PharmaSUG 2025 - Paper DS-336

Creating JSON Transport Data for Regulatory Life Sciences

Mary Dolegowski and Matt Becker, SAS

ABSTRACT

The capacity to convert data into a JSON structure has become indispensable as the use of APIs in regulatory life sciences becomes more critical. This capability facilitates interoperability, data sharing, and conformance among systems. In this context, we illustrate the process of converting SDTM (Study Data Tabulation Model) data to the Dataset-JSON format, which is a critical prerequisite for regulatory submissions to agencies such as the FDA.
We emphasize two SAS methods for this conversion: the CASL2JSON function in conjunction with PROC JSON and PROC CAS. Although Dataset-JSON is employed to illustrate these methods, the underlying processes are inherently data-agnostic, which allows for their application to other regulatory data structures. Seamless adaptation to changing industry standards and submission requirements is guaranteed by this adaptability.

INTRODUCTION

The life sciences industry has experienced a surge in demand for structured, machine-readable formats as regulatory agencies continue to modernize their data submission requirements. JSON (JavaScript Object Notation) is the foundation of this evolution. It is a lightweight, widely accepted data interchange format that is essential for regulatory compliance, scalable data exchange, and system interoperability. Particularly as APIs and digital submissions become increasingly prevalent in regulatory procedures, the capacity to efficiently convert clinical trial data into JSON has become indispensable.

The objective of this paper is to convert SDTM (Study Data Tabulation Model) datasets to the Dataset-JSON format, a standardized structure that is now under review by the FDA for electronic submissions. We present practical methods for executing this transformation by utilizing SAS capabilities, such as PROC JSON, and PROC CAS (SAS Viya only) with the CAS2JSON procedure. The techniques presented are intentionally designed to be data-agnostic, allowing for broader applicability across regulatory data formats and ensuring that organizations remain agile as submission standards evolve, even though Dataset-JSON is used as the illustrative target. The purpose of this document is not to predict how that transportation is going to take place but to go over the capabilities within SAS that allow for the easy formulation and creation using SDTM data and other data standards to create JSON dataset.

METHODS FOR CREATING JSON

This section is going to cover two methods for converting existing data in SAS into the JSON structure provided by CDISC for Dataset-JSON v1.1. Both examples will take a sample of SDTM AE data and convert it into Dataset-JSON. The focus is going to be on the data extraction and not the metadata extraction. The metadata extraction in both cases will be dependent on the database and so not covered in this paper.

PROC JSON

PROC JSON is a PROC that has been available within SAS for some time. It's an easy and convenient method for taking data that is already tabular and converting it into a JSON structure. It is an option available in all SAS environments as is all the code presented in this section.

The code in Program 1 takes a given dataset and works to pull out all of the required fields available and perform some general housekeeping in order to get the required elements structured in a way that will easily feed into the JSON schema.

```
/*just for testing to move the dataset in use from CAS to work*/
/* caslib _all_ assign; */

/*these will eventually come from the folder list selected and an interaction with the
CDISC API*/
%let dataset = ae_dm;
%let domain = AE;
%let label = Adverse Events;
%let start_time = %sysfunc(datetime(),e8601dz.);

%let IGD = %sysfunc(CATS(IG., &domain.));

/* moving the cas dataset into work */
data work.&dataset.;
      set public.&dataset.;
run;

/*pull out all the varibles in use in the dataset*/
proc contents data=work.&dataset.
      out = work.&dataset._contents
      NOPRINT;
run;

proc sort data=work.&dataset._contents;
      by VARNUM;
run;

/*set the observations to a macro var and the modified date to a macro var*/
proc SQL noprint;
      select distinct nobs into :nobs from work.&dataset._contents;
      select distinct strip(put(modate,e8601dz.)) into :modate from
work.&dataset._contents;
run;
%put &nobs.;
%put &modate.;

/*turn contents into a table to easy set up in JSON*/
data work.&dataset._contents;
      retain OID Name Label Type Length keySequence;
      set work.&dataset._contents (Keep = Name Type Length Label VARNUM);
      OID = CATS("IT.", "&domain.",".", Name);
      keySequence = VARNUM;
      drop VARNUM;
run;
```

