Paper SI05

JSON for Submissions: Why You Should Care, What You Need to Know, and What to Do Now

Bill Qubeck, Sycamore Informatics, USA
Sridhar Vijendra, Sycamore Informatics, India

ABSTRACT

The FDA has published a Federal Register Notice seeking public comment on the potential adoption of CDISC Dataset-JSON v1.1 as a new exchange standard for regulatory submissions marking a pivotal moment in the evolution of clinical data delivery. Dataset-JSON, a structured, machine-readable format, is being considered as a long-term replacement for the legacy SAS XPT format. For sponsors, CROs, and technology providers, this signals a shift that will require new tools, revised processes, and a deeper focus on metadata governance and validation strategies.

This session explores why Dataset-JSON matters, what the format offers, and how organizations should prepare. Building on insights from the recent PHUSE/CDISC pilot, we will examine Dataset-JSON's key attributes, including support for complex metadata, native compatibility with modern programming languages (e.g., R, Python), and its potential to enable real-time validation and automation. We'll also address critical challenges, such as backward compatibility, integration into legacy systems, and validation concerns under GxP constraints.

Participants will walk away with a practical understanding of Dataset-JSON, its regulatory implications, and the steps required to adopt it; from metadata model alignment to the validation and traceability of generated outputs. With the FDA actively seeking community input, now is the time for industry stakeholders to weigh in, prepare internally, and contribute to shaping the standards that will define the future of clinical data submissions.

INTRODUCTION

For several decades, clinical trial data submitted to the U.S. Food and Drug Administration (FDA) have relied on the SAS Version 5 Transport (XPT) file format. Introduced in the 1980s, the SAS v5 XPT format became the de facto standard because of its simplicity, platform independence, and ability to be read by a variety of software systems. Its structure ensured consistent exchange of clinical and nonclinical data across different organizations and computing environments, a vital requirement in the regulated pharmaceutical industry. Over time, however, the limitations of the SAS v5 XPT format became increasingly evident. It supports only 8-character variable names, 40-character labels, and a restricted set of data types and encodings. As data complexity grew with larger trials, global operations, and increasing use of real-world evidence and nontraditional data sources these constraints made it difficult to represent modern clinical datasets efficiently and accurately.

To address these challenges, the Clinical Data Interchange Standards Consortium (CDISC) and the FDA explored modern, flexible alternatives. One such initiative led to the development of Dataset-JSON, a text-based, human- and machine-readable format aligned with contemporary data exchange practices. Dataset-JSON leverages JavaScript Object Notation (JSON), a lightweight, widely adopted format that supports complex data structures, Unicode characters, and efficient web-based data transfer. Its goal is to enable faster processing, easier validation, and improved traceability while maintaining compatibility with CDISC models such as SDTM and ADaM.

Despite these advancements, the FDA continues to require the SAS v5 XPT format for regulatory submissions. This decision ensures backward compatibility, system stability, and interoperability across legacy tools and processes used in the review environment. However, the agency has conducted pilot projects—such as those exploring Dataset-JSON—to assess the feasibility of adopting more modern standards in the future. These pilots signal a gradual transition toward more efficient and flexible data exchange mechanisms while preserving the reliability that SAS v5 XPT has provided for decades.

With the rise of advanced, cloud-based, GxP-compliant Statistical Computing Environments (SCEs) and the growing adoption of polyglot programming (SCE that enable users to program in SAS, R, and Python) approaches across the industry, we are witnessing a transformative shift in how we analyze data and prepare regulatory submissions. This evolution is redefining both our tools and our mindset. As we look beyond the long-standing SAS XPT format toward modern, flexible standards like Dataset-JSON, the question arises:

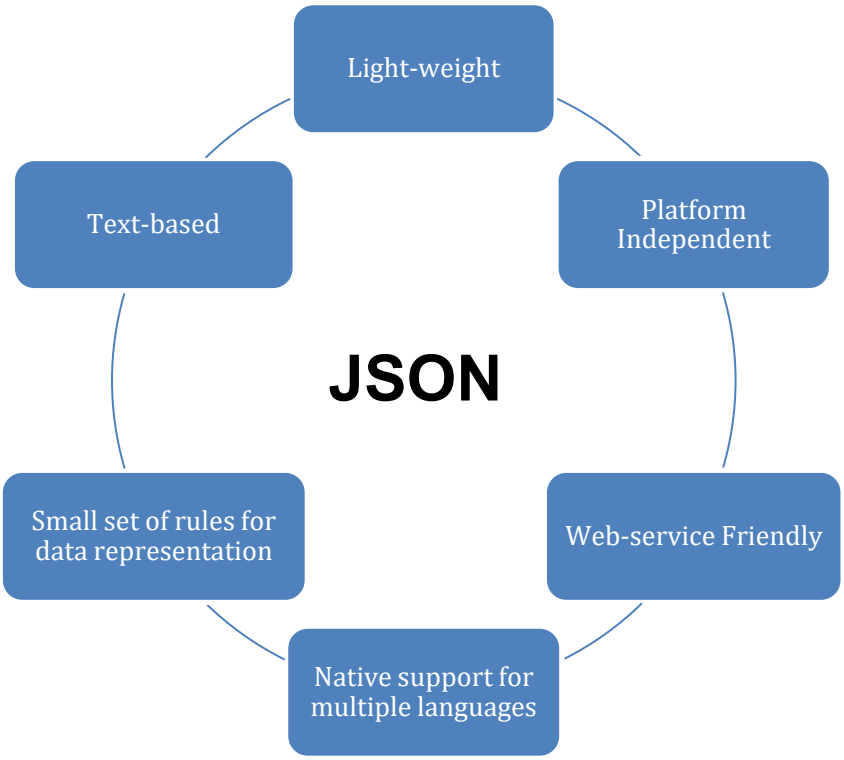
- What kind of change can we expect?

