Paper SI06

From XPT to Dataset-JSON: Enabling Real-Time Validation and Streamlined Submissions

Hugo Signol, Cytel, Geneva, Switzerland
Sebastià Barceló, Cytel, Geneva, Switzerland

ABSTRACT

Context

Dataset-JSON overcomes XPT limitations. It eliminates the long-standing restrictions of SAS Transport (XPT) format, its compact footprint makes it well-suited for API-based workflows, reducing data management complexity. By leveraging APIs, Dataset-JSON removes the need for file-based data handling, enabling direct use of databases as a single source of truth. Furthermore, APIs allows efficient data filtering before loading, reducing unnecessary data transfer and improving overall efficiency. Moreover, this format combined with APIs accelerates the regulatory submission process by fostering early collaboration between sponsors and regulatory authorities. Data can be shared as soon as it is collected and transformed into standardized formats such as SDTM, SEND, or ADaM. This early exchange of information enables regulators to start their review process sooner, leading to faster approvals. As a result, the "time to market" for new treatments can be significantly reduced, benefiting both the industry and patients. Finally, developing software tools for Dataset-JSON is considerably easier than for XPT, making it a more accessible format for industry.

Rules & Dataset-JSON-Viewer

Our team participated in the CDISC Dataset-JSON-Viewer Hackathon during the end of 2024 Q4. The goal was to create a tool able to view Dataset-JSON extracts along with critical functionality (sort, filter, search, etc.) as the SAS viewer and going further if possible. At the time of the final submission to Hackathon, we had a basic dynamic viewer, with some extra functionalities like live descriptives statistics. To bring extra value to this viewer, we are implementing rules to be manually (via a button) or automatically checked. These rules can be CDISC conformance or CORE rules, as well as all standard checks performed at regular intervals. Soon, when we could imagine, for example some API calls for P21 or CORE rules, this will allow people from the industry to gain a considerable amount of time by checking their data even before submitting it.

INTRODUCTION

For decades, XPT format dictated how data moved and how validation occurred. When submitting study data to authorities, organizations use the SAS® XPORT (XPT) format, which dates to 1989. While submission processes have advanced and are still advancing, XPT remains mandatory despite being outdated. Indeed, it has many restrictions, including limited data types, variable names limited to 8 characters, labels limited to 40 characters and value to 200 characters. There is also no direct way to store metadata (which provokes the use of define.xml). One other restriction is that XPT is not “human-readable” as its binary and may requires sponsors to maintain SAS environments to be able to read it. Finally, despite all these weaknesses, the real and only advantage of XPT right now is that it is the only proven and mandated in the industry, so all systems or organization structures are built based on it and sponsors perfectly know how to deal with it.

To address these issues, CDISC developed Dataset-JSON as a modern potential replacement for XPT. This JavaScript Object Notation (JSON) format is more modern and is widely used for data exchange. It is a text-based and human-readable format that overcomes a lot of XPT restrictions (limited characters, minimal metadata and so on...) The goal is to be technology-neutral, so it is easy to use many programming languages and technology. Based on the CDISC Dataset-XML v1.0 specification, it could allow us to meet regulatory requirements for submission while also supporting broader data exchange needs. It has already been tested in hackathons that produce open-source tools for converting it into formats like SAS ® datasets, R data frames or Python pandas. Sponsors already gave approbation about potential improvement in efficiency, cost savings, and alignment with digital data ecosystems. It offers a modern alternative that is easier to develop against and well-suited to API-based access patterns. While APIs are not our focus here, it can be worth noting some of their advantages. Unlike static files, APIs let teams pull only the pieces of data they need, whether that is a full column or a subset of data, The big win here is speed, alongside making it easier to connect with modern tools and system.