Program 1. PROC JSON – Data Prep

The final section of code, Program 2, is where all the prep work in Program 1 gets applied. All of the structuring of the data into the scheme for Dataset-JSON occurs within a single PROC JSON. The structure of PROC JSON is simple, there is an out file, followed by the structure we want that output to follow. The ‘write open object’ creates an array and always needs a corresponding ‘write close.’ The ‘write values’ create a key-value pair or the start of a named array. Finally, we used the export option to select either the keys or values datasets, this is where all the work in Program 1 pays off as it helps to streamline the core components of the Dataset-JSON.

The code is really comprised of two sections, the metadata and the data itself. The metadata is going to need to be pulled from wherever the data is stored but can easily be automated, in this case we have most of those values filled in with place holders for this example. The data section utilizes the export option.

```
proc json out = "/nfsshare/sashls2/data/DatasetJson/datasetJSON_procjsonv11_ae.json";
      write open object;
             write values "datasetJSONCreationDateTime" "&start_time.";
             write values "datasetJSONVersion" "1.1.0";
             write values "fileOID" "www.sponsor.xyz.org.project123.final";
             write values "dbLastModifiedDateTime" "&modate.";
             write values "originator" "Sponsor XYZ";
             write values "sourceSystem";
                    write open object;
                           write values "originator" "blah";
                           write values "sourceSystem" "1.2.3";
                    write close;
             write values "studyOID" "xxx";
             write values "metaDataVersionOID" "xxx";
             write values "metaDataRef" "https://metadata.location.org/api.link";
             write values "itemGroupOID" "&IGD.";
             write values "isReferenceData" false;
             write values "records" &nobs.;
             write values "name" "&domain.";
             write values "label" "&label.";
             write values "columns";
                    write open array;
                           export work.&dataset._contents / keys nosastags;
                    write close;
             write values "rows";
                    write open array;
                           export work.&dataset. / nokeys nosastags trimblanks;
                    write close;
      write close;
run;
```

Program 2: PROC JSON - Conversion

As Program 1 and Program 2 show the process for converting an existing SAS dataset into Dataset-JSON is both repeatable and relatively simple.

PROC CAS – CASL2JSON

In this section we are going to cover a newer method for generating JSON data from within a SAS Viya environment, the CASL2JSON function within PROC CAS.