The move to JSON-based submissions could bring a more efficient, transparent, and interoperable workflow; one that aligns seamlessly with today's cloud-native, multi-language analytical ecosystems. JSON's structured, human-readable design and support for modern data types open opportunities for automation, real-time validation, and integration with a wide range of analytical and visualization tools.

The transition will not happen overnight, but the momentum is building. The combination of regulatory pilots, maturing infrastructure, and industry readiness suggests that the shift from XPTs to JSONs may be closer than we think; marking a pivotal moment in the digital transformation of clinical trial submissions.

WHAT IS JSON?

JavaScript Object Notation (JSON) is a lightweight, text-based data interchange format widely used for representing structured information in a way that is both human-readable and machine-readable. Based on a simple key value structure, JSON enables flexible and efficient data exchange across systems, programming languages, and platforms. Unlike older, fixed width or binary formats such as SAS XPT, JSON supports hierarchical and nested data, making it ideal for representing complex relationships and metadata. Because of its simplicity, extensibility, and compatibility with modern web and cloud-based technologies, JSON has become a preferred format for many data-driven applications, including clinical research, where it is increasingly being explored as a potential alternative transport format for regulatory data submissions.



WHAT IS DATASET-JSON?

In clinical research and regulatory space, CDISC's Dataset-JSON standard adapts the JSON format for use in storing and transporting clinical study datasets. It was developed as a modernized evolution of the earlier Dataset-XML standard, designed to address the limitations observed in XML; such as verbosity, larger file sizes, and less efficient parsing. While Dataset-XML successfully provided a non-SAS alternative to the long-standing SAS Version 5 Transport (XPT) format, the industry's growing adoption of cloud-based analytics, APIs, and multi-language data science environments highlighted the need for a more lightweight, flexible, and scalable format leading to the creation of Dataset-JSON.

Dataset-JSON offers several advantages over traditional XPT files, including enhanced metadata representation, improved handling of variable attributes and missing values, and seamless interoperability with modern Statistical Computing Environments (SCEs). Although Dataset-JSON is currently under FDA evaluation, it represents an important milestone in the ongoing evolution of CDISC standards—advancing the industry toward more open, interoperable, and technology-agnostic data exchange that supports automation, traceability, and reproducibility in regulatory submissions.

WHAT DOES DATASET-JSON LOOK LIKE?

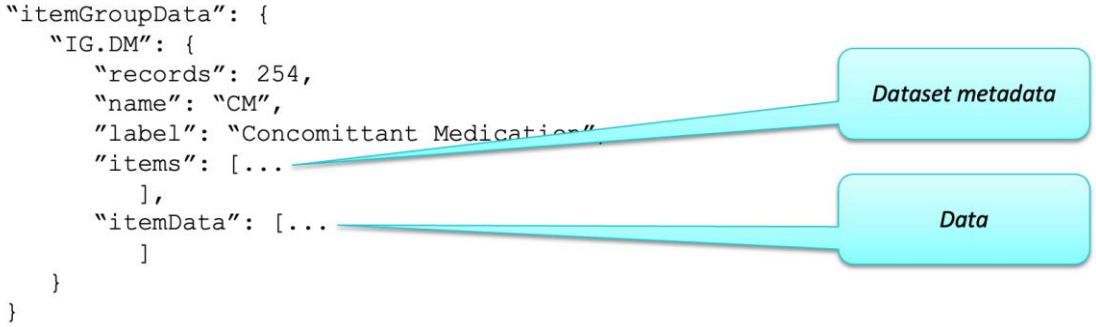
The CDISC Dataset-JSON standard defines a structured, machine-readable format for representing clinical trial datasets using JSON. Designed to serve as a modern transport format for regulatory submissions, Dataset-JSON maintains the same conceptual model as traditional SAS XPT datasets—tabular data with variables, records, and metadata—while adopting a more extensible and efficient syntax suited to modern computing environments.

Each Dataset-JSON file represents a single dataset and is organized into three main components:

The JSON Three-Component Structure

Designed as a modern transport format, Dataset-JSON maintains the tabular conceptual model of traditional datasets but adopts an extensible JSON syntax. Each file is organized into three main components:

1. **Metadata Section (itemGroupData):** Provides high-level context, including the dataset name, label, domain, and creation details. It supports study identifiers, version information, and pointers to controlled terminologies or codelists.



2. **Variables Section (items):** Defines the dataset structure, listing names, labels, data types, lengths, and formats. Crucially, it supports additional metadata like controlled terminology references, origins, and derivation details, significantly improving traceability and alignment with CDISC Define-XML.

```
"items": [
  {"name": "USUBJID", "label": "Unique Subject Identifier", "type":
"character"},
  {"name": "AGE", "label": "Age", "type": "numeric"},
  {"name": "SEX", "label": "Sex", "type": "character"}
]
```

3. **Data Section (itemData):** Contains the actual observations. Missing values are explicitly represented as null, avoiding the ambiguity inherent in XPT's missing value representations.

```
"itemData": [
  {"USUBJID": "1001-1001", "AGE": 36, "SEX": "M"},
  {"USUBJID": "1001-1012", "AGE": 65, "SEX": "F"}
]
```

The Dataset-JSON structure aligns closely with the metadata captured in Define-XML, ensuring interoperability across existing CDISC standards.