This summary table from *Dr Deng - JSON vs SAS XPT Data Formats in Clinical Trial Data Submissions,* is a good hint on the key challenges and implications it raises:

	SAS® XPT	*Dataset-JSON*
Data Model	Binary flat-table format (one row per record)	Text-based JSON schema supporting tabular (and potentially nested) data
Readability	Binary, not human-readable (needs software to interpret)	Human-readable JSON text; easy to inspect/validate manually
Metadata	Limited to variable name/label in file, requires a separate define.xml for full metadata	Can link to CDISC define-XML and is extensible for rich metadata
Exchange Method	File-based only (transport file via eCTD)	Supports both file-based and web/API exchange
Tool support	Native support in SAS, some R/Python libraries can read XPT	Broad support in many languages and tools (e.g. JSON libraries in SAS, R, Python...)
File size	Fixed format (binary), typically larger uncompressed, but can be zipped	JSON is text and can be large if uncompressed, but CDISC JSON schema and compression often yield smaller datasets (pilot noted reduced size)
Authorities' acceptance	Current standard, required by authorities for SDTM/ADaM submissions	Currently under evaluation, not yet accepted
Standard versions	Fixed to SAS XPORT v5	Extensible schema (CDISC released v1.1 in Dec 2024)

Figure 1: SAS® XPORT and JSON comparisons

Hackathons have already proven to be a powerful driver for advancing new standards like Dataset-JSON. A hackathon is an event where people form diverse backgrounds and companies come together in a focused and hands-on environment. The goal is to propose some prototype solutions, test ideas and to finally share feedback. For Dataset-JSON, CDISC has organized hackathons that delivered concrete results. These events demonstrated the format's flexibility by producing open-source tools capable of converting Dataset-JSON into widely used structures like SAS® datasets, R data frames and Pythons pandas, as mentioned before. CDISC was satisfied with the results of these hackathons which proved that it can be easily integrated into existing workflows, with participants submitting working tools and perhaps more importantly, giving the confidence that Dataset-JSON can support real-world regulatory and research needs.

Looking ahead, next logical step was to promote a Dataset-JSON Viewer hackathon. Indeed, while Dataset-JSON offers clear advantages over the mandated XPT format, it remains, as its core, just a dataset transport format. This means that it is designed first and foremost to move data from one place to another in a standardized way, but not necessarily to be visualized by people by a single look. Opening a raw JSON file exposes itself to a dense block of text that is difficult to interpret without the right tools. As today, working with XPT often requires specialized software, a viewer designed for Dataset-JSON will lower that barrier, enabling anyone to inspect data directly.

During the CDISC Dataset-JSON-Viewer hackathon (Q4 2024), our team delivered a proposal for a viewer. We've built on that idea with two pieces: a viewer that opens Dataset-JSON natively, and a rules engine that runs checks using CORE (CDISC Open Rules Engine), so you can catch issues early, long before submission.

DATASET-JSON-VIEWER

Our motivation for creating Dataset-JSON-Viewer came from two main points. First, we wanted to replicate the core functionalities of the SAS® viewer. Indeed, many users in clinical and statistical workflows are used to the SAS viewer for quick inspection of data. This is why we wanted to provide a familiar alternative that reproduces the essential features such as browsing, searching, sorting, filtering. This ensures that users continue to work in a known environment, while benefiting from the flexibility of an open-source solution. By building the tool in R Shiny, we take advantage of its powerful interactive framework. It enables us to create a dynamic and reactive interface, making it possible to handle JSON datasets in a responsive and user-friendly way. In addition, R Shiny opens the door to data manipulation, validation and visualization. This means that more than just a static replacement for SAS® it is also a foundation to be extended to meet potential future user needs.

The tool consists of two panels: the landing page and the viewer. When you first arrive on the landing page, the File Management section lays out a clear starting point. You pick the format you are working with, as our viewer now handles NDJSON, XPT and sas7bdat as well, and you can then browse your files and upload them. The moment the upload is completed, the app gives you some context with a summary of what you just uploaded. For JSON specifically, the summary also surfaces key metadata.

The easiest way to start looking at your data is to filter it, and the app gives you two friendly ways to do that. First, user can manipulate visual filters. As you click and type (or move some sliders for numerical variables), the table updates in real-time, non-matching rows step out of view, while your original dataset stays untouched. It is useful for quick checks and as nothing is permanent, you can stack filters, remove them, and wander around your data without worrying you'll break anything. It also allows the filtering system to be quicker and more efficient. Then, when you are ready to define the exact slice of data that should drive downloads and analysis, you switch to R-syntax filters. These create a real subset of data.

```
AGE > 65 && REGION == "EU" && ARM == "Placebo".
```

A live indicator keeps you informed about what you have selected: "You currently have X out of Y observations". That count is the precise number of rows that will be exported or analyzed, so there is never a mismatch between what you see and what you download.

