1	1	1	2	3	4

ALLDATA

Obs	P	P1	P2	P3	P4	P5	V	Value
1	1	quiz					0	
2	2	quiz	sport				0	
3	3	quiz	sport	q1			0	
4	4	quiz	sport	q1	question		1	# of NY Knicks titles?
5	4	quiz	sport	q1	option		0	
6	5	quiz	sport	q1	option	option1	1	one
7	5	quiz	sport	q1	option	option2	1	two
8	5	quiz	sport	q1	option	option3	1	three
9	5	quiz	sport	q1	option	option4	1	four
10	4	quiz	sport	q1	answer		1	two
11	2	quiz	math				0	
12	3	quiz	math	q1			0	
13	4	quiz	math	q1	question		1	5 + 7 = ?
14	4	quiz	math	q1	option		0	
15	5	quiz	math	q1	option	option1	1	10
16	5	quiz	math	q1	option	option2	1	11

17	5	quiz	math	q1	option	option3	1	12
18	5	quiz	math	q1	option	option4	1	13
19	4	quiz	math	q1	answer		1	12
20	3	quiz	math	q2			0	
21	4	quiz	math	q2	question		1	12 - 8 = ?
22	4	quiz	math	q2	option		0	
23	5	quiz	math	q2	option	option1	1	1
24	5	quiz	math	q2	option	option2	1	2
25	5	quiz	math	q2	option	option3	1	3

The ORDINAL_ variables all have the value 1, so using them to merge the data sets will be difficult. Instead, we'll generate the final data set by reading the ALLDATA data set, noting the following.