Regulatory and Pilot Verification

The U.S. FDA has actively evaluated the Dataset-JSON v1.0 standard through pilot studies, concluding that it is viable as a primary transport option. The evaluation noted only minor issues related to conversion tools and interoperability, which were resolved by Dataset-JSON v1.1. This official assessment reaffirms its readiness as a modern standard, although the FDA noted that updates to internal systems are necessary before full adoption.

JSON vs XML (What is the Difference?)

JSON and XML share several similarities: both are self-describing, meaning the structure and meaning of the data can be understood from the content itself, and both support hierarchical or nested structures that allow complex data relationships to be represented in a readable format. Because they are plain text, they are platform-independent and easy to transmit across networks or between applications.

Despite their similarities, JSON and XML differ in structure, syntax, and performance. JSON uses braces {} and brackets [] to define objects and arrays, while XML relies on opening and closing tags such as <tag></tag>. JSON omits end tags, resulting in a more concise representation. Another key distinction is that JSON natively supports arrays and basic data types—such as strings, numbers, booleans, and null values—whereas XML treats all data as text unless a schema (like XSD) is used. Parsing also differs significantly: JSON can be directly parsed and converted into native objects in JavaScript and many other languages using built-in functions such as JSON.parse(), whereas XML typically requires a separate parser or DOM traversal to extract data.

JSON is generally considered superior for most applications because it is faster, lighter, and easier to work with than XML. Its syntax is less verbose, resulting in smaller file sizes and more efficient data transfer. Parsing JSON is quicker and less resource-intensive due to its simplicity and tight integration with modern programming languages. Additionally, JSON's readability and compatibility with JavaScript make it an excellent choice for APIs, web applications, and RESTful services. XML still has its place particularly where document markup, namespaces, or schema validation are important but for web and data exchange use cases, JSON offers clear advantages in performance, simplicity, and developer usability.

	Description
JSON is similar to XML…	• Human readable (aka "self-describing“) • Have a hierarchical structure (values within values)
How JSON Differs from XML…	• Doesn't use end tags • You can use arrays • Standard JavaScript functions (and other languages too) can parse it
Why JSON is Better?	• JSON is faster and easier than XML to parse • JSON files file sizes are smaller than XML

Example JSON and Corresponding XML

Total Number of Characters 336 vs 420

JSON (array of records)

```
[
  {
   "STUDYID": "ABC123",
   "DOMAIN": "AE",
   "USUBJID": "ABC123-001",
   "AESEQ": 1,
   "AETERM": "Headache",
   "AESTDTC": "2025-10-28",
   "AEENDTC": "2025-10-29",
   "AESEV": "MILD",
   "AESER": "N",
   "AEREL": "POSSIBLE",
   "AEACN": "DOSE NOT CHANGED",
   "AEOUT": "RECOVERED/RESOLVED",
   "AETOXGR": "1"
  }
]
```

XML (simple SDTM-style)

```
<AE_RECORDS>
  <AE>
    <STUDYID>ABC123</STUDYID>
    <DOMAIN>AE</DOMAIN>
    <USUBJID>ABC123-001</USUBJID>
    <AESEQ>1</AESEQ>
    <AETERM>Headache</AETERM>
    <AESTDTC>2025-10-28</AESTDTC>
    <AEENDTC>2025-10-29</AEENDTC>
    <AESEV>MILD</AESEV>
    <AESER>N</AESER>
    <AEREL>POSSIBLE</AEREL>
    <AEACN>DOSE NOT CHANGED</AEACN>
    <AEOUT>RECOVERED/RESOLVED</AEOUT>
    <AETOXGR>1</AETOXGR>
  </AE>
</AE_RECORDS>
```

THE SAS PROGRAMMER'S TECHNICAL MANDATE: MASTERING THE BRIDGE

For the long-time SAS programmer, the shift is a profound I/O (Input/Output) imperative focused on interoperability, not migration. The goal is to build a reliable, audited bridge between the familiar SAS environment and the new standard.

Essential SAS Toolset Mastery

SAS programmers must master the procedures designed for JSON I/O, available in modern SCEs (like Sycamore CDR-SCE):

SAS Tool/Procedure	Primary Function	Programmer's Mandate
PROC JSON	**Writing (Export)**	The primary output tool. Must be used with options (like KEYS or WRITE OPEN/CLOSE) to enforce the complex, nested structure required by the CDISC Dataset-JSON schema.
JSON LIBNAME Engine	**Reading (Import)**	Used for round-trip testing and reading JSON source data, allowing the JSON file to be treated as a standard SAS dataset for comparison and validation.
PROC GROOVY or PROC LUA	**Advanced Handling**	Used for custom, highly efficient parsing or generation for very large datasets where complex mapping rules exceed standard PROC JSON performance.

The programmer's output logic must move beyond simple PUT statements, which are "tedious and error-prone for high-volume data", to standardized procedures to build modular submission macros capable of outputting both XPT and JSON.

GxP Validation and Risk Mitigation in the JSON Era

The flexibility of JSON heightens the GxP validation challenge, demanding schema-aware, metadata-driven validation.