To add some extra-value to the viewer, you can add conditional row highlighting based on one condition (for now). Highlights apply on top of your current view, so they work equally well with visual filters and R-syntax filters. Finally, the Live Descriptive Statistics module builds directly on that R-filtered subset. As soon as your subset changes, the module regenerates summary tables and plots in real time, so you can watch the distribution, add a condition, at once. If you want comparisons, you can also specify an optional grouping variable to break results out by category.

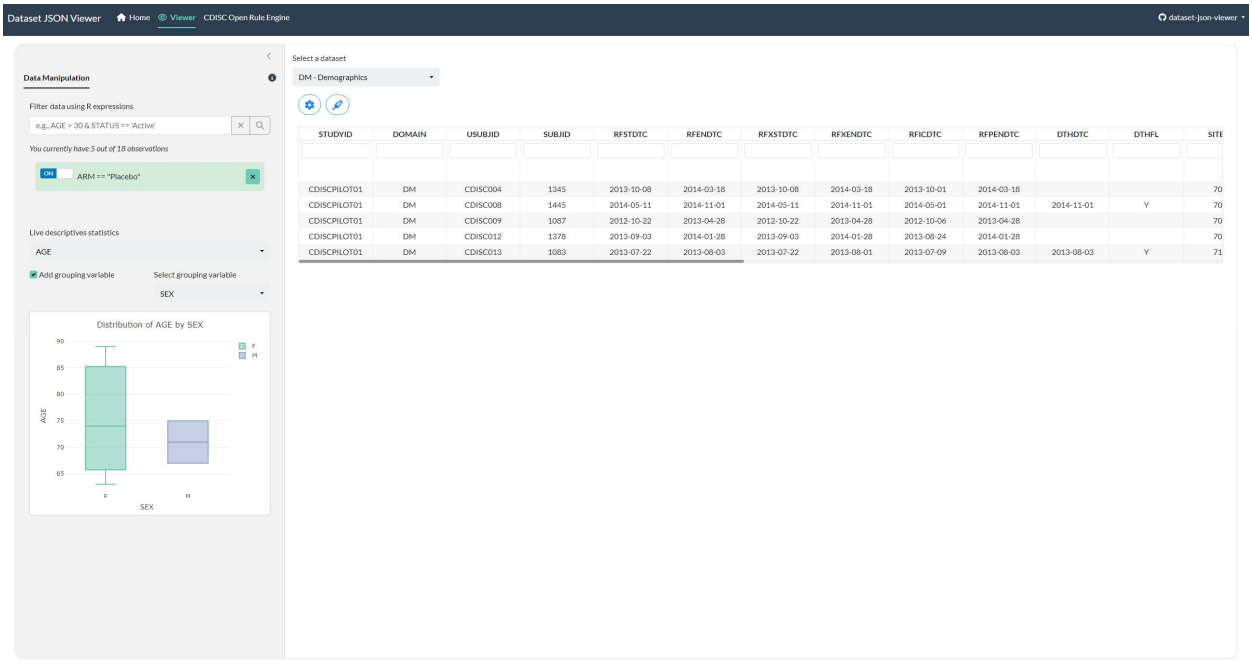

Figure 2: Viewer panel of Dataset-JSON-Viewer

GOING FURTHER: CORE INTEGRATION

The key question was to investigate where to get real extra value of JSON format within this viewer. One logical following path was the implementation of CDISC Open Rules Engine (CORE). It is a component of CDISC Foundational Standards, designed to provide clear guidance for their correct implementation in studies. To support this, the CORE aims to deliver a set of executable Conformance Rules for each Foundational Standard, along with a reference open-source execution engine available through the CDISC library. It ensures consistency across implementations. It consists of both the Rules; human-readable specifications and executable; and the Engine, an open-source application that runs the Rules against a clinical data package and returns a report. Hosted on GitHub under the CDISC Open-Source Alliance, the Engine can be deployed across several environments such as cloud or desktop and is supported by an optional user-interface. The idea is to let users choose how they really want to implement and use the tool by using the native user interface or create their own with some added features and functionalities.

