Copying Data Between SAS ® and JSON Files

Bruce Gilsen

Federal Reserve Board

February 2021

Copying Data Between SAS ® and JSON Files

Bruce Gilsen, Federal Reserve Board

Bruce Gilsen has worked at the Federal Reserve Board for 37 ½ years. He spends most of his time as an in-house SAS consultant.

His Bruce Force fantasy baseball team will battle for its 6th league title in 32 years in 2021. If there's a season.

Topics

- Introduction
- Copy SAS data set to a JSON file: simple examples
- Copy JSON files into SAS
  - Non-nested vs nested JSON files
  - Reading JSON files into SAS: automating the process
  - Examples 1-6
- JSONL files
- JSONPP DATA step function

Introduction

- JavaScript Object Notation (JSON)
  - Open standard file format and data interchange format
  - Used for some of same purposes as XML
  - JSON info readily available on internet
- Starting in SAS 9.4
  - Copy SAS data sets to JSON files w/PROC JSON
- Starting in SAS 9.4TS1M4
  - Copy JSON files to SAS data sets w/JSON engine

Introduction

- Examples in this presentation
  - Use Linux
  - Apply to both Linux, Windows SAS
- This presentation: basic information, simple examples
- SAS Institute documentation
  - More details, many additional features

Copy SAS data set to JSON file: Example 1

- Data set ONE (DATE is a SAS date)

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

- Copy ONE to JSON file /my/home/m1xxx00/test1.json

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

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"
  }
]
```

SAS data set:

country	city	income	date
usa	chicago	100	20201001
usa	cleveland	200	20201101
canada	montreal	300	20201201

Common PROC JSON options (defaults underlined)

- PRETTY (vs. NOPRETTY): JSON file in human-readable format w/indentation, multiple lines
- NOPRETTY: entire JSON file one long record
- NOSASTAGS (vs. SASTAGS): suppress SAS metadata at top of JSON file. Else, JSON file has } at bottom, this at top:

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

- [NO]FMTCHARACTER, [NO]FMTDATETIME, [NO]FMTNUMERIC: use existing formats when write variables to JSON file?
- TRIMBLANKS vs. NOTRIMBLANKS: remove trailing blanks from character data written to JSON file
- PRETTY, NOSASTAGS in Example 1

Copy SAS data set to JSON file: Example 2

- Include SAS metadata at top of JSON file
- Data set ONE (DATE is a SAS date)

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

- Copy ONE to JSON file /my/home/m1xxx00/test2.json

```
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"
    }
  ]
}
```


country	city	income	date
usa	chicago	100	20201001
usa	cleveland	200	20201101
canada	montreal	300	20201201

Copy SAS data set to JSON file: Example 3

- Write JSON file as one long record
- Data set ONE (DATE is a SAS date)

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

- Copy ONE to JSON file /my/home/m1xxx00/test3.json

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

SAS data set:

```
country city     income date
usa     chicago   100  20201001
usa     cleveland 200  20201101
canada  montreal  300  20201201
```

JSON file, one long record:

```
[{"country":"usa","city":"chicago","income":100,"date":"20201001"}
   ,{"country":"usa","city":"cleveland","income":200,"date":"20201
   101"},{"country":"canada","city":"montreal","income":300,"date"
   :"20201201"}]
```

Copy SAS data set to JSON file: Example 4

- Don’t use associated SAS date formats when write SAS date values (DATE in this case)
- Data set ONE (DATE is a SAS date)

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

- Copy ONE to JSON file /my/home/m1xxx00/test4.json

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

JSON file:

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "date": 22189
  },
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "date": 22220
  },
  {
    "country": "canada",
    "city": "montreal",
    "income": 300,
    "date": 22250
  }
]
```

SAS data set:

country	city	income	date
usa	chicago	100	20201001
usa	cleveland	200	20201101
canada	montreal	300	20201201

Copy SAS data set to JSON file: Example 5

- Don’t remove trailing blanks from character variables
- Data set ONE (DATE is a SAS date)

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

- Copy ONE to JSON file /my/home/m1xxx00/test5.json

```
proc json out="/my/home/m1xxx00/test5.json" pretty nosastags
  notrimblanks;
  export one;
run;
```