A GxP Framework for JSON Validation: The Three-Pillar Protocol

To maintain GxP compliance (per 21 CFR Part 11) and guarantee electronic record integrity, the submission pipeline must enforce a Three-Pillar Validation Protocol:

1. **Structural Validation (Schema Check):** Ensures the JSON file adheres to the physical rules defined in the CDISC Dataset-JSON v1.1 Implementation Guide. This validates the correct placement of all containers, brackets, and key-value pairs.
2. **Metadata Conformance (Define-XML Cross-Check):** A semantic check ensuring the metadata embedded within the JSON perfectly aligns with the standalone Define-XML file. The programmer must ensure attributes, origins, and derivation details in the JSON "Variables Section" are perfectly consistent, ensuring complete traceability of the output.
3. **Round-Trip Integrity (Lossless Data Verification):** The ultimate safeguard: reading the generated JSON back into the native environment and performing a bit-for-bit comparison against the original validated source dataset.

This demands automated validation tools that check against both the Dataset-JSON Schema and the Define-XML file before final submission.

CRITICAL INTEROPERABILITY CHALLENGES FOR LOSSLESS DATA for SAS Programmers

Pilot studies identified specific technical obstacles where the fundamental differences between the SAS language and the JSON standard risk data integrity. See the Appendix and references for more information.

1. Date/Time Epoch and Representation: The Temporal Collision

This challenge arises from the diverse ways different systems measure time.

- **The Conflict:** The SAS date epoch is January 1, 1960. If not handled explicitly, an integer-based SAS date exported directly could be misinterpreted by the regulatory reviewer's system, resulting in grossly incorrect patient timelines.
- **Mitigation Logic:** Programmers must ensure dates and datetimes are converted to the unambiguous ISO 8601 string format in the JSON output, preserving data integrity upon ingestion.
    - SAS Example: Use PUT functions with explicit formats (e.g., IS8601DA, IS8601DT) to enforce the correct string structure before export via PROC JSON.

2. Floating-Point Precision and Rounding: The Numeric Integrity Crisis

This affects all numeric data requiring high decimal accuracy.

- **The Conflict:** Various JSON libraries and analytical tools apply differing floating-point and rounding strategies, creating an "interoperability issue".
- **The Risk:** Minor rounding differences constitute a loss of data integrity and validation failure upon round-trip testing.
- **Mitigation Logic:** For high-precision variables, the best practice is to represent the value as a JSON string (text) to prevent the receiving system's JSON library from introducing unintended rounding.

3. Explicit Missing Value Handling: The Ambiguity Gap
This challenge centers on reconciling SAS's rich system of missing codes with JSON's limited vocabulary.

- **The Conflict:** SAS uses system-missing and 27 special missing values. Dataset-JSON mandates missing values be represented explicitly as the keyword null.
- **The Risk:** Failure to map all SAS missing types consistently to null during export, or allowing an empty string for a numeric missing value, violates the Dataset-JSON specification.
- **Mitigation Logic:** The programmer must verify that their procedures correctly translate every missing type to the explicit JSON keyword, thereby "avoiding ambiguity and improving clarity".

4. Unicode and Character Encoding: The Global Data Barrier
This challenge addresses the representation of global clinical trial data.

- **The Conflict:** The legacy SAS XPT format is largely ASCII-based, while JSON supports full Unicode (UTF-8).
- **The Risk:** If the SAS environment is not explicitly set to use UTF-8 encoding, non-ASCII characters (e.g., foreign language names) can be corrupted, resulting in encoding errors.
- **Mitigation Logic:** The SAS environment must be configured for UTF-8 encoding. Programmers need procedures to confirm that non-ASCII characters are losslessly encoded in the JSON output.

The "Gotchas": Top 5 Technical Watch-Outs for SAS Programmers
These are the most critical technical pitfalls where the flexible JSON format clashes directly with GxP strictness, grouped by the programmer primarily affected.

1. The Date Epoch Mismatch

Conflict: SAS uses the 1/1/1960 epoch. Exporting this integer risks misinterpretation by Unix/web systems, leading to grossly incorrect patient timelines.
Example of Failure: A SAS date variable representing a patient's death date of 2020-01-01 is exported as the integer 21915. If the receiving system reads this as a Unix date, it will incorrectly interpret the event as having occurred in 1980.
Watch Out: Always use explicit conversion logic to output dates as ISO 8601 strings.

2. The Floating-Point Precision Trap

Conflict: Different JSON libraries introduce varying floating-point and rounding strategies.
Risk: Minor rounding differences constitute a loss of data integrity.
Watch Out: For critical precision variables, output the numeric value as a JSON string (text) to prevent unintended rounding by the receiving library.

3. The Container Control and Metadata Loss Gotcha

Conflict: The CDISC schema requires precise nesting (itemGroupData $\rightarrow$ items $\rightarrow$ itemData). Default PROC JSON often produces a flat array that doesn't match the required nesting.
Risk: The file will not validate against the Dataset-JSON schema, resulting in immediate rejection, and critical metadata may be lost.
Watch Out: Utilize advanced features like WRITE OPEN and WRITE CLOSE within PROC JSON or rely on pre-validated community macros to enforce the precise nested structure.

