

Working with JSON Files Using SAS

Susan Farmer

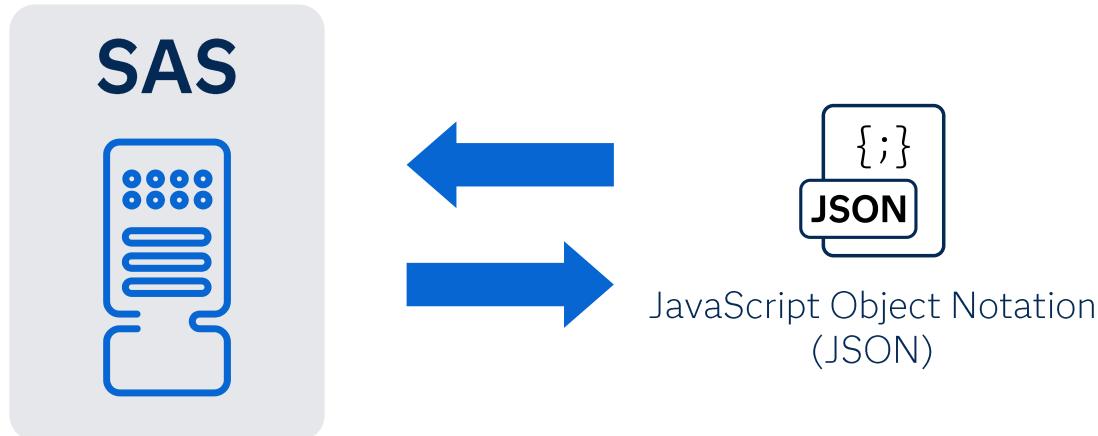
Principal Technical Training Consultant

California Remote Office

Susan.Farmer@sas.com

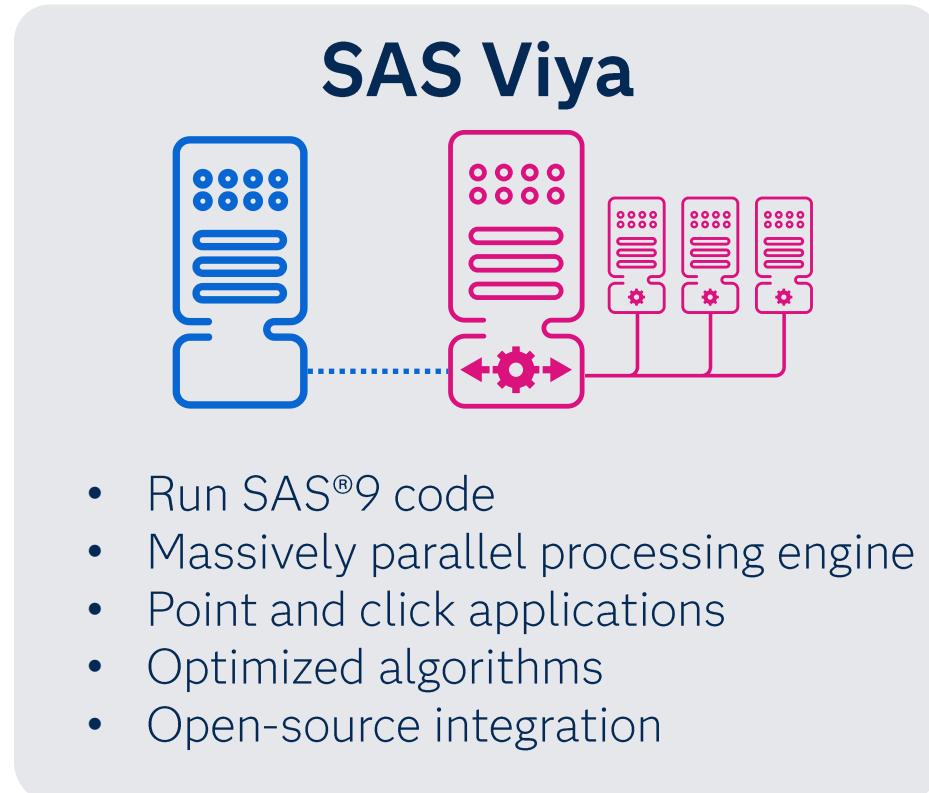


Working with JSON Files Using SAS!

