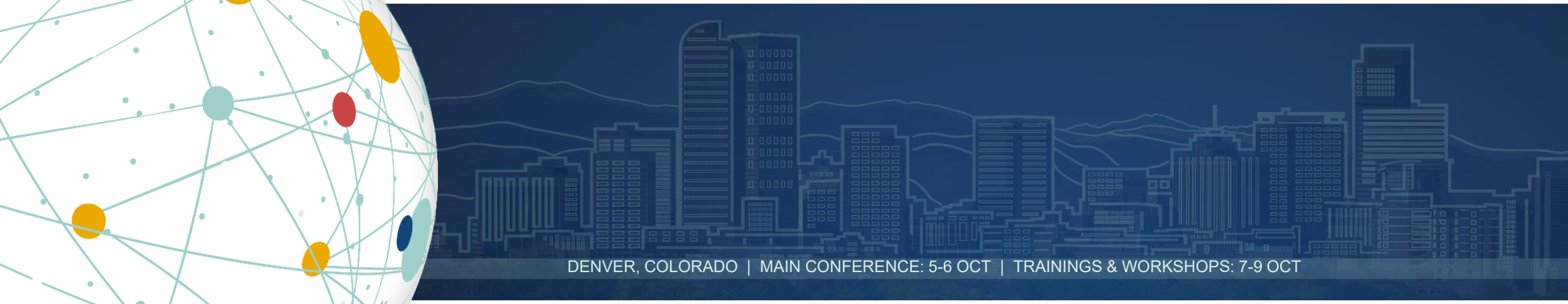


cdisc®

2026 US Interchange

THE FUTURE IS CONNECTED: STANDARDS AND AI POWERING DIGITAL TRANSFORMATION



Beyond XPT: Reimagining the Clinical Submission for 2041

Raj Chennamaneni · Sudhir Kedhare · Stephen Hamburg

Jazz Pharmaceuticals | Denver 2026


GENOMICS
A T G C T A
G C A T G C
T T A C G T
WEARABLES
72
bpm
7,346
steps
SENSORS
98
SpO2
72
HR
16
RR
REAL-WORLD DATA
IMAGING
XPT

Agenda

01 Why XPT is running out of road

02 What the data looks like now

03 Nine candidate formats, in depth

04 Seven layers for 2041

01

PART ONE

Why XPT is running out of road

PART ONE

SAS XPT: thirty-six years as the default

01

- 1989: SAS Institute releases the Version 5 transport format
- Fixed 80-byte records with IBM floating-point numbers
- 1999: FDA guidance introduces XPT for electronic submissions
- 2014: Declared the preferred format for regulatory submissions
- 2025: FDA names Dataset-JSON v1.1 as a potential alternative
- Every validator, pipeline and review tool is built around it

No competing format displaced it for thirty-six years

cdisc®

PART ONE

01

Where XPT falls short

- Two data types only: numeric and character
- Variable and dataset names limited to 8 characters
- Labels limited to 40 characters; text values to 200 bytes
- ASCII only: no non-Latin characters for global trials
- No API-based exchange; files move as static attachments
- Row-based and uncompressed: large files, slow to read

Built for 1989 hardware, not 2041 data

cdisc®

02

PART TWO

What the data looks like now

PART TWO

02

The data that outgrew the format

- Wearables: glucose monitors, cardiac patches, actigraphy
- Longitudinal records: diagnoses, medications, labs, imaging
- Real-world data: insurance claims, registries, pharmacy
- Biomarker and genomic data: 100–200 GB from one patient
- Data arrives continuously, not only at scheduled visits
- Nested, high-frequency and multilingual by nature

Fixed 80-byte records were never built for this

cdisc®

03

PART THREE

Nine candidate formats, in depth

01

SUBMISSION LAYER

Dataset-JSON v1.1

Dataset-JSON v1.1: what it is

- JSON-based exchange format designed to replace XPT
- Developed by CDISC with FDA and PHUSE; pilots met exchange needs
- Tabular like XPT: variable metadata first, then one row per record
- Adds integer, float, Boolean, date, datetime and URI types
- UTF-8 text, with no 8-character name or short label limits
- Built for REST API exchange as well as file transfer

Same tabular idea as XPT, with modern types and text

Dataset-JSON v1.1: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

3 AGE keeps its integer type

Dataset-JSON file

1 Header row → column metadata

```
{"name": "USUBJID", "dataType": "string"}
{"name": "SEX", "dataType": "string"}
{"name": "AGE", "dataType": "integer"}
```

2 Each record → one row array

```
["1001", "M", 23]
["1002", "F", 25]
["1003", "M", 24]
```

One readable JSON file holds the metadata and the rows

Dataset-JSON v1.1: strengths and limitations

Strengths

- Open source under the MIT licence
- Human-readable and smaller than XPT
- Full Unicode and richer data types
- Read by almost any language via REST