4. The Unicode Encoding Barrier

Conflict: The legacy SAS XPT format is largely ASCII-based. Non-ASCII characters are corrupted if the SAS environment is not configured for UTF-8.
Example of Failure: A patient's name "François" is corrupted to "Franois" during JSON export. This is a data integrity error.
Watch Out: Verify the SAS session and output encoding settings and implement tools to flag any characters that do not match the intended UTF-8 encoding scheme.

5. The Implicit vs. Explicit Missing Value Ambiguity

Conflict: SAS's various missing values vs. JSON's single requirement: explicit null. Risk: Failure to map all SAS missing types to null violates the Dataset-JSON specification.
Watch Out: Rigorously verify that every SAS missing value translates only to null in the JSON output.

These are the most critical technical pitfalls where JSON's flexibility clashes with SAS's rigid heritage, leading to risk of GxP non-compliance.

SPECIALIZED DEEP DIVE: THE OPEN-SOURCE PROGRAMMER

For R and Python programmers, the challenge shifts from I/O mechanics (which are native) to schema enforcement and GxP compliance within flexible open-source toolchains. See the Appendix and references for more information. The R/Python programmer must integrate the following:

The Open-Source Mandate: Schema Guardianship

1. **I/O & Structural Control:** Use packages like jsonlite (R) and json (Python) to explicitly control nesting, data typing, and null output.

2. **Validation Libraries:** Implement automated checks against the official CDISC Dataset-JSON schema (e.g., using jsonschema in Python).

3. **Metadata Injection: Utilize** packages designed to inject Dataset-JSON-required metadata (origins, Define-XML linkages) into the final JSON object.

The change management priority for R/Python developers is process standardization and schema alignment, not technical learning.

Open-Source "Gotchas"

1. **Dynamic Typing/Schema Enforcement Trap:** Flexible R/Python types can unintentionally switch (e.g., numeric to character), leading to submission rejection due to fixed CDISC types.
   Watch Out: Always implement pre-serialization validation using libraries to check the object against the formal CDISC Dataset-JSON schema.

2. **Native Date/Time Format Clash:** Default R/Python serialization yields Unix timestamps (integer) or language-specific string formats. Risk: Non-compliant date strings or ambiguous integers.
   Watch Out: Explicitly format date/time objects to output the ISO 8601 string format, guaranteeing unambiguous time stamping.

3. **Handling of Empty Values vs. null**: Missing values (NA, None) in open-source languages must be rigorously converted to the literal JSON keyword null.
   Watch Out: Rigorously test serialization settings to confirm that all forms of missing/empty values are converted to null, as required by the specification.

4. **Performance Bottleneck for Large Datasets**: Generic JSON serialization for massive datasets (e.g., large lab domains) can be a significant memory bottleneck.
   Watch Out: Use optimized libraries like Python's pandas or R's data.table. For extremely large files, explore NDJSON (Newline Delimited JSON) or streaming techniques.

5. **Flexible Data Structure (Container Control) Problem**: R/Python's easy custom object creation risks wrapping data in a non-standard object or array, leading to immediate validation failure because the structure does not match the CDISC specification.
   Watch Out: Write code that explicitly builds the required three-part hierarchy (Metadata, Variables, Data sections).

CHANGE MANAGEMENT: UNIFYING POLYGLOT TEAMS

The transition demands a change management strategy that empowers both the SAS compliance expert and the open-source optimizer.

The SAS Programmer's Procedural Shift

The procedural shift for the SAS programmer is significant:

- **From Implicit to Explicit:** Moving from relying on the fixed structure of a SAS dataset to explicitly building the JSON structure using complex I/O syntax.
- **Skill Reinvestment:** The focus shifts from manual I/O checks toward high-value automation and validation logic.

By mastering the SAS-to-JSON bridge, the SAS team solidifies their role as the guardians of validated compliance data.

The Open-Source Programmer's Focus

The R and Python programmer's focus shifts to process standardization and schema fidelity. Their challenge is ensuring the flexible code adheres to the strict CDISC schema through:

- **Schema Alignment:** Enforcing adherence to CDISC's specific nesting requirements.
- **Validation Integration:** Integrating external schema validation tools as a mandatory, auditable step.

THE DIGITAL IMPERATIVE: ACTIONABLE SUMMARY AND LONG-TERM VALUE

The evolution to Dataset-JSON is a necessary shift toward a flexible, interoperable, and technology-driven future. The success of this transition rests on procedural rigor and strategic investment, not merely file conversion.

The ultimate demonstration of compliance and the final step in validation is certifying that the generated file is lossless.

1. **Generation:** Generate the final, submission-ready Dataset-JSON file (SUBMISSION.JSON).
2. **Read-Back (The Trip):** Immediately read the SUBMISSION.JSON back into the same Statistical Computing Environment.
    - **SAS Action:** Use the JSON LIBNAME Engine to create a temporary SAS dataset (TEMP.SAS7BDAT).
3. **Comparison (The Protocol):** Execute a programmatic comparison between the original validated source and the read-back copy.
    - **SAS Action:** Use PROC COMPARE explicitly checking for differences in value, length, and format. Any reported difference signals a failure in the conversion process.

Only after a successful, zero-difference result from the Round-Trip Protocol can the programmer confidently certify the file for regulatory submission.

LEADERSHIP MANDATE AND COMPETITIVE ADVANTAGE

For the industry embracing Dataset-JSON translates directly into long-term competitive relevance:

- **Mitigation of Technical Debt:** The transition permanently eliminates the costly manual workarounds and technical limitations imposed by XPT (proxy variables, truncated metadata).