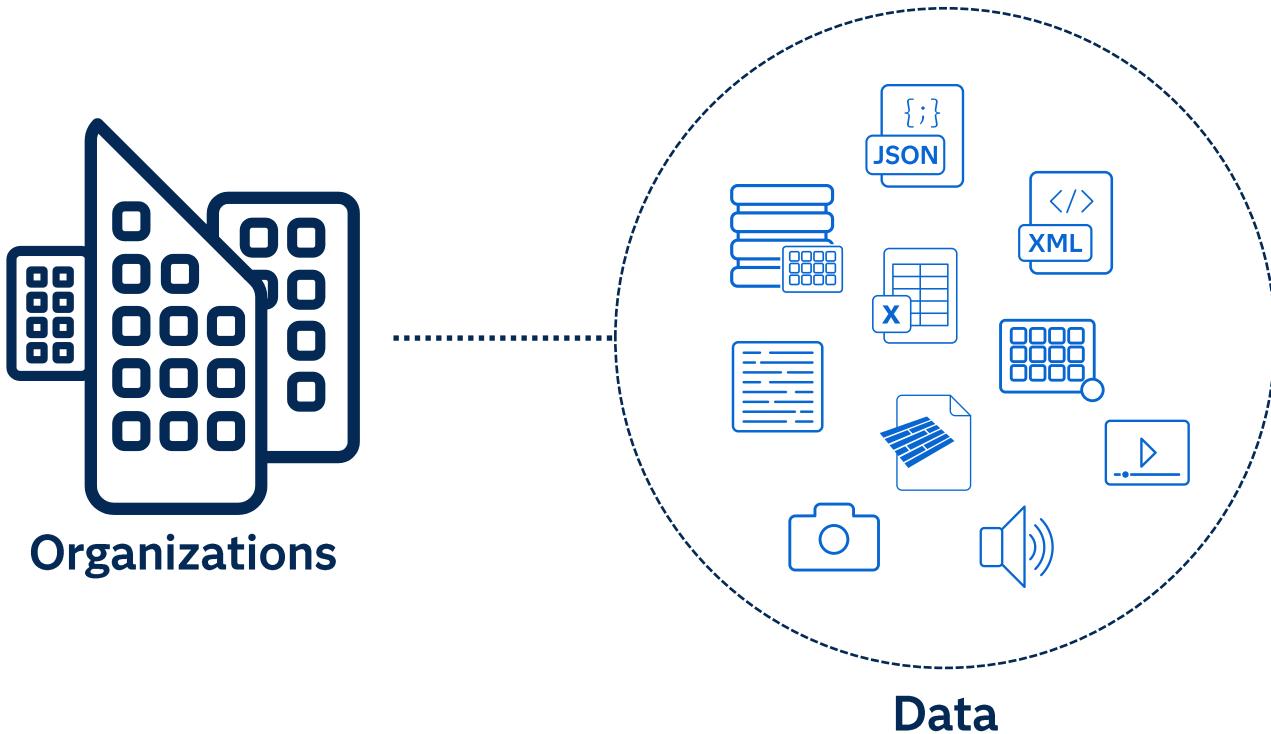




This engine is compatible with both **SAS®9** and **SAS Viya**.



Data Source Categories



Categories

Structured

Highly organized with a **predefined structure**

Easily machine-readable

Examples

Relational databases,
SAS tables, Excel
spreadsheets, parquet

Semi-structured

Some structure but less rigid than structured data

Moderately machine-readable

Examples

HTML, CSV, JSON, XML,
YAML, log files

Unstructured

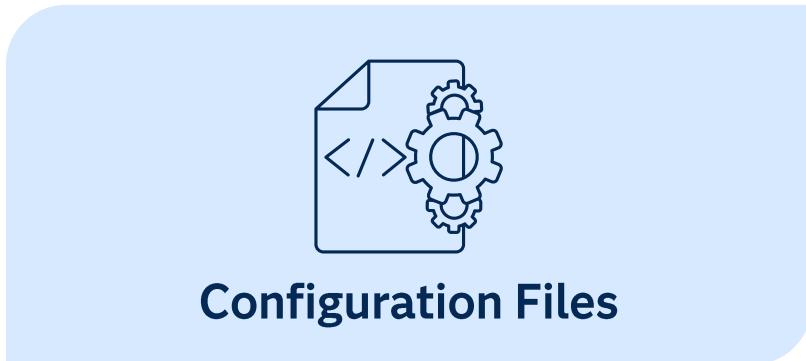
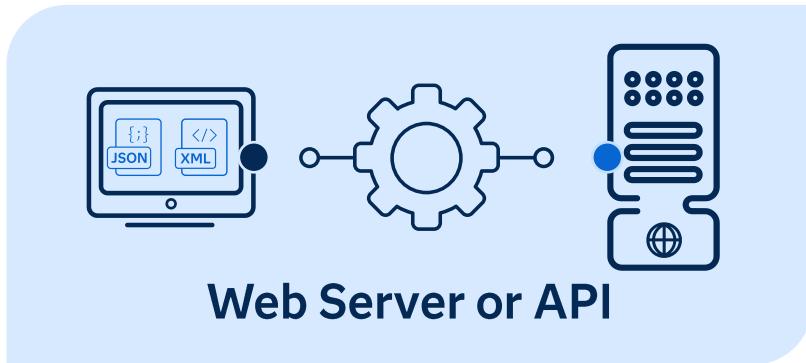
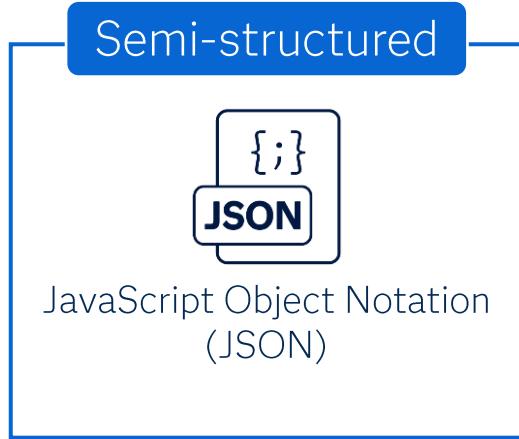
Lacks predetermined form or schema