Limitations

- FDA acceptance expected in 2026–2027
- Small numeric precision differences in FDA testing
- Transition cost from XPT tooling
- Still larger than Parquet or Arrow

Solves XPT's limits; adoption waits on the FDA

02

SUBMISSION LAYER

NDJSON

NDJSON: what it is

- Newline-delimited JSON: one valid JSON value per line
- Streaming form of Dataset-JSON v1.1, released alongside it
- First line holds metadata; each later line holds one record
- Uses the .ndjson, .jsonl or .jsonlines file extensions
- Parsed line by line, so memory use stays small
- Suits parallel processing and pipeline architectures

Same content as Dataset-JSON, read one line at a time

NDJSON: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

NDJSON file

1 Line 1 → metadata

```
{"name": "DM", "columns": [...]}
# one JSON value per line
# parsed once, kept small
```

2 Lines 2–4 → one record each

```
["1001", "M", 23]
["1002", "F", 25]
["1003", "M", 24]
```

3 Read or append one line at a time

Streams line by line; memory use stays small

NDJSON: strengths and limitations

Strengths

- Low memory: streams, never loads all
- Append without rewriting the file
- A bad line affects one record
- Spark and Hadoop split it in parallel

Limitations

- Larger than binary formats like Parquet
- Slow when only a few columns are needed
- Validation needs a separate JSON Schema

Scales Dataset-JSON to very large files

cdisc®

03

A CAUTIONARY TALE

Dataset-XML v1.0

Dataset-XML v1.0: what it is

- Released by CDISC in 2014 as part of the ODM family
- The first non-proprietary alternative to SAS XPT
- Define-XML metadata first, then the study data
- An ItemGroupData element wraps each record
- An ItemData element wraps each individual value
- Files are validated against accompanying XSD schemas

Every value wrapped in its own opening and closing tags

cdisc®

Dataset-XML v1.0: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

3 One value, two lines of tags

Dataset-XML file

1 Define-XML → variable metadata

```
<ItemDef Name="USUBJID" DataType="text"/>
<ItemDef Name="SEX" DataType="text"/>
<ItemDef Name="AGE" DataType="integer"/>
```

2 Every value wrapped in tags

```
<ItemGroupData ItemGroupOID="IG.DM">
<ItemData ItemOID="AGE" Value="23"/>
... repeated for every value
```

Tags around every value make files large and slow

cdisc

Dataset-XML v1.0: strengths and limitations

Strengths

- Removed XPT's name, type and length limits
- Handled hierarchical structures
- Schema validation built in

Limitations

- Tags inflated every file
- Parsing tags slowed processing
- Never adopted by industry

Solved the technical problems, but at a high cost

cdisc®

04

INTERNAL LAYER · STORAGE

Apache Parquet

Apache Parquet: what it is

- Created by Twitter and Cloudera in 2013; now an Apache project
- Columnar: all values of one variable are stored together
- Rows: (1001, M, 23) · (1002, F, 25) · (1003, M, 24)
- Columns: USUBJID (1001, 1002, 1003) · SEX (M, F, M) · AGE (23, 25, 24)
- Mean age reads only the AGE column, not every record
- A file footer holds the schema, types and min/max statistics

Organised by column, not by row

Apache Parquet: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

3 A query reads only the AGE column

Parquet file

1 Stored column by column

```
USUBJID   1001 | 1002 | 1003
SEX       M | F | M    (dictionary)
AGE       23 | 25 | 24
```

2 Footer → schema and statistics

```
AGE: int32, min 23, max 25
SEX: string, 2 distinct values
mean(AGE) reads the AGE column only
```

Columns compress well and queries skip what they don't need

Apache Parquet: strengths and limitations

Strengths

- 2–5× smaller than XPT, 5–10× smaller than Dataset-JSON
- Repeated values compress to almost nothing
- Skips data it does not need
- Python, R, Spark, Hive, DuckDB

Limitations

- Binary; needs tools to view
- No regulatory or CDISC mapping
- Weak for row-level updates
- Limited native SAS support

Built for fast reads of large data

cdisc®

05

INTERNAL LAYER · IN MEMORY

Apache Arrow

Apache Arrow: what it is

- Launched in 2016 by developers of pandas, R, Spark and others
- An in-memory columnar format, the same in every language
- Normally data is serialized to bytes, then rebuilt by the reader
- Arrow lets programs share memory directly: zero-copy reads
- Arrow files use the .arrow or .feather extensions
- Complements Parquet: Parquet on disk, Arrow in memory