- **Empowering Polyglot Teams:** By formalizing the JSON standard, we empower the SAS teams to remain the guardians of validated compliance data while enabling seamless, auditable interoperability with R and Python.

- **Enabling Automation:** The structured, machine-readable nature of Dataset-JSON is the key to unlocking real-time validation and automated submission processes, accelerating the clinical data lifecycle.

This proactive technical alignment is the foundation for a more connected, efficient, and data-driven future in drug development.

CONCLUSION: FINALIZING THE DIGITAL FOUNDATION

The evolution to Dataset-JSON is a necessary shift toward a flexible, interoperable, and technology-driven future. By proactively addressing the challenges of precision, encoding, epoch conversion, and structural validation through enhanced SAS procedures and rigorous open-source validation, organizations achieve the efficiencies of a true machine-readable, end-to-end digital integration. The time for industry stakeholders to weigh in, prepare internally, and contribute to shaping the standards is now.

APPENDIX: ADDITIONAL INFORMATION/CODE EXAMPLES

MORE INFORMATION FOR SAS CODING

Highly recommend the reader to refer to the extensive references for more information as well as Lex Jansen's git repo (also listed within the references, placed here for convenience: https://github.com/lexjansen/dataset-json-sas/blob/master/macros/write_datasetjson.sas). The repository provides SAS implementation for converting SAS Datasets to JSON in compliance with the CDISC Dataset-JSON v1.1 specification

SAS HELP: PROC JSON

The SAS Institute has simplified the creation process for JSON files via the PROC JSON. See the reference section for direct link to SAS help (at the time of this paper it was last updated on October 14, 2025). The PROC JSON can export a SAS dataset or subset by adding a WHERE data set option. Highly recommend using the PROC JSON PRETTY option as it applies formatting to the resulting JSON file (e.g., indention).

An Example (from SAS Help):

```
 proc json out="path-to-DefaultOutput.json" pretty;
   export sashelp.class (where=(age=11));
run;
```

SAS CODE – MAXIMUM CONTROL

Below is another example which illustrates leveraging PROC JSON but produces a more complex resulting JSON file; this example has been included to show that programmers can produce highly complex output using PROC JSON; there are many ways to achieve this output, it is only one example. Again, the purpose of this paper is to make you aware of the tools and procedures that are available.

```
/* Step 1: Define input dataset */
%let indata = sdtm.ae;
%let outjson = "C:\temp\class_dataset.json";

/* Step 2: Get dataset metadata */
proc contents data=&indata out=meta (keep=name type length label varnum) noprint;
run;

/* Step 3: Write Dataset-JSON structure using PROC JSON */
filename outjson &outjson;
proc json out=outjson pretty nosastags;
   write open object;
   /* Dataset-level metadata */
   write values "datasetMetadata";
   write open object;
   write values "name" "&indata";
   write values "recordCount" (select count(*) from &indata);
   write close;

   /* Variable metadata */
   write values "variables";
   write open array;
   export meta / nosastags;
   write close;

  /* Data records */
  write values "data";
  write open array;
  export &indata / nosastags;
  write close;
  write close;
run;
```

JSON Output:

```
{
  "clinicalData": {
    "studyOID": "STUDY123",
    "metaDataVersionOID": "MDV.STUDY123.SDTM.1.0",
    "itemGroupData": [
      {
```

```
      "itemGroupOID": "IG.AE",
      "records": [
       {
        "AESEQ": "1",
        "AETERM": "Headache",
        "AESEV": "MILD",
        "AESER": "N"
       },
       {
        "AESEQ": "2",
        "AETERM": "Nausea",
        "AESEV": "MODERATE",
        "AESER": "N"
       }
      ]
     }
    ]
  },
  "metaDataVersion": {
    "studyOID": "STUDY123",
    "itemDefs": [
      {"OID": "IT.AE.AESEQ", "name": "AESEQ", "dataType": "integer"},
      {"OID": "IT.AE.AETERM", "name": "AETERM", "dataType": "string"},
      {"OID": "IT.AE.AESEV", "name": "AESEV", "dataType": "string"},
      {"OID": "IT.AE.AESER", "name": "AESER", "dataType": "string"}
    ]
  }
}
```

SAMPLE SAS & R EQUIVALENCY

Below is a simple R Language example which requires three different R Packages (haven, jsonlite, and dplyr).

SAS Functionality	R Equivalent	Description
PROC JSON... EXPORT dataset (to export a SAS dataset)	**jsonlite::write_json(x, path)**	Exports an R object (x), typically a **data frame** or a **list**, directly to a JSON file specified by path.
PROC JSON... PRETTY (for formatting / indentation)	**pretty = TRUE** argument in write_json()	Applies readable formatting, including indentation and line breaks, to the resulting JSON file, just like the PRETTY option in SAS.
WHERE data set option (to subset data)	**Standard R data manipulation** (e.g., dplyr::filter(), base R subsetting [])	You subset your R data frame *before* passing it to write_json(). For example: data_subset <- data_full[data_full$Age > 30,] followed by write_json(data_subset, "output.json").