Challenging for machines

Examples

Text documents, PDFs,
images, videos, emails

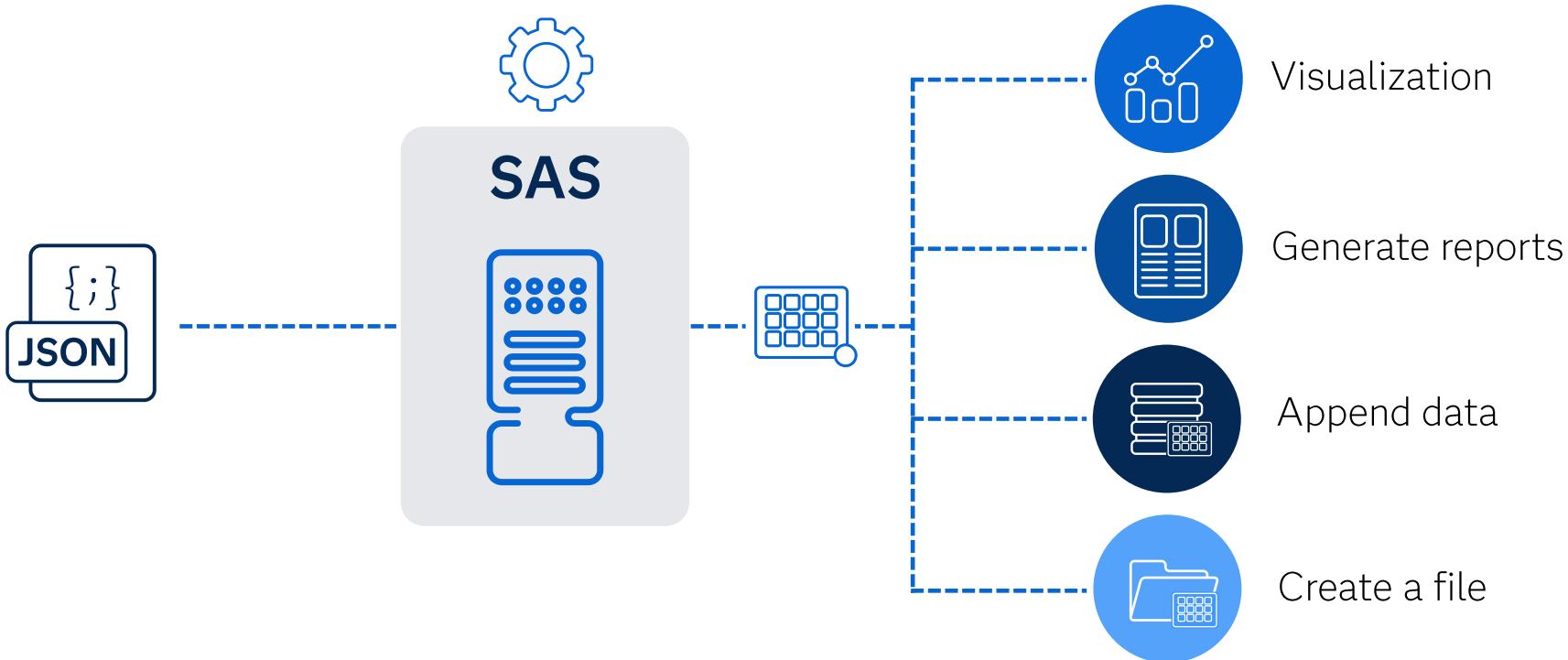
Working with JSON Files Using SAS



Extract & Load

Transform

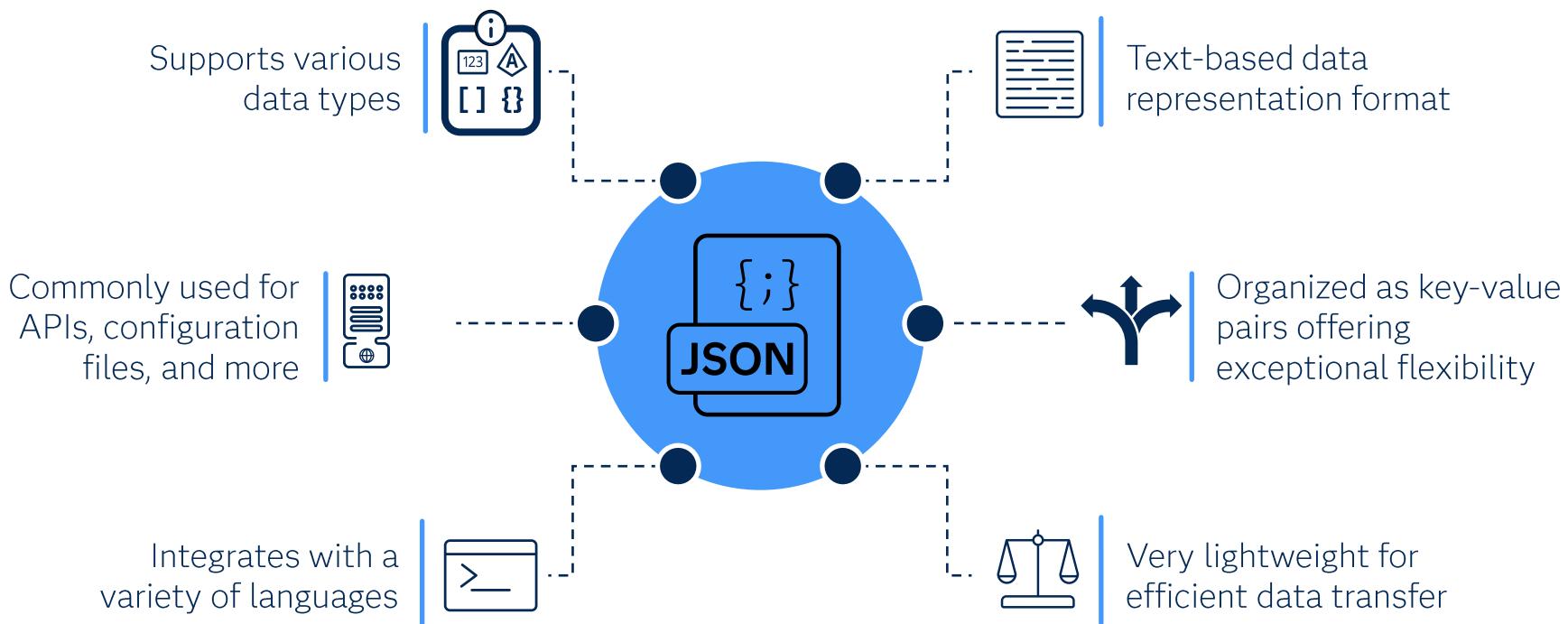
Analyze | Load



Working with JSON Files Using SAS

In this demonstration, we will provide a high-level overview of reading and writing JSON files using SAS.

What Is a JSON File?





