Paper SAS3506-2019

Creating and Controlling JSON Output with the JSON Procedure

Adam Linker, SAS® Institute Inc.

ABSTRACT

JSON is continuing to grow as the preferred data interchange format due to its simplicity and versatility. The JSON procedure gives SAS® users the ability to export SAS® data sets in JSON, as well as the ability to create custom JSON output. The procedure is simple to use and gives the user a huge amount of flexibility and control of the JSON output. This paper gives an overview of how to use the JSON procedure as well as detailed use cases to highlight the most important options to get the most out of the output generated by the procedure.

INTRODUCTION

JSON (JavaScript Object Notation) is a text-based data interchange format used to store information in a simple, easy to understand, and compact way. Because JSON is text based, it is easy for machines to parse and generate, it can be used across almost any platform, and it can be read and written easily by a human. In its simplest form, the JSON procedure allows the user to take a SAS® data set and create a JSON output file from the data. The user can also customize the JSON output by using a variety of options. In addition to that, the user can create their own JSON separate from the SAS® data set using the procedure. The user has control of their JSON output using these options along with the WRITE statement. This paper will explore how a user can get the most out of their JSON output using the tools provided by PROC JSON.

The following is a use case that will show how to use PROC JSON in its simplest form along with how to utilize the options that PROC JSON provides to customize and control the resulting JSON output file.

Use Case A

- The user has a SAS® data set that they want to convert to the JSON format.
- The JSON must be exactly representative of the SAS® data set (customers) shown in Figure 1.

	First_Name	Last_Name	Email	Phone_Number
1	jared	ells	jells@test.com	919-111-1111
2	mark	poppins	mpopp@test.com	919-222-2222
3	stu	gotz	sgotz@test.com	919-333-3333
4	cara	simmons	csimms@test.com	919-444-4444
5	tom	ford	tford@test.com	919-555-5555
6	jerry	black	jblac@test.com	919-666-6666
7	susan	brown	subro@test.com	919-777-7777
8	elise	lane	elane@test.com	919-888-8888
9	gabi	stone	gstone@test.com	919-999-9999
10	jason	fitz	jfitz@test.com	919-123-4567
11	pat	aires	paires@test.com	919-987-6543

Figure 1. Customers SAS® Data Set

- The user has a second SAS® data set (employees) that they also want to convert to JSON, and they want it to be in the same JSON file as the first data set. The second data set is shown in Figure 2.

	First_Name	Last_Name	ID_Number	Age	Start_Date	Job_Title
1	adam	linker	123456	23	20910	programmer
2	bob	smith	584932	33	16284	sales
3	sue	stevens	258432	57	10774	sales
4	paul	mills	355783	42	15066	marketing
5	sally	johnson	443767	48	14518	manager
6	max	hart	793424	50	13939	sales
7	casey	jones	648235	34	17014	programmer
8	blair	carter	363245	36	15522	marketing
9	drew	williams	347783	63	10471	manager
10	kate	lockett	753356	29	19145	programmer
11	eric	stuart	113547	48	14915	programmer
12	dan	baker	567821	40	16162	programmer
13	ella	walsh	983753	26	18383	sales

Figure 2. Employees SAS® Data Set

- The user does not want to have to modify the SAS® data set, but they want to change some of the information in the JSON output file:
  - Start_Date is a date that is represented as a SAS® data value that they want to be formatted as DD/MM/YYYY in the JSON output file.
  - Only output the observations where the age is less than 45.
  - The variable names should be suppressed.
  - Specify a table name for the data, "Employees under 45."

This can all easily be accomplished using PROC JSON, and further customization can also be done to create the exact JSON output file needed by the user. In the next section, we will see how PROC JSON can be used effectively in this use case.

SIMPLE OVERVIEW OF PROC JSON SYNTAX

Before diving into this use case, consider the syntax used in PROC JSON:

```
PROC JSON out=fileref | "external-file" <option(s)>;
   EXPORT <libref.>SAS®-data-set <(SAS®-data-set-option(s))> </option(s)>;
   WRITE value(s) </option(s)>;
   WRITE OPEN type;
   WRITE CLOSE;
run;
```

The PROC JSON statement consists of an output file provided by the user where all the JSON will be written, followed by any options to control the output.

The EXPORT statement identifies the SAS® data set to be exported and allows the user to control the resulting output by using options that are specific to PROC JSON as well as SAS® data set options that are applied to the input SAS® data set.