- The values to use are from the variable VALUE, for observations where V=1.
- The variable to assign the values to is from the highest non-blank "P variable". For example, in observation 4, V is 1, P5 is blank, and P4 is "question". So, we set QUESTION to "# of NY Knicks titles?".
- The first variable we encounter is "question" in observation 5. Each time we encounter "question" again indicates the beginning of a new observation in the final data set. While every variable needn't be present in every observation, we'll make the simplifying assumption that this "first" variable is always present.
- Variables could be numeric or character. The same variable could be numeric in some non-ALLDATA data sets and character in others. For example, if ANSWER was numeric for objects Q1 and Q2 in the JSON file (12 and 4 instead of "12" and "4"), it would be numeric in MATH_Q1 and MATH_Q2 and character with length 3 in SPORT_Q1.
- Variables that are character in any non-ALLDATA data set will be character in the final data set, using the longest length.
- We read ALLDATA and write DATA step code to a temporary file, and then %INCLUDE the file in a subsequent DATA step. This handles two potential issues reasonably easily.
    - The appropriate "P variable" value needs to be on the left side of an assignment statement (e.g., question="# of NY Knicks titles?").
    - By writing values from VALUE to the temporary file as text, we avoid some numeric/character conversion issues.

```
%macro readjson;
  %local i p_current p_allvars all_charvars currentvar p_numvars p_charvars;
  %let p_allvars=;
  proc sql noprint;

    /* Build list of all variables in the output data set */
  %do i=1 %to &ex35_JADPNUM;
    select distinct P&i
       into :p_current separated by ' '
```

```
     from xxx.alldata
     where v=1 and p=&i;
  %let p_allvars= &p_allvars &p_current;
  %put i=&i p_allvars=&p_allvars;
%end;

  /* Create macro variable allcharvars with quoted character
     variables space-separated from all data sets except ALLDATA.
     Same variable could be numeric in one data set and character
     in another, so check all data sets except ALLDATA and a variable
     that is character anywhere is considered character. */
select distinct quote(trim(name))
  into :all_charvars separated by " "
  from dictionary.columns
    where libname = "XXX" and memname ne "ALLDATA"
    and type="char"
  ;
quit;

                /* Split p_allvars into character and numeric variables */
%let p_numvars=;                                    /* numeric variables */
%let p_charvars=;                                 /* character variables */
%let currentvar = %scan(&p_allvars, 1, %str( ));       /* parse 1st word */
%let i = 1; /* parse 2nd, 3rd, ... word in %DO loop */
%do %while (&currentvar ne ) ;           /* stop when %scan returns null */
  %if %index(&all_charvars, "&currentvar") = 0
    %then %let p_numvars= &p_numvars &currentvar;
    %else %let p_charvars= &p_charvars &currentvar;
  %let i=%eval(&i+1);                  /* set counter to parse next word */
  %let currentvar = %scan(&p_allvars, &i, %str( ));   /* parse next word */
%end;

  /* If any character variables, create LENGTH statement for
     all character variables using the longest length of each
     variable in all data sets except ALLDATA */
%if p_charvars ne %then %do;
 /* 1st create data set with name and all lengths of character variables */
```

```
  proc sql;
     create table allcharvariables as
     select name, length
     from dictionary.columns
     where libname = "XXX" and memname ne "ALLDATA"
     and type = "char";
  quit ;

    /* Sort by name and descending order of length so that
       first.name has the longest length for each variable */
  proc sort data=allcharvariables;
    by name descending length;
  run;
    /* Create LENGTH statement with longest length of each variable */
  data _null_;
  set allcharvariables end=last;
    by name descending length;
    length length_statement $10000;
    retain length_statement;
    if first.name then
      length_statement = trim(length_statement)
      || " " || trim(name)
      || " $" || compress(put(length,3.));
    /* At end of DATA step create macro variable w/LENGTH statement info */
  if last then
    call symput("length_of_vars",trim(length_statement));
  run;
%end; /* of  %if p_charvars ne %then %do; */

  /* Write code to temporary file, will include it later */
filename out1 temp;
data _null_;
  set xxx.alldata end=last;
    /* VARFLAG = 1st variable in output data set, when
       encounter it, we know that new record is starting */
  length varflag $32;
  retain varflag "";
```

```
array pall $ p1-p&ex35_JADPNUM;            /* Names in the P variables */
file out1;                                /* Write generated code to file */
if varflag="" and v = 1 then do;             /* Found very 1st variable*/
  varflag=pall(p);         /* Save so we know when a new record starts */
  /* Write initial 1x code, first variable */
  %if &p_charvars ne %then %do;
    put "length &length_of_vars;";
    put "array allcharvars (*) $ &p_charvars;";
  %end;
  %if &p_numvars ne %then %do;
    put "array allnumvars (*) $ &p_numvars;";
  %end;
  put "keep &p_allvars;";
    /* If only numeric variables, always write a numeric value,
       otherwise check if current variable is character or numeric */
  %if &p_charvars eq %then %do;
    put pall(p) "=" value ";";                  /* Write numeric value */
  %end;
  %else %do;
    if pall(p) in (&all_charvars)
      then put pall(p) "='" value +(-1) "';"; /* Write character value */
      else put pall(p) "=" value ";";           /* Write numeric value */
  %end;
end;
else if v = 1 then do;     /* Found a variable to write after 1st time */
  if pall(p) = varflag then do;                 /* Start of new record */
     put "output;";                /* Write record we have accumulated */
       /* Clear out variables for new record in case not
          all variables have values in this record */
     %if &p_charvars ne %then %do;
       put "do i=1 to dim(allcharvars); allcharvars(i)='';end;";
     %end;
     %if &p_numvars ne %then %do;
       put "do i=1 to dim(allnumvars); allnumvars(i)=.;end;";
     %end;
  end;
    /* If only numeric variables, always write a numeric value,
```

```
          otherwise check if current variable is character or numeric */
      %if &p_charvars eq %then %do;
        put pall(p) "=" value ";"; /* Write numeric value */
      %end;
      %else %do;
        if pall(p) in (&all_charvars)
          then put pall(p) "='" value +(-1) "';"; /* Write character value */
          else put pall(p) "=" value ";";          /* Write numeric value */
      %end;
    end; /* of  else if v = 1 then do */
    if last then put "output;";                    /* Write final record */
  run;

    /* Run the DATA step to create the SAS data set */
  data finaldata5;
    %include out1;
  run;
%mend readjson;
%readjson;
```

EXAMPLE 6. CUSTOM CODING REQUIRED

The *SAS® 9.4 Global Statements: Reference* (*LIBNAME Statement: JSON Engine* chapter, *ALLDATA Data set* section) contains the first 24 records of the following ALLDATA data set.