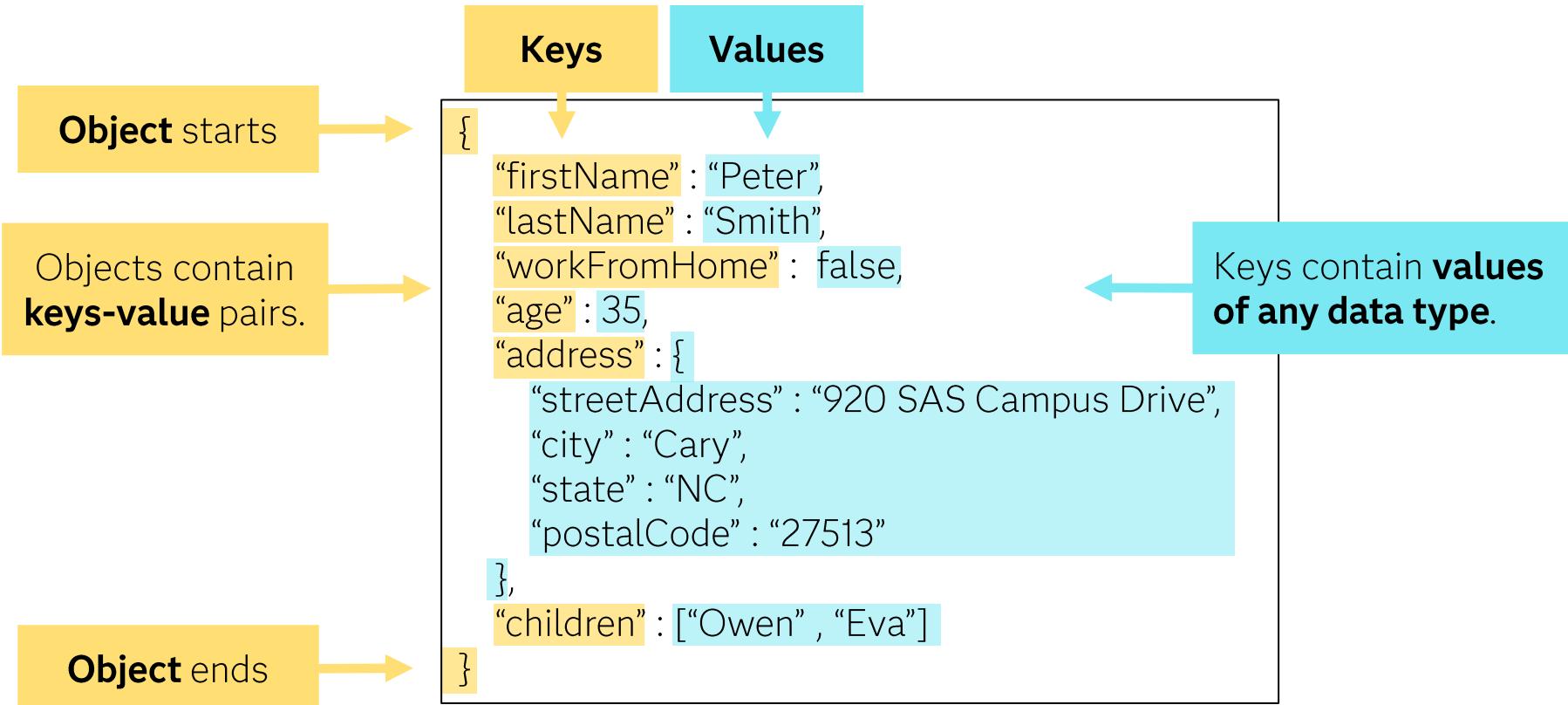
JSON data types

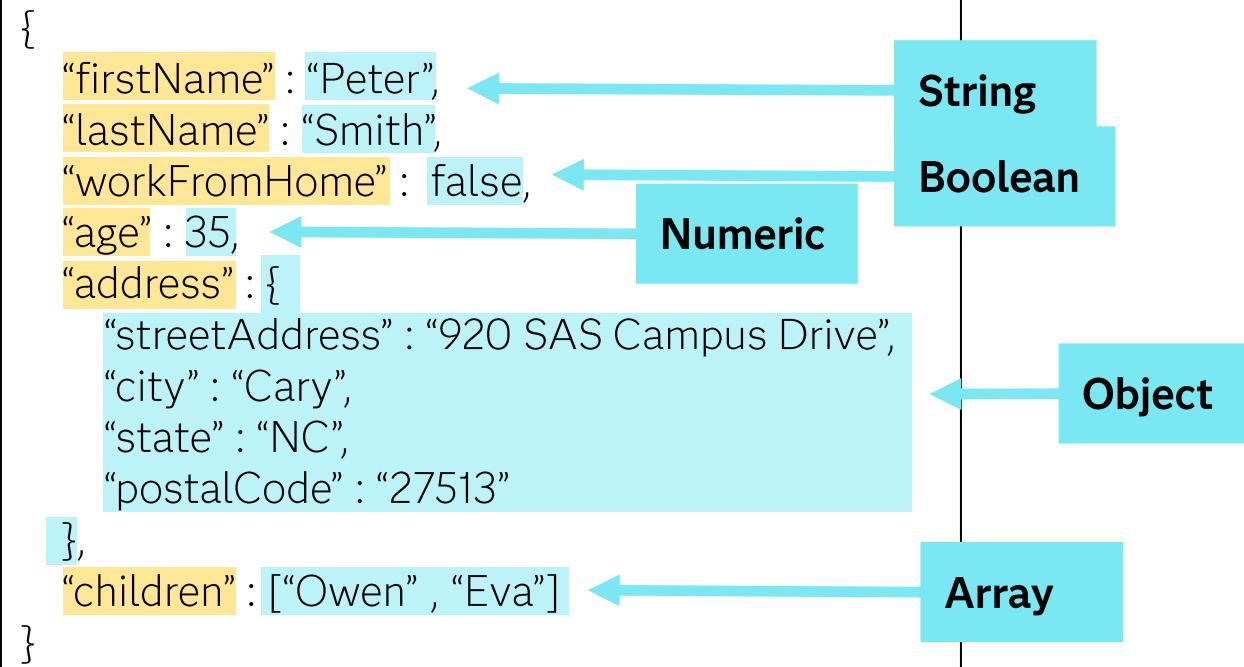
String	“Peter” “SAS” “Confidential”
Numeric	100 -10 2.1e5
Null	null
Boolean	true false
Array	[100, 200, 300] ["Peter", "SAS", null, true]
Object	{"key" : value} {"name" : "Peter", "age" : 35}



Any valid data type

JSON File Examples





Array starts

[

{

“firstName” : “Peter”,
“lastName” : “Smith”,
“age” : 35

}

,

{

“firstName” : “Fatima”,
“lastName” : “Patel”,
“age” : 28

}

,

{

“firstName” : “Apollo”,
“lastName” : “Papadopoulos”,
“age” : 45

}

Array ends

]

The JSON file contains
an **array of objects**.

```
[  
  {  
    1   "firstName": "Peter",  
        "lastName": "Smith",  
        "age": 35  
    },  
    {  
    2   "firstName": "Fatima",  
        "lastName": "Patel",  
        "age": 28  
    },  
    {  
    3   "firstName": "Apollo",  
        "lastName": "Papadopoulos",  
        "age": 45  
    }  
]
```