In addition to exporting data sets, PROC JSON gives the user the ability to write custom information to the output file with the WRITE statement, which allows the user to write one or more literal values to the JSON output file. The value can be either a string, a number, a Boolean value (TRUE or FALSE), or NULL. The WRITE OPEN and WRITE CLOSE statements allow the user to control the containers (more on containers later) in the JSON output file.

For the customers data set in the use case, the desired JSON output file can be created with a very simple use of PROC JSON:

```
proc json out="sample1.json";
  export customers;
run;
```

Output 1 shows the JSON output file that is generated from this statement.

```
{"SASJSONExport":"1.0","SASTableData+CUSTOMERS":[{"First_Name":"jared",
"Last_Name":"ells","Email":"jells@test.com","Phone_Number":"919-111-111
1"},{"First_Name":"mark","Last_Name":"poppins","Email":"mpopp@test.com"
,"Phone_Number":"919-222-2222"},{"First_Name":"stu","Last_Name":"gotz",
"Email":"sgotz@test.com","Phone_Number":"919-333-3333"},{"First_Name":"
cara","Last_Name":"simmons","Email":"csimms@test.com","Phone_Number":"9
19-444-4444"},{"First_Name":"tom","Last_Name":"ford","Email":"tford@tes
t.com","Phone_Number":"919-555-5555"},{"First_Name":"jerry","Last_Name"
:"black","Email":"jblac@test.com","Phone_Number":"919-666-6666"},{"Firs
t_Name":"susan","Last_Name":"brown","Email":"subro@test.com","Phone_Num
ber":"919-777-7777"},{"First_Name":"elise","Last_Name":"lane","Email":"
elane@test.com","Phone_Number":"919-888-8888"},{"First_Name":"gabi","La
st_Name":"stone","Email":"gstone@test.com","Phone_Number":"919-999-9999
"},{"First_Name":"jason","Last_Name":"fitz","Email":"jfitz@test.com","P
hone_Number":"919-123-4567"},{"First_Name":"pat","Last_Name":"aires","E
mail":"paires@test.com","Phone_Number":"919-987-6543"}]}
```


Output 1. JSON Output File Generated from the PROC JSON Example

Using default options will allow the user to create the required output for the customers data set. It is jumbled and hard to read, but it would be easy for a computer to parse. To create the desired output for the second data set, more work will need to be done.

USING OPTIONS TO CONTROL JSON OUTPUT

PROC JSON options enable the user to control and customize the generated output. Here is a list of the possible options:

FMTCHARACTER | NOFMTCHARACTER

Determines whether to apply a character SAS® format to the resulting output if a character SAS® format is associated with a SAS® data set variable.

FMTDATETIME | NOFMTDATETIME

Determines whether to apply a date, time, or datetime SAS® format to the resulting output if a date, time, or datetime SAS® format is associated with a SAS® data set variable.

FMTNUMERIC | NOFMTNUMERIC

Determines whether to apply a numeric SAS® format to the resulting output if a numeric SAS® format is associated with a SAS® data set variable.

KEYS | NOKEYS

Determines whether to include or suppress SAS® variable names in the JSON output file.

PRETTY | NOPRETTY

Determines how to format the JSON output. (Valid in PROC JSON statement only.)

SASTAGS | NOSASTAGS

Determines whether to include or suppress SAS® metadata at the top of the JSON output file.

SCAN | NOSCAN

Determines whether PROC JSON scans and encodes input strings to ensure that only characters that are acceptable are exported to the JSON output file.

TRIMBLANKS | NOTRIMBLANKS

Determines whether to remove or retain trailing blanks from the end of character data in the JSON output.

TABLENAME = "name"

Specifies a name for the exported SAS® data set. (Valid in EXPORT statement only.)

(SAS®-data-set-option(s))

Specifies SAS® data set options that apply to the input SAS® data set. (Valid in EXPORT statement only.)

Most of the options can be specified in the PROC JSON statement as well as in the EXPORT statement. If they are specified in both, the EXPORT statement takes precedence. Because of this, each data set can have its own options. Specify any common options to be used for each data set in the PROC JSON statement, and then specify options specific to each data set in the corresponding EXPORT statements.