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"
  }
]
```

SAS data set:

country	city	income	date
usa	chicago	100	20201001
usa	cleveland	200	20201101
canada	montreal	300	20201201

PROC JSON: beyond these examples

- Simple examples could suffice for basic usage
- PROC JSON documentation: much more detailed info
  - Controlling containers in JSON file
  - Organizing data in nested fashion
  - More complex output operations

Copy JSON files into SAS

- Non-nested vs nested JSON files
- Read JSON files into SAS: automate the process
- Examples 1-6

Non-nested vs nested JSON files

- Non-nested JSON files
  - Simply organized: don’t have nested levels
  - Example: test1.json in prior section
  - Layout sort of corresponds to rectangular SAS data set
  - Much simpler to read into SAS
- Nested JSON files
  - Keys are nested
  - Layout doesn’t correspond to rectangular SAS data set

Non-nested JSON file test1.json

- Simply organized: no nested levels, layout ~rectangular SAS data set
- Each object corresponds to observation in SAS data set

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

Nested JSON file test2.json: INCOMETAX, SALESTAX keys nested inside TAXES key

```
[
  {
    "country": "usa",
    "city": "chicago",
    "income": 100,
    "taxes": {
              "incometax": 10,
              "salestax": 5
     }
  },
  {
    "country": "usa",
    "city": "cleveland",
    "income": 200,
    "taxes": {
              "incometax": 20,
              "salestax": 10
     }
  }
]
```

Non-nested vs nested JSON files: Ordinal variables

- Variables names start with ORDINAL_
- When read JSON file, provide relationship between generated data sets
- Non-nested JSON files
  - Can drop ordinal variables
- Nested JSON files
  - Can sometimes use ordinal variables to merge generated data sets, get meaningful result

Reading JSON files into SAS: automating the process

- Paper objective: automate reading JSON files into SAS
- Area of ongoing research
- Examples reflect results to date, update in future
- Suggestions greatly appreciated!
- Examples
  - Examples 1-3: JSON files read, relatively automated
  - Example 4: JSON files need 1 change to be relatively automated
  - Example 5: Read file differently to be somewhat automated
  - Example 6: Need specific knowledge about JSON file to get desired result, hard to automate

Example 1: read non-nested JSON file test1.json into SAS

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

- Specify SAS library where SAS files is copied (libref xxx)
- Remove existing SAS files w/PROC DATASETS (omit if empty)
- Access JSON file test1.json w/read-only privileges (libref ex31)
- Copy JSON file contents to SAS data sets in XXX library, data sets ALLDATA, ROOT created
- List data sets in XXX w/PROC DATASETS

```
libname xxx '/my/home/m1xxx00/example31';
proc datasets library=xxx kill; run;quit;
libname ex31 json '/my/home/m1xxx00/test1.json';
proc copy in=ex31 out=xxx; run;
proc datasets lib=xxx;
run;quit;
```

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

- JSON file not nested
- To create final data set, just drop ORDINAL_ variables from ROOT

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

ROOT

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

Example 2: read nested JSON file test2.json with 2 levels into SAS

- INCOMETAX, SALESTAX keys nested inside TAXES key
- Not directly analogous to SAS data set
- First object of test2.json shown below

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

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

- Specify SAS library where SAS files is copied (libref xxx)
- Remove existing SAS files w/PROC DATASETS (so PROC SQL generates data set name list correctly)

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

- Access JSON file test2.json w/read-only privileges (libref ex32)
  - AUTOMAP=CREATE: generate JSON map
  - MAP=: where JSON map file is written
  - ORDINALCOUNT=ALL: generate all possible ordinal variables (provides relationship between generated data sets). Default=2, need all possible variables to merge generated data sets.

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

- Copy JSON file contents to SAS data sets in XXX library, create data sets ALLDATA, ROOT, TAX
- List data sets in XXX w/PROC DATASETS

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

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

- Read DICTIONARY table, create macro variable
  - All SAS data sets in XXX except ALLDATA, space-separated
  - Each data set name preceded by xxx
  - &ALL_BUT_ALLDATA in this example: xxx.ROOT xxx.TAX
- Merge data sets 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;
```

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

FINALDATA2

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 nested JSON file test3.json with 3 levels into SAS

- Not all values need be present: MONTREAL doesn't have estimated income tax or 2nd car tax payment

```
{
   "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
                           }
     }
  }      ]
```

- Repeat previous example’s steps
  - Specify SAS library and JSON file, copy JSON file contents to SAS data sets, list data sets

```
libname xxx '/my/home/m1xxx00/example33’;
proc datasets library=xxx kill; run;quit;
libname ex33 json '/my/home/m1xxx00/test3.json’
map='user33.map' automap=create ordinalcount=all;
proc copy in=ex33 out=xxx; run;
proc datasets lib=xxx;run;quit;
```

- Data sets ALLDATA, ROOT, TAX, TAX_CARTAX, TAX_INCOMETAX created

- One object of test3.json

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

- What output data sets contain (other than ORDINAL variables)
  - ROOT: COUNTRY, CITY, INCOME
  - TAX: SALESTAX
  - TAX_CARTAX: CARTAX_H1, CARTAX_H2
  - TAX_INCOMETAX: INCOMETAX_ESTIMATED, INCOMETAX_WITHHELD

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	.