Each **distinct key** will be used as a **column name**.

firstName	lastName	age
-----------	----------	-----

Values

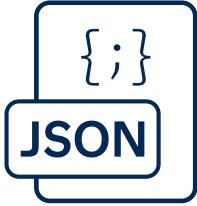
```
[  
  {  
    1   "firstName": "Peter",  
        "lastName": "Smith",  
        "age": 35  
    },  
    {  
      2   "firstName": "Fatima",  
          "lastName": "Patel",  
          "age": 28  
    },  
    {  
      3   "firstName": "Apollo",  
          "lastName": "Papadopoulos",  
          "age": 45  
    }]  
]
```

firstName	lastName	age
Peter	Smith	35
Fatima	Patel	28
Apollo	Papadopoulos	45

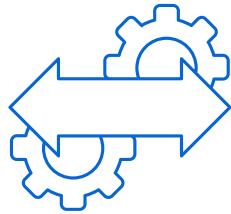
The **values for each key in the object** will be placed in a row **corresponding with the column name**.

Reading JSON Files using the JSON Engine

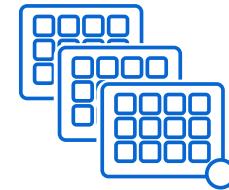




JSON File

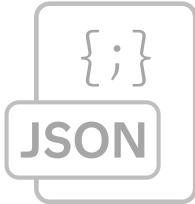


JSON LIBNAME Engine

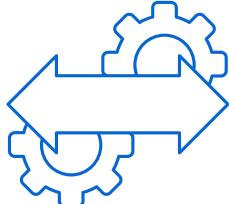


SAS Tables

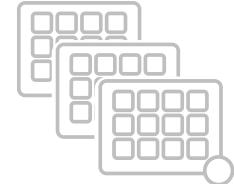
Available starting
in SAS 9.4M4



JSON File



JSON LIBNAME Engine



SAS Tables

```
LIBNAME libref JSON "filepath/filename.json";
```

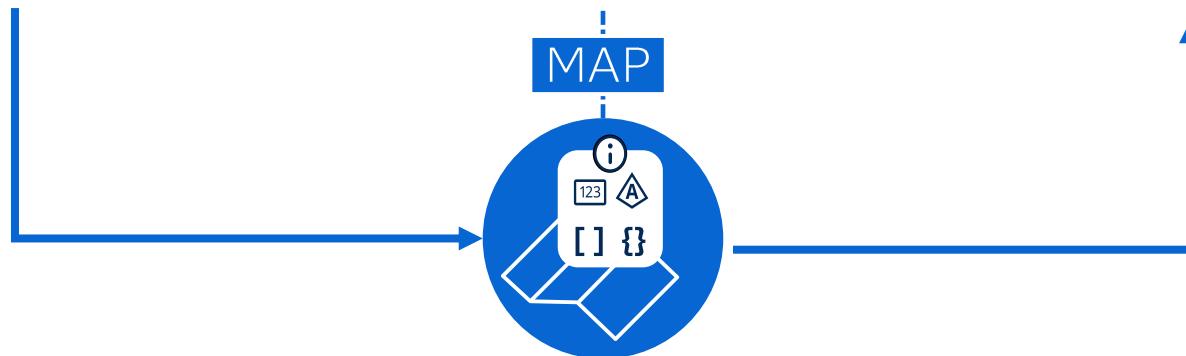
Specify the **file name** and **location** of the JSON file.

```
FILENAME fileref "filepath/filename.json";
```

Assign a **fileref** to the JSON file.

```
LIBNAME libref JSON FILEREF = fileref;
```

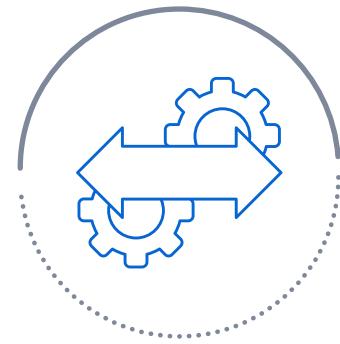
Use the **FILEREF=** option to specify the file reference name.



The JSON engine uses a **JSON map file** to describe the data in the specified JSON file.

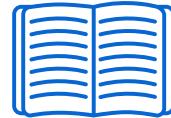
JSON Data Type		SAS
String	↔	Character
Numeric	→	Numeric
Null	→	SAS missing value
Boolean	→	Numeric 0 or 1
Array	→	SAS table
Object	→	SAS table

JSON Engine Considerations



Converts JSON data types

JSON data types will be converted to valid SAS data types.



Read-only engine

The JSON file is read only once, when the JSON engine LIBNAME statement is assigned.



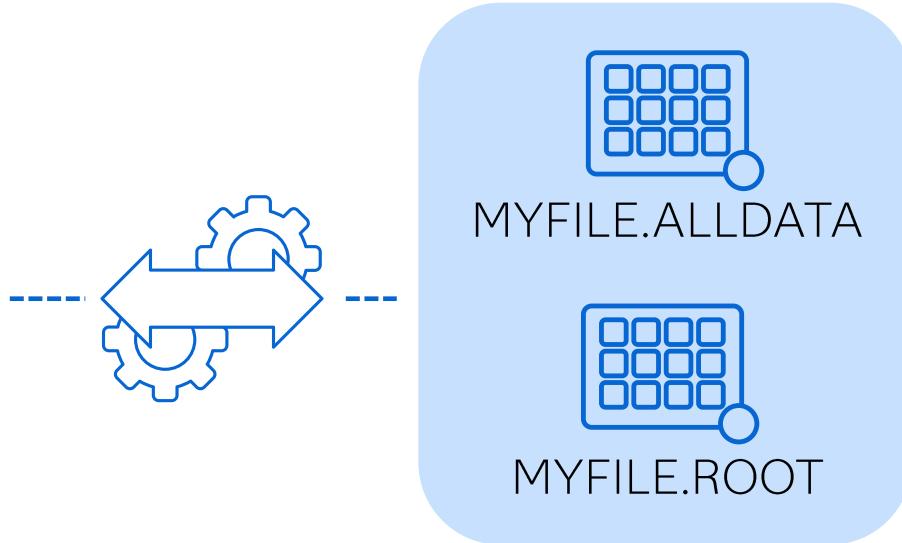
Requires valid JSON

The JSON engine requires complete, valid JSON structure to map the data into SAS.