OBS	P	P1	P2	P3	P4	V	Value
1	1	stores				0	
2	2	stores	Name			1	Bob's Mart
3	2	stores	opened			1	06-01-2001
4	2	stores	sales			0	
5	3	stores	sales	Hot_Dogs		0	
6	4	stores	sales	Hot_Dogs	count	1	39
7	4	stores	sales	Hot_Dogs	price	1	1.09
8	3	stores	sales	Salami		0	
9	4	stores	sales	Salami	count	1	20
10	4	stores	sales	Salami	price	1	5.99
11	3	stores	sales	Canteloupes		0	
12	4	stores	sales	Canteloupes	count	1	26
13	4	stores	sales	Canteloupes	price	1	1.39
14	3	stores	sales	Mustard		0	

15	4	stores	sales	Mustard	count	1	6
16	4	stores	sales	Mustard	price	1	2.19
17	2	stores	Code			1	12BMx2
18	1	stores				0	
19	2	stores	Name			1	Grab 'n' Git
20	2	stores	opened			1	06-03-2012
21	2	stores	sales			0	
22	3	stores	sales	Hot_Dogs		0	
23	4	stores	sales	Hot_Dogs	count	1	18
24	4	stores	sales	Hot_Dogs	price	1	1.19

DATA step code is used to read ALLDATA and create the following data set.

OBS	StoreName	Code	Item	Count	Price
1	Bob's Mart	12BMx2	Hot_Dogs	39	1.09
2	Bob's Mart	12BMx2	Salami	20	5.99
3	Bob's Mart	12BMx2	Canteloupes	26	1.39
4	Bob's Mart	12BMx2	Mustard	6	2.19
5	Grab 'n' Git	10GNx9	Hot_Dogs	18	1.19
6	Grab 'n' Git	10GNx9	Salami	3	7.99
7	Grab 'n' Git	10GNx9	Mustard	6	2.19
8	Grab 'n' Git	10GNx9	Beer	20	8.99
9	Larry's Quick Shoppe	17LQx2	Hot_Dogs	39	1.09
10	Larry's Quick Shoppe	17LQx2	Salami	20	5.99
11	Larry's Quick Shoppe	17LQx2	Mustard	6	2.19
12	Larry's Quick Shoppe	17LQx2	Beer	7	8.99
13	Larry's Quick Shoppe	17LQx2	Wine	15	12.99

The DATA step code takes advantage of knowledge about the contents of ALLDATA. For example, it assigns the values in P3 to a variable called ITEM. In some cases, like this one, it might be very difficult to automate copying the JSON file into SAS, and custom code that takes advantage of content knowledge as shown in the SAS documentation might be the best or even only approach.

JSONL FILES

As noted in Hemedinger (2018), a format growing in popularity is newline-delimited JSON (a.k.a. JSONL or JSON Lines). Each text line represents a valid JSON object, but there is no hierarchical relationship between the lines so a JSONL file is not valid JSON. JSONL files can be converted to a JSON file in a DATA step, as noted in Hemedinger (2018) and by Tom Abernathy in a SAS Communities thread (2020b).

Here is a small JSONL file, four.jsonl, with data values from a prior example.

```
{"country": "usa","city": "chicago","income": 100,"date": "20201001"}

{"country": "usa","city": "cleveland","income": 200,"date": "20201101"}

{"country": "canada", "city":"montreal", "income":300, "date": "20201201"}
```

Here is code to convert JSONL file four.jsonl to JSON file four.json.

```
filename jsonl "/my/home/m1xxx00/four.jsonl";

filename json "/my/home/m1xxx00/four.json";
```

```
data _null_;
  infile jsonl end=eof;
  file json;
  input;
  if _n_=1 then put '[' @ ;
  else put ',' @ ;
  put _infile_;
  if eof then do;
     put ']';
  end;
run;
```

Here is the resulting JSON file, which can be read using the techniques in the prior section.

```
[{"country": "usa","city": "chicago","income": 100,"date": "20201001"}
,{"country": "usa","city": "cleveland","income": 200,"date": "20201101"}
,{"country": "canada", "city": "montreal", "income": 300, "date": "20201201"}
]
```

JSONPP DATA STEP FUNCTION

When copying a SAS data set to a JSON file, the PROC JSON statement option PRETTY creates a JSON file in a human-readable format with indentation and multiple lines instead of one long record.