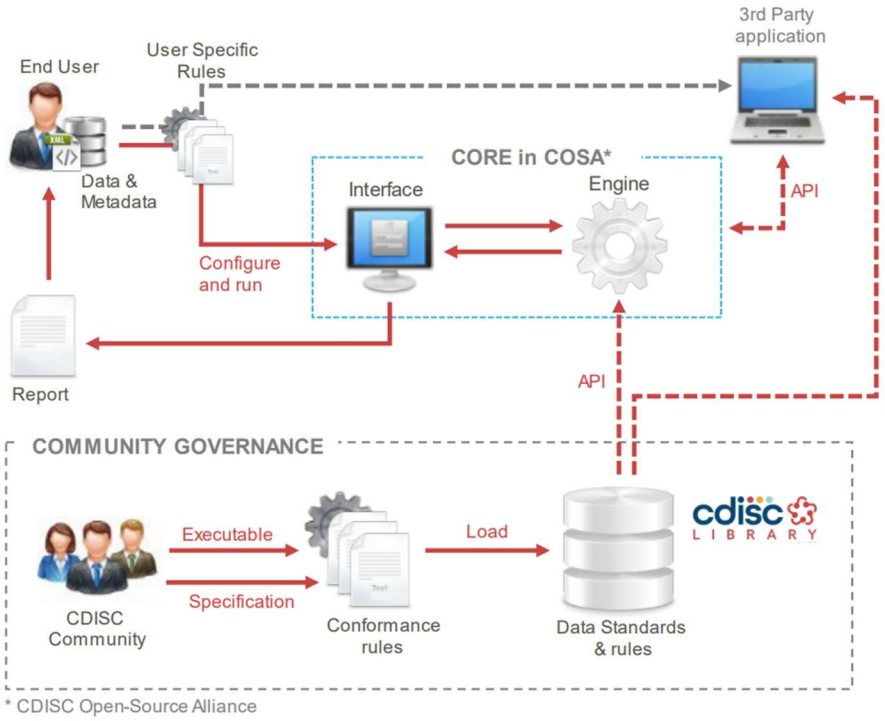

Figure 3: CORE Project Concept Diagram

In day-to-day use, the workflow is straightforward: obtain or update locally cached rules and related metadata; execute a validation run against a folder (or selected files) of study data; save results in a consumable format (JSON or Excel). Executable releases include a core binary so teams can run validations without a Python environment, e.g.,

```
./core validate -s sdtmig -v 3-4 -d ./xpt
```

CORE's documentation also emphasizes that report outputs are available in Excel for day-to-day review, with JSON outputs supported for programmatic downstream use. Before validating, a one-time (or periodic) cache step pulls rules, controlled terminology (CT), and supporting metadata using a CDISC Library API key:

```
python core.py update-cache
```

Validation command and key parameters

The `validate` command accepts a concise set of arguments that let teams target standards, select data and metadata, tailor rule scope, and control output.

Standards and versions. Specify the standard (e.g., SDTMIG, ADaM, TIG) and its version; a TIG sub-standard (SDTM/SEND/ADaM/CDASH) can be provided when applicable.

Data and define. Point to a directory of datasets or individual files; optionally reference Define-XML and indicate its version. An option is provided to schema-validate Define-XML.

Controlled Terminology. Provide one or more CT package identifiers. If Define-XML 2.1 is supplied, CORE will use the CT indicated by the define; with Define-XML 2.0, a CT package should be specified explicitly.

Medical dictionaries (optional). Paths can be provided to WHODrug, MedDRA, LOINC, MED-RT, UNII, and

SNOMED (with version/URL/edition options) to enable dictionary-backed checks.
Rule selection. Teams may restrict execution to a subset of rules or include local/custom rules located on disk; custom standards can be referenced after being added to the cache. Discovery commands (list-rules, list-rule-sets, list-ct) help browse what is available.
Reporting. Choose output destination and format (JSON or XLSX). A raw JSON mode emits the engine's native structure for integration use cases.
Progress and logs. Options control progress display and log verbosity for console usage and CI pipelines.