Reading Simple JSON Files

person_info_simple.json

```
{  
  "firstName": "Peter",  
  "lastName": "Smith",  
  "workFromHome": false,  
  "age": 35,  
  "spouse": null,  
  "hireDate": "06/01/2020"  
}
```



```
filename myfile "&path./data/person_info_simple.json";
```

```
libname myfile JSON fileref=myfile;
```

MYFILE.ALLDATA

```
{  
    "firstName" : "Peter",  
    "lastName" : "Smith",  
    "workFromHome" : false,  
    "age" : 35,  
    "spouse" : null,  
    "hireDate" : "06/01/2020"  
}
```

P	P1	V	Value
1	firstName	1	Peter
1	lastName	1	Smith
1	workFromHome	1	false
1	age	1	35
1	spouse	1	null
1	hireDate	1	06/01/2020

Shows the **number of levels** for each key in the JSON file.

```
{
    "firstName": "Peter",
    "lastName": "Smith",
    "workFromHome": false,
    "age": 35,
    "spouse": null,
    "hireDate": "06/01/2020"
}
```

MYFILE.ALLDATA

P	P1	V	Value
1	firstName	1	Peter
1	lastName	1	Smith
1	workFromHome	1	false
1	age	1	35
1	spouse	1	null
1	hireDate	1	06/01/2020

P1-Pn are character columns that show **keys** for **each nested level**.

```
{  
    "firstName": "Peter",  
    "lastName": "Smith",  
    "workFromHome": false,  
    "age": 35,  
    "spouse": null,  
    "hireDate": "06/01/2020"  
}
```

Each key
contains a
value.

MYFILE.ALLDATA

P	P1	V	Value
1	firstName	1	Peter
1	lastName	1	Smith
1	workFromHome	1	false
1	age	1	35
1	spouse	1	null
1	hireDate	1	06/01/2020

Numeric data type
showing whether a
Value is available.

```
{  
    "firstName": "Peter",  
    "lastName": "Smith",  
    "workFromHome": false,  
    "age": 35,  
    "spouse": null,  
    "hireDate": "06/01/2020"  
}
```

MYFILE.ALLDATA

P	P1	V	Value
1	firstName	1	Peter
1	lastName	1	Smith
1	workFromHome	1	false
1	age	1	35
1	spouse	1	null
1	hireDate	1	06/01/2020

Character value that
shows the
untransformed
JSON value.

```
{  
  "firstName": "Peter",  
  "lastName": "Smith",  
  "workFromHome": false,  
  "age": 35,  
  "spouse": null,  
  "hireDate": "06/01/2020"  
}
```

MYFILE.ALLDATA

P	P1	V	Value
1	firstName	1	Peter
1	lastName	1	Smith
1	workFromHome	1	false
1	age	1	35
1	spouse	1	null
1	hireDate	1	06/01/2020

The ALldata table is useful for examining the file's structure, not analysis.

You can improve your program efficiency by **suppressing** the creation of the ALldata table.

```
{  
  "firstName": "Peter",  
  "lastName": "Smith",  
  "workFromHome": false,  
  "age": 35,  
  "spouse": null,  
  "hireDate": "06/01/2020"  
}
```

MYFILE.ALLDATA

P	P1	V	Value
1	firstName	1	Peter
1	lastName	1	Smith
1	workFromHome	1	false
1	age	1	35
1	spouse	1	null
1	hireDate	1	06/01/2020

```
libname myfile JSON fileref=myfile NOALLDATA;
```

The NOALLDATA option suppresses creation of the ALLDATA data set.

{

```
{"firstName": "Peter",  
"lastName": "Smith",  
"workFromHome": false,  
"age": 35,  
"spouse": null,  
"hireDate": "06/01/2020"}
```

}

Boolean data types are converted to 0 or 1.

Null data types are converted to SAS missing values.

MYFILE.ROOT

ordinal_root	firstName	lastName	workFromHome	age	spouse	hireDate
1	Peter	Smith		0	35	.