SAMPLE R CODE

Below is a simple R Language example which requires three different R Packages (haven, jsonlite, and dplyr).

```
# 1. Load the required package
library(jsonlite)

# 2. Create a sample R data frame (similar to a SAS dataset)
my_data <- data.frame(
  ID = c(1, 2, 3),
  Name = c("Alice", "Bob", "Charlie"),
  Value = c(100.5, 200.7, 150.2)
)

# 3. Export the data frame to a JSON file
#    - 'my_data' is the object to export
#    - 'output.json' is the file path
#    - 'pretty = TRUE' is similar to the PROC JSON PRETTY option
write_json(my_data, "output.json", pretty = TRUE)
```

R CODE – MAXIMUM CONTROL

Similar to SAS, the programmer can customize the resulting JSON file in many different ways.

```
# ------------------------------------------------------------
# Step 1: Load required packages
# ------------------------------------------------------------
library(haven)   # for reading SAS datasets
library(jsonlite) # for writing JSON
library(dplyr)

# ------------------------------------------------------------
# Step 2: Define file paths
# ------------------------------------------------------------
infile  <- "C:/clinical/data/ae.sas7bdat"     # Input SAS file
outfile <- "C:/clinical/output/ae.json"       # Output JSON file
studyid <- "STUDY123"

# ------------------------------------------------------------
# Step 3: Read SAS dataset
# ------------------------------------------------------------
ae <- read_sas(infile)

# ------------------------------------------------------------
# Step 4: Build metadata (variables)
# ------------------------------------------------------------
meta <- data.frame(
  OID = paste0("IT.AE.", names(ae)),
  name = names(ae),
  dataType = sapply(ae, function(x) {
    if (is.numeric(x)) "float" else "string"
  }),
  stringsAsFactors = FALSE
)

# ------------------------------------------------------------
# Step 5: Build itemGroupData (actual records)
# ------------------------------------------------------------
records <- apply(ae, 1, as.list)
itemGroup <- list(
  itemGroupOID = "IG.AE",
  records = records
)

# ------------------------------------------------------------
# Step 6: Assemble Dataset-JSON structure
# ------------------------------------------------------------
dataset_json <- list(
  clinicalData = list(
    studyOID = studyid,
    metaDataVersionOID = paste0("MDV.", studyid, ".SDTM.1.0"),
    itemGroupData = list(itemGroup)
  ),
  metaDataVersion = list(
    studyOID = studyid,
    itemDefs = meta
  )
)

# ------------------------------------------------------------
# Step 7: Write JSON to file
# ------------------------------------------------------------
write_json(dataset_json, path = outfile, pretty = TRUE, auto_unbox = TRUE)

cat("Dataset-JSON file created successfully:\n", outfile, "\n")
```