Here is what PROC JSON will look like to produce the desired JSON output file:

```
proc json out="sample2.json" pretty;
  export x.customers;
  export x.employees (where=(age>45)) /*SAS data set option - only get employees with age>45*/
         / nokeys fmtdatetime tablename="Employees under 45"; /*PROC JSON options for this data set*/
                                                              /*nokeys to suppress variable names*/
                                                              /*formats dates and gives a custom tablename*/
run;
```

In the previous example, the output file looks messy and is not very easy to read. It will work fine if the user does not need to or want to look at the JSON file, but if the user specifies the PRETTY option, the JSON output will be formatted in much more human-readable and structured way. This makes it much easier to make sure the user has the JSON exactly the way they want it.

PROC JSON allows users to export multiple data sets in the same JSON output file. The first export statement will remain the same for the first data set. In the second export statement, the WHERE= data set option specifies which observations will be written to the JSON output file, and the SAS® datetime format is applied to the Start_Date column of the employees data set by adding the FMTDATETIME option.

The TABLENAME= option specifies the new table name to use in the output file. The last option, NOKEYS, will suppress the variable names as well.

Below is the resulting output from the PROC JSON example code. The left column shows the beginning of the JSON output file containing the first data set (only a few observations are shown). The second column shows the end of the JSON output file containing the second data set (a portion of the JSON output file is not shown). Only the observations with an age greater than 45 were exported.


```
{
  "SASJSONExport": "1.0 PRETTY",
  "SASTableData+CUSTOMERS": [
    {
      "First_Name": "jared",
      "Last_Name": "ells",
      "Email": "jells@test.com",
      "Phone_Number": "919-111-1111"
    },
    {
      "First_Name": "mark",
      "Last_Name": "poppins",
      "Email": "mpopp@test.com",
      "Phone_Number": "919-222-2222"
    },
    {
      "First_Name": "stu",
      "Last_Name": "gotz",
      "Email": "sgotz@test.com",
      "Phone_Number": "919-333-3333"
    },
    {
      "First_Name": "cara",
      "Last_Name": "simmons",
      "Email": "csimms@test.com",
      "Phone_Number": "919-444-4444"
    },
    {
      "First_Name": "tom",
      "Last_Name": "ford",
      "Email": "tford@test.com",
      "Phone_Number": "919-555-5555"
    },
    {
      "First_Name": "jerry",
      "Last_Name": "black",
      "Email": "jblac@test.com",
      "Phone_Number": "919-666-6666"
    },
    {
      "First_Name": "susan",
      "Last_Name": "brown",
      "Email": "subro@test.com",
      "Phone_Number": "919-777-7777"
    },
```

```
"SASJSONExport": "1.0 NOKEYS PRETTY",
"SASTableData+Employees under 45": [
  [
    "sue",
    "stevens",
    258432,
    57,
    "07/01/1989",
    "sales"
  ],
  [
    "sally",
    "johnson",
    443767,
    48,
    "10/01/1999",
    "manager"
  ],
  [
    "max",
    "hart",
    793424,
    50,
    "03/01/1998",
    "sales"
  ],
  [
    "drew",
    "williams",
    347783,
    63,
    "09/01/1988",
    "manager"
  ],
  [
    "eric",
    "stuart",
    113547,
    48,
    "11/01/2000",
    "programmer"
  ]
]
}
```


Output 2. Portions of the JSON Output File Created with PROC JSON Using Options to Control the Output

Just a few options change the entire format of the JSON output file and give the user control of how the output file will look. This is extremely useful for applications that have very specific requirements for how the JSON must be formatted.

CONTROLLING CONTAINERS WITH THE WRITE STATEMENT

JSON itself consists of two types of data structures: arrays and objects. These are containers. In JSON, an array is opened and closed with a bracket [] and contains a list of values separated by a comma. An object is opened and closed with a brace {} and contains a list of key:value pairs. Keys and their corresponding values are separated with a colon, and each key:value pair within the object is separated by a comma.

The WRITE VALUES statement and the WRITE OPEN/CLOSE statements allow the user to open, close, and nest containers in the JSON output file as well as write separate values to the JSON output file.

A note about how implicit containers work with PROC JSON: In order for the JSON to be valid, it must all be included in a top-level container at the very least. When the user does not explicitly specify which type of container to open at the top level, PROC JSON will choose what type of container to open. If the EXPORT statement is the first statement after the PROC JSON statement, the top-level container is a JSON object. However, if the NOSASTAGS option is specified in either the PROC JSON statement or the EXPORT statement, the top-level container is a JSON array. PROC JSON will automatically close the implicitly opened top-level container. If the WRITE VALUES statement is the first statement after the PROC JSON statement, PROC JSON opens a JSON object as the top-level container, and likewise, PROC JSON will close the implicitly opened top-level container.