The primary difference with using the CASL2JSON function is that all of the work can be contained within a single PROC as shown in Program 3. There is some data management that needs to take place but then all of the required data is organized in a dictionary structure as show after comment, ‘sourceSystem’. The order for the Dataset-JSON is essentially reverse, as we have to create the internal arrays and then link them to the next level up. For example, the ‘sourceSystem’ array has to be created first to then be linked into the ‘DatasetJSON’ core array, in the line ‘DatasetJSON.sourceSystem = sourceSystem;’.

```
%let start_time = %sysfunc(datetime(),e8601dx.);

proc cas;

      set stdjson;

/*Set the lib and dataset*/

      libloc = "PUBLIC";
      table = "ae_dm";

/*Set some of the details for the metadata, this could also be pulled from the
repository*/
```

```
      domain = "ae";
      domain_label = "Adverse Events";
/*    current_time = &start_time.; */


/*General data extraction and organization to prepare the date for JSON formatting*/

      details_results = CATS("r_", domain);
      columninfo_results = CATS("r_", domain);

      table.tableInfo result = details_results /
      name = table, caslib = libloc;

/*Calculating the number of rows and oulling the last modified date from the dataset*/

      nrows = details_results.TableInfo[,"rows"];
      last_mod_date = details_results.TableInfo[1,"ModTimeFormatted"];

      format last_mod_date e8601dz.2;

      table.fetch result = fetch_results /
             table ={name = table, caslib = libloc},
             INDEX = true,
             to = nrows[1];

/*items = a description of all the variables aka column names and details*/

      table.columninfo result = columninfo_results /
             table ={name = table, caslib = libloc};

      i = 1;

      do row over columninfo_results.ColumnInfo;
             items[i].OID = CATS("IT.", domain, ".", row.Column);
             items[i].name = row.Column;
             items[i].label = row.Label;
             items[i].type = row.Type;
             items[i].length = row.FormattedLength;
             i = i + 1;
      end;

/*itemData - a list of all the values*/

      do row over fetch_results.fetch;
             new_row = getvalues(row);
             loc = new_row[1];
             itemDataTble[loc] = new_row;
      end;

/*sourceSystem*/
      sourceSystem.originator = "SAS";
      sourceSystem.sourceSystem = "Source System to be entered here";

/*Metadata*/
      DatasetJSON.datasetJSONCreationDateTime = "&start_time.";
      DatasetJSON.datasetJSONVersion = "1.1.0";
      DatasetJSON.fileOID = "File OID to be entered here";
      DatasetJSON.dbLastModifiedDateTime = last_mod_date;
      DatasetJSON.originator = "Sponsor name to be enetered here";
      DatasetJSON.sourceSystem = sourceSystem;
      DatasetJSON.studyOID = "Study OID to be entered here";
    DatasetJSON.metaDataVersionOID = "Meta Data Version OID to be entered here";
    DatasetJSON.metaDataRef = "Meta Data Ref to be entered here";
    DatasetJSON.records = nrows[1];
    DatasetJSON.name = domain;
    DatasetJSON.label = domain_label;
    DatasetJSON.columns = items;
    DatasetJSON.rows = itemDataTble;

      final = casl2json(DatasetJSON);

      final = compbl(final);

      file outfile "/nfsshare/sashls2/data/DatasetJson/datasetJSON_C2J_V11.json";
      print final;

run;
```

Program 3: PROC CAS - CASL2JSON Code

PROC CAS with the CASL2JSON function, is a useful method that uses the built-in dictionary capabilities in PROC CAS to help to organize data and then output a JSON file.

CONCLUSION

As the regulatory ecosystem becomes increasingly dependent on standardized data exchange and API-driven submissions, the ability to convert structured clinical data into formats like Dataset-JSON is no longer optional - it is a strategic imperative. The SAS-based methods detailed in this paper not only support compliance with current FDA submission formats but also provide a scalable, adaptable foundation for future data interoperability challenges. By adopting a data-agnostic, standards-ready approach, organizations can improve submission readiness, reduce manual effort, and stay aligned with the pace of regulatory innovation.

There are many ways to be able to deal with creating JSON Data within an analytics environment. All of them have their pros and cons. Each environment provides useful options and can be met with you at any stage within your submission journey. It is also important to note that one of the benefits of SAS Viya is its ability to interact with open source. As new and creative solutions for creating this JSON transport data becomes available there are multiple ways to be able to pick and choose the best parts of all and combine them into one.

This paper is just to showcase two simple methods for creating JSON data from SAS. Others, like Lex Jansen's work (link provided in references) provide another method. There are even more methods utilizing open-source languages like R or Python. Users will have to determine which will fit into their organization's workflows.

REFERENCES

CDISC. 2025. "Dataset-JSON". Accessed April 11, 2025. https://www.cdisc.org/standards/data-exchange/dataset-json

CDISC's Git Hub. 2025. "DataExchange-DatasetJson" Accessed APril 11, 2025. https://github.com/cdisc-org/DataExchange-DatasetJson/tree/publish_v11

Mary Dolegowski's Git Hub. 2025. "DS-336". Accessed April 11, 2025. https://github.com/mary-dolegowski/DS-336

CONTACT INFORMATION

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

Mary Dolegowski
SAS
mary.dolegowski@sas.com

Matt Becker

SAS
matt.becker@sas.com