Example (Define-XML-aware SDTMIG run to Excel)
A representative validation that leverages Define-XML and emits an Excel report:

```
./core validate \
  -s sdtmig -v 3-4 \
  -d ./json \
  -dxp ./define.xml -dv 2.1 -vx \
  -ct sdtmct-2024-06-28 \
  -o ./reports/run_2025-10-01 -of json
```

The command expresses standard/version, data location, Define-XML path and version (with optional schema validation), CT selection (or implicit CT via Define-XML 2.1), and a report target. The same options are available via python core.py validate … when running from source. We integrated CORE into a Shiny front end (Figure 4), so reviewers interact through a simple UI instead of the terminal. In the app, users choose the Standard and Version, pick a Controlled Terminology package, and specify the MedDRA version from dropdowns. They then set the data path to the study datasets and the output path for the report. When they click "Run validation," Shiny assembles the corresponding command and invokes the CLI with those arguments. For now, only these parameters are implemented, with additional options planned for later iterations.

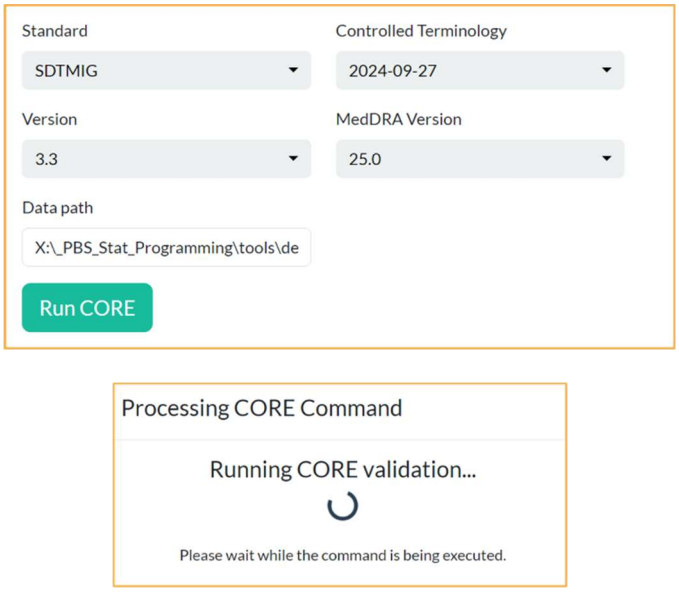

Figure 4: Shiny interface to run Open Rule Engine

The Dataset-JSON viewer can generate CORE reports inside each project, every run is saved to a small SQLite database, so the project carries its full issue history with it. When a new report comes in, the app lines it up with what is already there and refreshes the picture automatically: new issues appear, ongoing ones stay visible, anything that has disappeared is marked as resolved. The tool will then generate a dashboard that gives clear signals. You can see a completion rate (number of closed tasks over total number). There are headlines counter and plots which summarize how many tasks you still have pending, allowing you to check the distribution per dataset as well.

In the interactive summary, clicking a row name takes you to the page for that rule or dataset. Reviewers can open, close or reopen tasks. The system records who made the change and when. At the end, we have four statuses of tasks: open, closed, re-open, resolved. Closing an issue needs to be justified with some text, that will be called explanation ("fixed in source data," "per protocol," "vendor fix planned"), so decisions are understandable months later. The idea was also to have a built-in comment thread next to rule summary and affected records. As JSON is AI-friendly, we can use historical JSON reports to guide an assistant that suggests explanation to close tasks, with no need to expose any study datasets. Because these reports follow a consistent schema (rules, severities, counts, dataset/variable context), the assistant can recognize patterns and map similar findings to likely explanations. Also, JSON makes it easy to jump from a failed check straight to the exact cell that needs attention. Every finding comes with just enough coordinates to be dataset name, the record keys (like USUBJID/SEQ), the variable, and the value that triggered the rule. With that, the viewer can open the right dataset, scroll to the correct row, and briefly highlight the precise cell so reviewers see the problematic value immediately. Because those JSON pointers are stable, you can copy a deep link into a ticket or comment, and anyone who follows it arrives at the same place, even after new runs. The result is faster triage, clearer hand-offs, and a clean audit trail of what was seen and where.