The previous two examples give a good indication of how this works. In the first example, the EXPORT statement is the first statement after the PROC JSON statement, and the top-level container is an object. (It opens and closes with a brace {} and contains a comma-separated list of key:value pairs.) In the second example, once again the EXPORT statement is the first statement after the PROC JSON statement, but this time, the NOSASTAGS option is used, so the top-level container is an array. (It opens and closes with brackets [] and contains a comma-separated list of values.)

The following use case will show how the WRITE VALUES and WRITE OPEN/CLOSE statements can be used to control the containers in the JSON output file.

Use Case B

- The user needs to create a JSON file to use in an application that has very specific requirements regarding the JSON format.
- The requirements are:
    - The top-level container must be an array.
    - Title and Description are two attributes that must come before the actual data from the table.
    - The data from the table should be in an array container.
    - The data comes from the employee SAS® data set from the second example.

All these requirements can be met using PROC JSON's WRITE statements. Here is the PROC JSON code that can be used to create the desired JSON output file:

```
proc json out="sample3.json" pretty nosastags;
  write open array;  /*open top level container as an array*/

    write open object;    /*Open an object container for the data*/
      write values "Title" "Employees";  /*These values will be written to the output file*/
      write values "Description" "Information about employees for the company";

      write open array;  /*Open an array container to and export the data set withing it*/
        export employees / fmtdatetime;
      write close;

    write close;

  write close;  /*Always close any container that has been explicitly opened*/
run;
```

With this SAS® code using the PROC JSON statement, the resulting JSON output file will look like this:


```
[
  {
    "Title": "Employees",
    "Description": "Information about employees for the company",
    "Employees": [
      {
        "First_Name": "adam",
        "Last_Name": "linker",
        "ID_Number": 123456,
        "Age": 23,
        "Start_Date": "04/01/2017",
        "Job_Title": "programmer"
      },
      {
        "First_Name": "bob",
        "Last_Name": "smith",
        "ID_Number": 584932,
        "Age": 33,
        "Start_Date": "08/01/2004",
        "Job_Title": "sales"
      },
```

```
      {
        "First_Name": "eric",
        "Last_Name": "stuart",
        "ID_Number": 113547,
        "Age": 48,
        "Start_Date": "11/01/2000",
        "Job_Title": "programmer"
      },
      {
        "First_Name": "dan",
        "Last_Name": "baker",
        "ID_Number": 567821,
        "Age": 40,
        "Start_Date": "04/01/2004",
        "Job_Title": "programmer"
      },
      {
        "First_Name": "ella",
        "Last_Name": "walsh",
        "ID_Number": 983753,
        "Age": 26,
        "Start_Date": "05/01/2010",
        "Job_Title": "sales"
      }
    ]
  }
]
```


Output 3. Portions of the JSON Output File Created with the PROC JSON Example Using the WRITE Statement to Control Containers

The WRITE OPEN statement will open a container and the WRITE CLOSE statement will close it. The top-level container in this example is an array (enclosed in brackets) because of the WRITE OPEN ARRAY statement. Next an object is opened with WRITE OPEN OBJECT. This is an example of a nested container (a container inside another container). The two attributes that were required to come before the data table are written to the output file with the WRITE VALUES statements. Another container is opened, this time an array where the exported data set will be. Lastly, all the opened containers are closed. When using WRITE OPEN to control containers, users must always remember that a corresponding WRITE CLOSE statement must be included for any container that is explicitly opened, or it will result in an error.

CREATING HIERARCHICAL DATA USING PROC JSON

A lot of architectures require JSON data in a hierarchical format. What this means is the data is arranged in a series of "levels" in the JSON file. For example, if a user had data of regions in North America, it could be arranged in a hierarchical fashion such as:

- Continent
    - Country
        - State/Province
            - City

Continent would be the highest level of the hierarchy. Next would be Country, then State/Province, and finally City.

It is possible to get JSON data into this form using PROC JSON by utilizing the WRITE statements to control the containers as shown above. The hierarchical structure is shown in the JSON output below with North America being in the outermost container as the highest level of the hierarchy.

Next, you can see the countries within North America in the second level of the hierarchy enclosed in a container nested within the top-level container. State/Province is in a level below that (North Carolina and South Carolina), and the lowest level is City, which shows the cities in North Carolina and South Carolina. Each container opened within another represents a new level in the hierarchy.