123

A

A

123

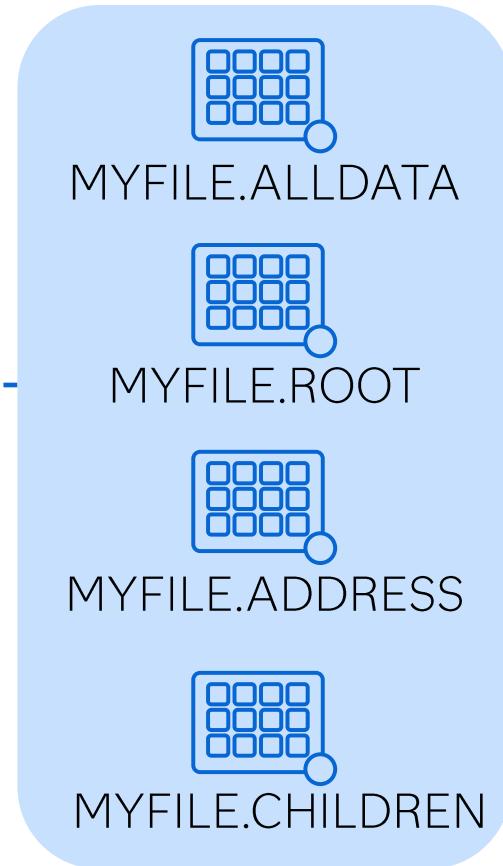
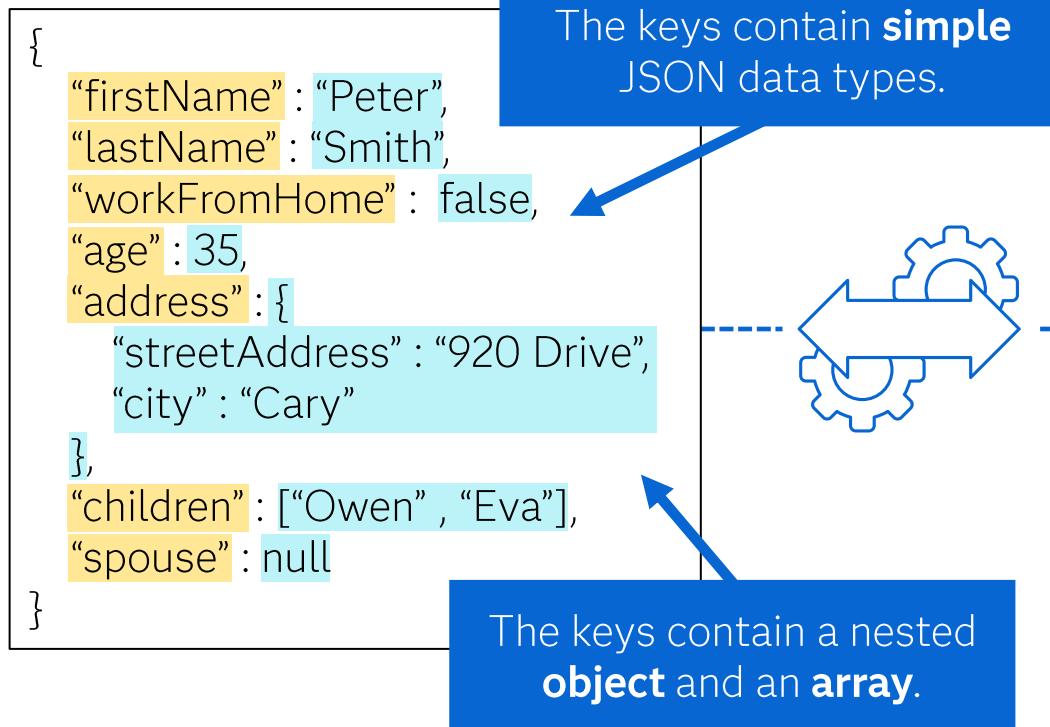
123

123

A

The **ordinal_root** column enables you to join tables.

Reading JSON Files with Nested Arrays and Objects



MYFILE.ALldata

```
{
    "firstName": "Peter",
    "lastName": "Smith",
    "workFromHome": false,
    "age": 35,
    "address": {
        "streetAddress": "920 Drive",
        "city": "Cary"
    },
    "children": ["Owen", "Eva"],
    "spouse": null
}
```

P	P1	P2	V	Value
1	firstName		1	Peter
1	lastName		1	Smith
1	workFromHome		1	false
1	age		1	35
1	address		0	
2	address	streetAddress	1	920 Drive
2	address	city	1	Cary
1	children		0	
2	children	children1	1	Owen
2	children	children2	1	Eva
1	spouse		1	null

MYFILE.ALldata

```
{
    "firstName": "Peter",
    "lastName": "Smith",
    "workFromHome": false,
    "age": 35,
    "address": {
        "streetAddress": "920 Drive",
        "city": "Cary"
    },
    "children": ["Owen", "Eva"],
    "spouse": null
}
```

All **first-level values**

P	P1	P2	V	Value
1	firstName		1	Peter
1	lastName		1	Smith
1	workFromHome		1	false
1	age		1	35
1	address		0	
2	address	streetAddress	1	920 Drive
2	address	city	1	Cary
1	children		0	
2	children	children1	1	Owen
2	children	children2	1	Eva
1	spouse		1	null

MYFILE.ALldata

```
{
    "firstName": "Peter",
    "lastName": "Smith",
    "workFromHome": false,
    "age": 35,
    "address": {
        "streetAddress": "920 Drive",
        "city": "Cary"
    },
    "children": ["Owen", "Eva"],
    "spouse": null
}
```

Nested object

P	P1	P2	V	Value
1	firstName		1	Peter
1	lastName		1	Smith
1	workFromHome		1	false
1	age		1	35
1	address		0	
2	address	streetAddress	1	920 Drive
2	address	city	1	Cary
1	children		0	
2	children	children1	1	Owen
2	children	children2	1	Eva
1	spouse		1	null

MYFILE.ALldata

```
{
    "firstName": "Peter",
    "lastName": "Smith",
    "workFromHome": false,
    "age": 35,
    "address": {
        "streetAddress": "920 Drive",
        "city": "Cary"
    },
    "children": ["Owen", "Eva"],
    "spouse": null
}
```



Nested **array**

P	P1	P2	V	Value
1	firstName		1	Peter
1	lastName		1	Smith
1	workFromHome		1	false
1	age		1	35
1	address		0	
2	address	streetAddress	1	920 Drive
2	address	city	1	Cary
1	children		0	
2	children	children1	1	Owen
2	children	children2	1	Eva
1	spouse		1	null

```
{  
  "firstName": "Peter",  
  "lastName": "Smith",  
  "workFromHome": false,  
  "age": 35,  
  "address": {  
    "streetAddress": "920 Drive",  
    "city": "Cary"  
  },  
  "children": ["Owen", "Eva"],  
  "spouse": null  
}
```

The **ROOT** table contains all the **first-level key-value** pairs.

MYFILE.ROOT

ordinal_root	firstName	lastName	workFromHome	age	spouse
1	Peter	Smith		0	35

```
{  
  "firstName": "Peter",  
  "lastName": "Smith",  
  "workFromHome": false,  
  "age": 35,  
  "address": {  
    "streetAddress": "920 Drive",  
    "city": "Cary"  
  },  
  "children": ["Owen", "Eva"],  
  "spouse": null  
}
```

The **ADDRESS** table contains the keys and values from the **nested object** in the “address” key.

MYFILE.ADDRESS

ordinal_root	ordinal_address	streetAddress	city
1	1	920 Drive	Cary

```
{  
  "firstName": "Peter",  
  "lastName": "Smith",  
  "workFromHome": false,  
  "age": 35,  
  "address": {  
    "streetAddress": "920 Drive",  
    "city": "Cary"  
  },  
  "children": ["Owen", "Eva"],  
  "spouse": null  
}
```

The **CHILDREN** table contains each element from the **nested array** in the “children” key.

MYFILE.CHILDREN

ordinal_root	ordinal_children	children1	children2
1	1	Owen	Eva

You can join the tables using the **ordinal_root** columns.