Because everything is organized at the project level, several users can work at once without stepping on each other. One person can triage new items while another focuses on long-running issues; both see the same up-to-date status and the same conversation. We plan to add an export button so that at any moment, the team can produce a clean summary for stakeholders: progress since the previous run, what remains open by severity, the current state and full activity history exported as CSV or PDF.

CONCLUSION

This work sets out to show that pairing Dataset-JSON with an open viewer and executable rules makes real-time validation practical and materially streamlines the path to submission. The evidence across the paper demonstrates that the combination delivers earlier defect discovery, tighter feedback loops, and a review experience that stays close to how teams already work while removing long-standing XPT constraints.

Concretely, the viewer provides fast, human-friendly access to modern and legacy formats, with instant filtering, search, sorting, conditional highlighting, and live descriptive statistics, so reviewers can interrogate exactly the slice of data they intend to export or analyze. Layered on top, the CORE integration assembles standards, CT and MedDRA versions, and data paths into a single action, produces structured reports, and records every run in a lightweight project database that tracks opens, closures, regressions, and rationale. This makes validation continuous rather than episodic, converts findings into accountable, auditable work, and shortens the iteration from “issue found” to “issue resolved.”

Importantly, while CORE can execute rules against XPT files, adopting Dataset-JSON amplifies the benefits in several practical ways. JSON is text-based and predictable across tools, carries richer, explicit metadata, and avoids legacy transport limits that complicate interoperability. It is also web- and API-friendly, which enables selective retrieval (only the variables or records you need), streaming validation as data lands, easier diffing and version control, and simpler automation without a dependency on specific proprietary runtimes. In short, CORE on XPT works; CORE on Dataset-JSON works better, faster to parse, richer in context, and easier to integrate into modern pipelines.

REFERENCES

1. Clinical Data Interchange Standards Consortium. CDISC | Clear Data. Clear Impact. [Internet]. [cited 2025 Oct 1]. Available from: https://www.cdisc.org/ cdisc.org
2. Hume S. No more XPT? Piloting new Dataset-JSON for FDA submissions. Clinical Leader [Internet]. 2023 [cited 2025 Oct 1]. Available from: https://www.clinicalleader.com/doc/no-more-xpt-piloting-new-dataset-json-for-fda-submissions-0001 clinicalleader.com
3. CDISC Data Exchange Standards Team. CDISC Dataset-JSON specification, version 1.1. [Internet]. 2024 Dec 5 [cited 2025 Oct 1]. Available from: https://cdisc-org.github.io/DataExchange-DatasetJson/doc/dataset-json1-1.html cdisc-org.github.io
4. Deng C. JSON vs SAS XPT data formats in clinical trial data submissions. On Biostatistics and Clinical Trials [Internet]. 2025 May 1 [cited 2025 Oct 1]. Available from: https://onbiostatistics.blogspot.com/2025/05/json-vs-sas-xpt-data-formats-in.html onbiostatistics.blogspot.com
5. De Donder N, Van Reusel P. CDISC Conformance rules and the CDISC Open Rules Engine: continuing the road to adoption. PHUSE EU Connect 2023, Paper SI04 [Internet]. 2023 [cited 2025 Oct 1]. Available from: https://www.lexjansen.com/phuse/2023/si/PAP_SI04.pdf lexjansen.com
6. Clinical Data Interchange Standards Consortium. CORE | CDISC (CDISC Open Rules Engine) [Internet]. Page updated 2024 Jul 1 [cited 2025 Oct 1]. Available from: https://www.cdisc.org/core cdisc.org

CONTACT INFORMATION

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

Author Name: Hugo Signol
Company: Cytel
Email: hugo.signol1@cytel.com

Co-Author Name: Sebastià Barceló
Company: Cytel
Email: sebastia.barcelobauza@cytel.com