Below is the desired JSON output for a list of cities in North Carolina and South Carolina with populations greater than 100,000:

```
{
  "North America": {
    "United States of America": {
      "North Carolina": [
        [
          "Charlotte",
          827097
        ],
        [
          "Raleigh",
          451066
        ],
        [
          "Greensboro",
          285342
        ],
        [
          "Durham",
          257636
        ],
        [
          "Winston-Salem",
          241218
        ],
        [
          "Fayetteville",
          201963
        ],
        [
          "Cary",
          159769
        ],
        [
          "Wilmington",
          115933
        ],
        [
          "High_Point",
          110268
        ]
      ],
      "South Carolina": [
        [
          "Columbia",
          133803
        ],
        [
          "Charleston",
          132609
        ],
        [
          "North_Charleston",
          108304
        ]
      ]
    },
    "Mexico": null,
    "Canada": null
  }
}
```


Output 4. JSON Output File Generated in a Hierarchical Format with PROC JSON

The SAS® code to create this output requires a series of nested containers to be opened. The top-level container is opened and North America (the highest level in the hierarchy) is written to the JSON output file. Each time an object is opened using WRITE OPEN OBJECT, that represents a new level below the previous.

In the second level, which represents the Country, United States of America is written to the output file as well as Mexico and Canada. For the purposes of this example, only United States will have data in the level below.

The third level (State) is created by opening a third object within the Country level. In the third level, arrays are opened as the fourth level to output the data for both the North Carolina and South Carolina data sets. The WHERE= option outputs only the cities within the data sets that have a population greater than 100,000. The NOKEYS option suppresses the variable names.

Below is the SAS® code that was used to create that JSON output:

```
proc json out="hierarchy.json" pretty nosastags;
  write open object;  /*Top-level container for Level 1 - Continent*/
    write values "North America";

    write open object;  /*Level 2 - Country*/
      write values "United States of America";

      write open object;  /*Level 3 - State*/
        write values "North Carolina";

          write open array;  /*Level 4 - City*/
            export x.north_carolina_cities (
                                          where=(population > 100000) ) / nokeys;
          write close;  /*Close Level 4*/

        write values "South Carolina";

          write open array;  /*Also Level 4 - City*/
            export x.south_carolina_cities (
                                          where=(population > 100000) ) / nokeys;
          write close;  /*Close Level 4*/

      write close;  /*Close Level 3*/

      write values "Mexico" NULL;
      write values "Canada" NULL;

    write close;  /*Close Level 2*/

  write close;  /*Close top-level (Level 1)*/

run;
```

CONCLUSION

For SAS® users who need a way to convert SAS® data sets into JSON or create their own unique, customized JSON from scratch, there is no better and easier solution than using PROC JSON. It gives the user control of the JSON output file through the utilization of options as well as the ability to control containers, write directly to the output file, and choose exactly what to include or not include in the resulting JSON file.

The options provided by PROC JSON are powerful because of the ease of use and versatility. A user can format an entire JSON output file with just a few simple options. There is no need to manually write or edit a JSON file.

Along with the WRITE OPEN statement, which allows users to control the containers in the JSON output file, the WRITE VALUES statement gives users customization beyond just the ability to output data in SAS® data sets by giving the user the ability to write custom information to the JSON output file.

Organizing data in a hierarchical fashion in JSON is a common way of representing the data. PROC JSON gives the user the ability to create this type of structure in the resulting output file using the WRITE OPEN and WRITE CLOSE statements to control the containers.

The features of PROC JSON combined make it a great tool for SAS® users that want to get the most out of their JSON output.

REFERENCES

SAS® Institute Inc. 2017. "JSON Procedure" In Base SAS® 9.4 Procedures Guide, Seventh Edition. Cary, NC: SAS® Institute Inc. Available at https://documentation.sas.com/?docsetId=proc&docsetTarget=p0ie4bw6967jg6n1iu629d40f0by.htm&docsetVersion=9.4&locale=en (accessed January 31, 2019).

RECOMMENDED READING

- PROC JSON Tip Sheet at https://support.sas.com/rnd/base/Tipsheet_PROC_JSON.pdf
- JSON category of sasCommunity.org Planet at http://www.sascommunity.org/planet/blog/category/json/

CONTACT INFORMATION

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

Adam Linker
919-531-1940
Adam.Linker@sas.com