- Repeat prior example's steps, create macro variable w/SAS data sets in XXX except ALLDATA (data set name preceded by xxx) space-separated, merge data sets by ORDINAL_ROOT
- Macro variable value:

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;
```

- Final data set FINALDATA3

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 JSON file with name collisions and retained variables

- test4.json

- 1st object of 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" }
            ]
         }
       ]
    },
```

- JSON file differences from prior examples
  - 2 different TYPE properties. Name collision interferes w/merging values from component data sets.
  - TYPE (of employee) property used for 4 data records, NAME + AGE properties used for 2 data records

Complete JSON file

```
[
    {
     "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" }
           ]
        }
      ]
    }
  ]
```

- Repeat first part of code from prior examples

```
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;
```

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

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

.....

- Approach in documentation
  - Manually edit JSON map file, make multiple changes
- Our approach
  - Manually rename TYPE property for type of employee to TYPEEMP in JSON file
  - First merge INFO and INFO_PHONE by ORDINAL_INFO
  - Then merge that data set with ROOT by ORDINAL_ROOT
- To find name collisions
  - Check ALLDATA for variable names in more than one “P” variable (P1, P2, and P3 in this case)
  - “P” variables contain properties from JSON file in way that reflects how they’re nested
  - Macro in paper

```
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 first merge:

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 second merge:

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

Example 5: read ALLDATA data set

- test5.json
- Run same code, get 6 data sets with ORDINAL_ variables plus ALLDATA
- All ORDINAL_ variables = 1
- Read ALLDATA instead to create final SAS data set
  - In paper

```
{
  "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"
         }
     }
  }
}
```

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 = ?

MATH_Q1

Obs	ordinal_root	ordinal_quiz	ordinal_math	ordinal_q1	question	answer
1	1	1	1	1	5 + 7 = ?	12

MATH_Q2

Obs	ordinal_root	ordinal_quiz	ordinal_math	ordinal_q2	question	answer
1	1	1	1	1	12 - 8 = ?	4

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

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

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

SPORT_Q1

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

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 = ?
					…….			

- For obs when V=1, *highest non-blank P variable = Value*
  - e.g., in obs 4, V=1, P5=blank, P4=question, assign

    `question="# of NY Knicks titles?";`
- Very 1st variable (question in obs 4) indicates new observation
- Variables can be character or numeric, same variable can be both or have varying length
- If variable character in any non-ALLDATA data set, it's character, length is longest length in any non-ALLDATA data set

Example 6: custom coding required

- ALLDATA in LIBNAME Statement: JSON Engine documentation “The ALLDATA Data Set” section
- DATA step code used to read ALLDATA, create data set
  - Takes advantage of knowledge about ALLDATA contents
  - Example: P3 values used to make new variable, ITEM
  - Some cases: very hard to automate copying JSON file into SAS
  - Custom code that takes advantage of content knowledge might be best or only approach

- ALLDATA

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

- Final SAS 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	1

JSONL files

- Format growing in popularity
- Newline-delimited JSON a.k.a. JSONL or JSON Lines
- Each text line represents valid JSON object
- No hierarchical relationship between lines so not valid JSON
- Can convert JSONL file to JSON file in SAS DATA step
- Example: small JSONL 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"}
```

- JSONL file four.jsonl

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

- 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;
```

- JSONL file four.jsonl

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

- JSON file four.json

```
[{"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

- PROC JSON option PRETTY creates JSON file in human-readable format w/indentation, multiple lines
- JSONPP DATA step function copies single record JSON file to “pretty” JSON file
- JSON file test1.json from PROC JSON Example 3 above

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

- Copy single record JSON file to “pretty” JSON file

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

- “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

- Starting in SAS 9.4
  - Copy SAS data sets to JSON files w/PROC JSON
  - Often simple
- Starting in SAS 9.4TS1M4
  - Copy JSON files to SAS data sets w/JSON engine
  - Simple, hard, or very hard
  - Some can be automated
  - Area of ongoing research, appreciate help
- JSONL files
- JSONPP DATA step function

For more information, please contact

Bruce Gilsen
Federal Reserve Board, mail stop N-122
Washington, DC 20551
phone: 202-452-2494
e-mail: bruce.gilsen@frb.gov