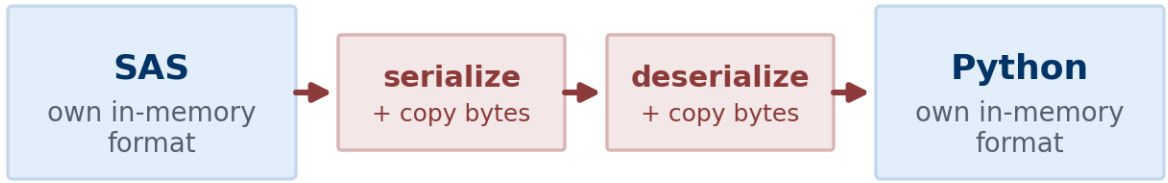
One memory layout, shared across languages

APACHE ARROW

Zero-copy: share memory, don't serialize it

The same columnar buffer is read in place by SAS, R and Python — no encode, decode, or duplicate copies.

WITHOUT ARROW · COPY + SERIALIZE

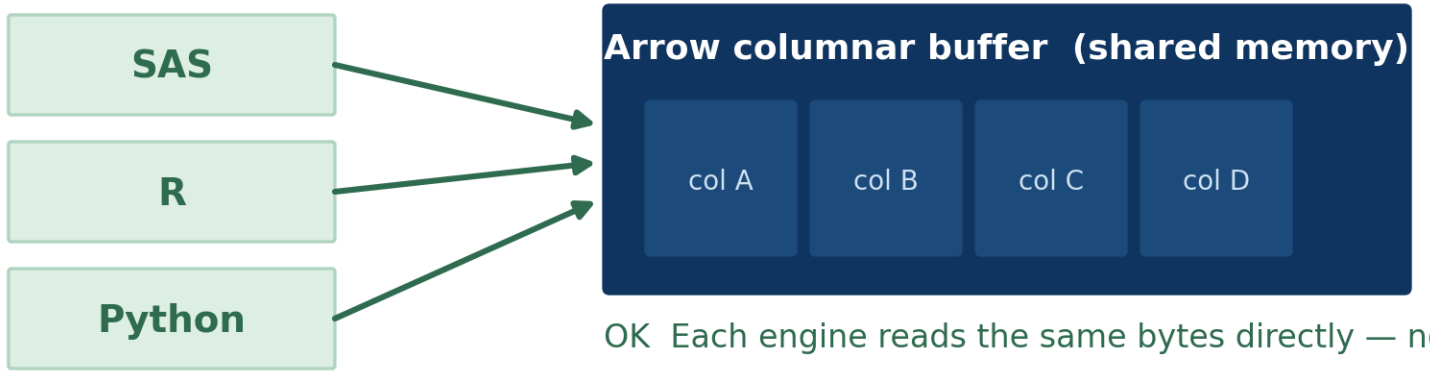


X Every hand-off re-encodes and duplicates the data, burning CPU and memory.

WHY IT MATTERS

- No CPU wasted on (de)serialization
- No RAM wasted on duplicate copies
- One standard: SAS, R, Python
- Columnar layout = fast scans

WITH ARROW · ONE SHARED BUFFER



OK Each engine reads the same bytes directly — no serialize, no deserialize, no copy.

Apache Arrow: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

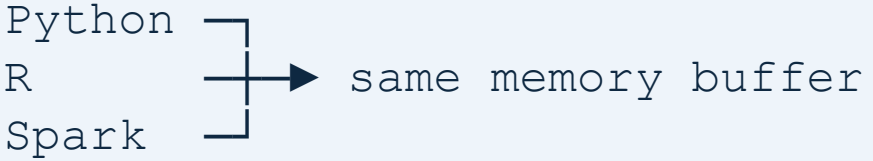
3 All languages read one AGE buffer

Arrow in memory

1 One columnar buffer per variable

```
USUBJID   ["1001","1002","1003"]
SEX       ["M","F","M"]
AGE       [23, 25, 24]
```

2 Shared without copying



No serialization and no copies between languages

Apache Arrow: strengths and limitations

Strengths

- Zero-copy reads within and across programs
- Columnar layout uses CPU cache and SIMD
- Native in Python, R, Spark and more

Limitations

- Held in memory; not a storage format
- Needs Parquet or another format to persist
- No regulatory or CDISC interest

The fastest way to move and analyse data in memory

06

INTERNAL LAYER · STREAMING

Apache Avro

Apache Avro: what it is

- Developed in the Apache Hadoop project; open source
- Row-oriented binary data with a JSON schema inside the file
- Built for write-heavy streaming pipelines
- Schema evolution: fields can be added or removed over time
- Old readers read new data; new readers read old data
- Part of the Hadoop and large-scale pipeline ecosystem

The schema travels with the data

Apache Avro: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

Avro file

1 JSON schema in the file header

```
{"type": "record", "name": "DM",
"fields": [..., {"name": "AGE",
                 "type": "int"}]}
```