REFERENCES

1. **ANDERSON, JESSE AND HUME, SAM.** "DATASET-JSON AS ALTERNATIVE TRANSPORT FORMAT FOR REGULATORY SUBMISSIONS." YOUTUBE VIDEO. OCTOBER 26, 2023. HTTPS://YOUTU.BE/LJR6D4AW7NA?SI=IK0EGWCKPKKWNYKF.
2. **ANDERSON, JESSE AND HUME, SAM.** "DATASET-JSON PILOT REPORT AND NEXT STEPS." PRESENTATION, CDISC JAPAN INTERCHANGE. JUNE 2024. HTTPS://WWW.CDISC.ORG/SITES/DEFAULT/FILES/2024-06/2024-JP-INTERCHANGE-SESSION5-DATASET-JSON.PDF.
3. **ANDERSON, JESSE AND HUME, SAM.** CDISC "DATASET-JSON AS ALTERNATIVE TRANSPORT FORMAT FOR REGULATORY SUBMISSIONS." PRESENTATION, CDISC PLENARY. OCTOBER 2023. HTTPS://WWW.CDISC.ORG/SITES/DEFAULT/FILES/2023-10/2023-CDISC-DATASET-JSON-PLENARY-V5_0.PDF
4. **CLINICAL DATA INTERCHANGE STANDARDS CONSORTIUM (CDISC).** "DATASET-JSON." ACCESSED OCTOBER 28, 2025. HTTPS://WWW.CDISC.ORG/STANDARDS/DATA-EXCHANGE/DATASET-JSON.
5. **CLINICAL DATA INTERCHANGE STANDARDS CONSORTIUM (CDISC).** DEFINE-XML SPECIFICATION. AUSTIN, TX: CDISC. ACCESSED OCTOBER 23, 2025. HTTPS://WWW.CDISC.ORG/STANDARDS/DATA-EXCHANGE/DEFINE-XML.
6. **FOOD AND DRUG ADMINISTRATION (FDA).** "ELECTRONIC STUDY DATA SUBMISSION; DATA STANDARDS; CLINICAL DATA INTERCHANGE STANDARDS CONSORTIUM DATASET-JAVASCRIPT OBJECT NOTATION; REQUEST FOR COMMENTS." FEDERAL REGISTER 90, NO. 67 (APRIL 9, 2025). HTTPS://WWW.REGULATIONS.GOV/DOCUMENT/FDA-2025-N-0129-0001.
7. **FOOD AND DRUG ADMINISTRATION (FDA).** "PROVIDING REGULATORY SUBMISSIONS IN ELECTRONIC FORMAT — STANDARDIZED STUDY DATA: GUIDANCE FOR INDUSTRY. SILVER SPRING, MD: U.S. FOOD AND DRUG ADMINISTRATION, JUNE 2021. ACCESSED OCTOBER 23, 2025. HTTPS://WWW.FDA.GOV/REGULATORY-INFORMATION/SEARCH-FDA-GUIDANCE-DOCUMENTS/PROVIDING-REGULATORY-SUBMISSIONS-ELECTRONIC-FORMAT-STANDARDIZED-STUDY-DATA.
8. FOOD AND DRUG ADMINISTRATION (FDA). STUDY DATA TECHNICAL CONFORMANCE GUIDE. SILVER SPRING, MD: U.S. FOOD AND DRUG ADMINISTRATION. ACCESSED OCTOBER 23, 2025. HTTPS://WWW.FDA.GOV/REGULATORY-INFORMATION/SEARCH-FDA-GUIDANCE-DOCUMENTS/STUDY-DATA-TECHNICAL-CONFORMANCE-GUIDE-TECHNICAL-SPECIFICATIONS-DOCUMENT.
9. FOOD AND DRUG ADMINISTRATION (FDA). "DATASET-JSON PILOT REPORT AND NEXT STEPS." YOUTUBE VIDEO, 15:47. JULY 25, 2024. HTTPS://WWW.YOUTUBE.COM/WATCH?V=LJR6D4AW7NA.
10. FOOD AND DRUG ADMINISTRATION (FDA). "DATASET-JSON PILOT REPORT AND NEXT STEPS." NEWS & EVENTS FOR HUMAN DRUGS. JULY 25, 2024. HTTPS://WWW.FDA.GOV/DRUGS/NEWS-EVENTS-HUMAN-DRUGS/DATASET-JSON-PILOT-REPORT-AND-NEXT-STEPS-07252024.
11. FOOD AND DRUG ADMINISTRATION (FDA). "STUDY DATA STANDARDS RESOURCES." LAST REVIEWED JULY 25, 2024. HTTPS://WWW.FDA.GOV/INDUSTRY/FDA-DATA-STANDARDS-ADVISORY-BOARD/STUDY-DATA-STANDARDS-RESOURCES.
12. HUME, SAM. "NO MORE XPT: PILOTING NEW DATASET-JSON FOR FDA SUBMISSIONS." CLINICAL LEADER, JULY 12, 2023. HTTPS://WWW.CLINICALLEADER.COM/DOC/NO-MORE-XPT-PILOTING-NEW-DATASET-JSON-FOR-FDA-SUBMISSIONS-0001.
13. JANSEN, LEX. 'DATASET-JSON MACROS IN GIT REPO' ACCESSED OCTOBER 29, 2025. HTTPS://GITHUB.COM/LEXJANSEN/DATASET-JSON-SAS/BLOB/MASTER/MACROS/WRITE_DATASETJSON.SAS.
14. JANSEN, LEX. "WORKING WITH DATASET-JSON USING SAS." PAPER PRESENTED AT PHARMASUG 2022. HTTPS://WWW.LEXJANSEN.COM/PHARMASUG/2022/AD/PHARMASUG-2022-AD-150_PPT.PDF.
15. JANSEN, LEX. "DATASET-JSON: SAS© IMPLEMENTATION" ACCESSED OCTOBER 29, 2025. HTTPS://WWW.CDISC.ORG/SITES/DEFAULT/FILES/PDF/CDISC_COSA_WEBINAR_20231005_DATASET-JSON_SAS.PDF.
16. LINKER, ADAM. "CREATING AND CONTROLLING JSON OUTPUT WITH THE JSON PROCEDURE." PAPER SAS3506-2019. ACCESSED OCTOBER 28, 2025. HTTPS://SUPPORT.SAS.COM/RESOURCES/PAPERS/PROCEEDINGS19/3506-2019.PDF.
17. PHARMACEUTICAL USERS SOFTWARE EXCHANGE (PHUSE). "DATASET-JSON AS ALTERNATIVE TRANSPORT FORMAT FOR REGULATORY SUBMISSIONS." PHUSE ADVANCE HUB. LAST MODIFIED JUNE 10, 2025. HTTPS://ADVANCE.HUB.PHUSE.GLOBAL/WIKI/SPACES/WEL/PAGES/26806026/DATASET-JSON+AS+ALTERNATIVE+TRANSPORT+FORMAT+FOR+REGULATORY+SUBMISSIONS.
18. SALTWARE SOLUTIONS. "UNPACKING THE COMPLEXITIES OF CLINICAL DATA SUBMISSION STANDARDS." LAST MODIFIED AUGUST 2024. HTTPS://WWW.SALTWARE.SOLUTIONS/UNPACKING-CLINICAL-SUBMISSION-STANDARDS/.

19. SAS GLOBAL FORUM. "CREATING AND CONTROLLING JSON OUTPUT WITH PROC JSON." SAS GLOBAL FORUM PROCEEDINGS. (YEAR AND AUTHORS VARY, FOCUSING ON PROC JSON CONTROL). ACCESSED OCTOBER 23, 2025. HTTPS://SUPPORT.SAS.COM/.
20. SAS. "JSON PROCEDURE." ACCESSED OCTOBER 23, 2025. HTTPS://DOCUMENTATION.SAS.COM/DOC/EN/PGMSASCDC/V_068/PROC/N0NEJFK9Q0PZMNN181L92QK3AH2E.HTM.

CONTACT INFORMATION

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

Author Name: Bill Qubeck
Company: Sycamore Informatics
Address: Waltham, MA, USA
Email: bill.qubeck@sycamoreinformatics.com
Website: www.sycamoreinformatics.com