MYFILE.ROOT

ordinal_root	firstName	lastName	workFromHome	age	spouse
1	Peter	Smith	0	35	.

MYFILE.ADDRESS

ordinal_root	ordinal_address	streetAddress	city
1	1	920 Drive	Cary

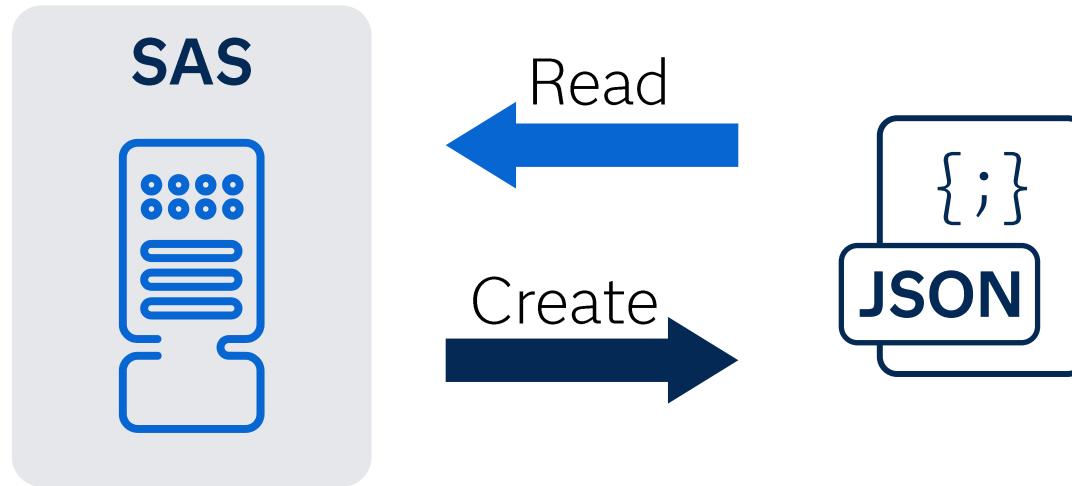
MYFILE.CHILDREN

ordinal_root	ordinal_children	children1	children2
1	1	Owen	Eva

Reading JSON Files with Nested Objects and Arrays

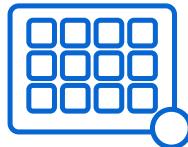
This demonstration illustrates how to read JSON files with nested arrays and objects into a single prepared table.

Creating a JSON File with SAS

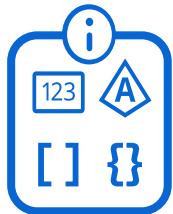




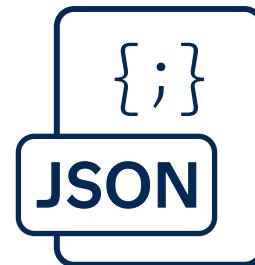
SAS



Export tables as an
array of objects



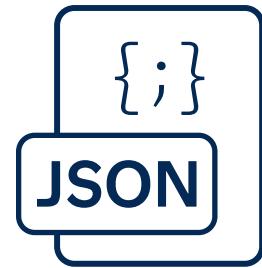
Write any valid
JSON data types



JSON

```
PROC JSON OUT=fileref | external-file <options>;  
EXPORT sas-data-set </options>;  
WRITE VALUES value(s) </options>;  
WRITE OPEN type;  
WRITE CLOSE;  
RUN;
```

Create 



The PROC JSON statement creates the **external JSON file**.

The EXPORT statement identifies the **table** to export.

The WRITE VALUES statement **writes simple JSON values** to the output JSON file.

The WRITE OPEN statement opens and nests a **JSON container (array or object)** in the output file.

The WRITE CLOSE statement **closes** a JSON container that is open in the output file.

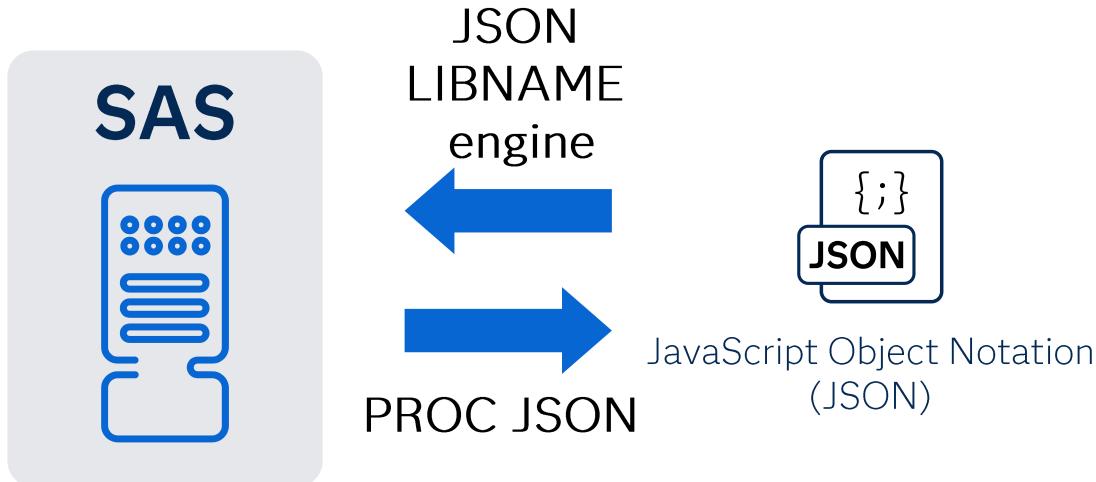
```
PROC JSON OUT=fileref | external-file <options>;
  EXPORT sas-data-set </options>;
  WRITE VALUES value(s) </options>;
  WRITE OPEN type;
  WRITE CLOSE;
RUN;
```

Creating JSON Files with PROC JSON

This demonstration illustrates how to use the JSON procedure to create JSON files.

Summary

Working with JSON Files Using SAS



This engine is compatible with both **SAS®9** and **SAS Viya**.

Working with JSON Files Using SAS

Susan Farmer

Principal Technical Training Consultant

California Remote Office

Susan.Farmer@sas.com