The JSONPP DATA step function copies an existing single record JSON file to a “pretty” JSON file.

Here’s the single record JSON file created in Example 3 in the section on copying a SAS data set to a JSON file.

```
[{"country":"usa","city":"chicago","income":100,"date":"20201001"},{"country"
:"usa","city":"cleveland","income":200,"date":"20201101"},{"country":"canada"
,"city":"montreal","income":300,"date":"20201201"}]
```

This DATA step statement copies single record JSON file test1.json to “pretty” JSON file test1pretty.json.

```
rc = jsonpp('/my/home/m1xxx00/test1.json',
            '/my/home/m1xxx00/test1pretty.json');
```

Here is the resulting “pretty” JSON file.

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "date": "20201001"
  },
```

```
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "date": "20201101"
  },
  {
    "country": "canada",
    "city": "montreal",
    "income": 300,
    "date": "20201201"
  }
]
```

CONCLUSION

JavaScript Object Notation (JSON) is an open standard file format and data interchange format used for some of the same purposes as XML. More information about JSON is readily available on the internet.

Starting in SAS ® 9.4, you can copy SAS data sets to JSON files with PROC JSON. Starting in SAS ® 9.4TS1M4, you can copy JSON files to SAS data sets with the JSON engine.

Copying data from SAS to JSON with PROC JSON is relatively straightforward. Copying data from JSON to SAS can be much more complicated in some cases. Copying data from JSON to SAS in an automated way that does not rely on extensive knowledge of data specifics is of course desirable. This paper included examples where copying JSON files to SAS could be automated relatively easily, cases where automation required a good deal of coding, and a case where prior knowledge of the data made automation difficult or perhaps not realistic. Determining how to copy additional types of JSON files into SAS in an automated way is an area of ongoing research, and input from readers of this paper would be greatly appreciated.

REFERENCES

Hemedinger, Chris. 2018. “Create newline-delimited JSON (or JSONL) with SAS,” The SAS Dummy blog, published November 14, 2018. Available at https://blogs.sas.com/content/sasdummy/2018/11/14/jsonl-with-proc-json/

SAS Communities thread. 2020a. “Automating reading JSON files into SAS”. Available at https://communities.sas.com/t5/SAS-Programming/Automating-reading-JSON-files-into-SAS/m-p/710327#

SAS Communities thread. 2020b. “SAS EG - how to read in JSON file”. Available at https://communities.sas.com/t5/SAS-Programming/SAS-EG-how-to-read-in-JSON-file/m-p/710217

SAS Institute Inc. 2017a. "JSON Procedure". In *Base SAS® 9.4 Procedures Guide, Seventh Edition*. Cary, NC: SAS Institute Inc. Available at https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=proc&docsetTarget=p0ie4bw6967jg6n1iu629d40f0by.htm&locale=en

SAS Institute Inc. 2017b. "LIBNAME Statement: JSON Engine". In *Base SAS® 9.4 Global Statements: Reference*. Cary, NC: SAS Institute Inc. Available at https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=lestmtsglobal&docsetTarget=n1jfdetszx99ban1rl4zll6tej7j.htm&locale=en

ACKNOWLEDGMENTS

Support from the following people is greatly appreciated. Chevell Parker at SAS Institute provided sustained technical support that greatly increased my understanding of JSON files. Tom Abernathy at Pfizer Inc. provided detailed information on the SAS Communities site, including the DATA step to convert a JSONL file to a JSON file. Donna Hill (technical document review), Heidi Markovitz (SAS content review), and Sandesh Shetty (JSON information) at the Federal Reserve Board all contributed substantially to the development of this paper.

The following applies to examples 4 and 6 in the section “Copy a JSON file into SAS”:

CONTACT INFORMATION

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

Bruce Gilsen
Federal Reserve Board, Mail Stop N-122, Washington, DC 20551
202-452-2494
bruce.gilsen@frb.gov