2 Binary records, appended in order

```
record 1   1001 · M · 23   (binary)
record 2   1002 · F · 25   (binary)
record 3   1003 · M · 24   (binary)
```

3 Each record is written as it arrives

The schema travels with the data; records stream in fast

Apache Avro: strengths and limitations

Strengths

- Embedded schema prevents mismatches
- Compact binary with fast writes
- Schema evolution for long trials
- Large-scale pipeline ecosystem

Limitations

- Binary data is not human-readable
- Slow for column-level analytics
- No direct SAS integration
- No regulatory or CDISC interest

Excellent for writing and streaming, weaker for analysis

07

SUBMISSION LAYER · REAL-WORLD DATA

HL7 FHIR

HL7 FHIR: what it is

- Fast Healthcare Interoperability Resources, developed by HL7
- Work began in 2011; R4 released in 2019 and R5 in 2023
- Data is grouped into resources: Patient, Observation, Condition
- Linked resources form a connected graph for each patient
- Represented as JSON or XML and exchanged over REST APIs
- FDA has explored FHIR for real-world data submissions

The global standard for clinical care data

HL7 FHIR: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

FHIR resources

1 Each subject → Patient resource

```
{"resourceType": "Patient",
"id": "1001",
"gender": "male"}
```

2 Each value → linked Observation

```
{"resourceType": "Observation",
"code": LOINC 30525-0 (Age),
"subject": "Patient/1001", 23 years}
```

3 One row becomes linked resources

Rich, linked care data that must be mapped back to SDTM

HL7 FHIR: strengths and limitations

Strengths

- Global standard across health systems
- REST APIs for real-time exchange
- Bound to SNOMED CT, LOINC, RxNorm, ICD-10
- Models complex care relationships

Limitations

- Complex mapping to SDTM and ADaM
- No common mapping across sponsors
- Linked resources increase data size

Rich care data, but a second standard to maintain

08

INTERNAL LAYER · REVIEWER TOOLS

SQLite

SQLite: what it is

- Created by D. Richard Hipp in 2000
- A serverless, self-contained relational database in one file
- One of the most widely deployed databases in the world
- Runs on Windows, Linux, macOS, Android and iOS
- A single file could hold every SDTM and ADaM dataset
- Reviewers can join and query datasets with plain SQL

A whole study in one queryable file

SQLite: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

3 Reviewers query AGE with SQL

SQLite database: study.db

1 Stored as a table

```
CREATE TABLE dm (USUBJID TEXT,
  SEX TEXT, AGE INTEGER);
-- all SDTM and ADaM tables in one file
```

2 Queried with plain SQL

```
SELECT SEX, AVG(AGE) FROM dm
GROUP BY SEX;
→ F 25.0 | M 23.5
```

A whole study in one file that anyone can query

SQLite: strengths and limitations

Strengths

- One file instead of dozens of datasets
- Smaller than plain Dataset-JSON
- Built into Python, R and most languages

Limitations

- A database, not an exchange standard
- No built-in clinical metadata
- One writer at a time; no streaming

Convenient for querying, not for exchanging data

cdisc®

09

INTERNAL LAYER · SCIENTIFIC DATA

HDF5

HDF5: what it is

- Released in 1998 for very large scientific datasets
- Used in imaging, genomics, astronomy and physics
- Organised like folders and files: groups and datasets
- Each group and dataset carries its own metadata
- Handles gigabytes to terabytes within one file
- Fits genomics, sensor, MRI and CT data from trials

A file system inside a single file

HDF5: how the data is stored

Source dataset: DM

USUBJID	SEX	AGE
1001	M	23
1002	F	25
1003	M	24

3 Read just part of the AGE dataset

HDF5 file: study.h5

1 Groups and datasets, like folders

```
/DM/USUBJID   /DM/SEX   /DM/AGE
/signals/1001/ECG   (large arrays)
each dataset carries attributes
```

2 Metadata attributes, partial reads

```
AGE.attrs: label="Age", units="years"
read AGE[0:2]  →  [23, 25]
without loading the whole file
```

Hierarchy, metadata and partial reads for very large data

HDF5: strengths and limitations

Strengths

- Hierarchy plus rich metadata
- Chunked compression, often ~10×
- Partial reads of large arrays
- Python, R, MATLAB; HDFView browser

Limitations

- Steep learning curve
- No native SAS support
- No regulatory or CDISC mapping
- Limited concurrent writing

Built for data too large and complex for tables

04

PART FOUR

Seven layers for 2041

L1

PART FOUR · LAYER 1 OF 7

Layer 1: Submission

Role in 2041

- Standard SDTM and ADaM to regulators
- Open, validated and human-readable
- Replaces XPT as the default
- Covers structured trial data

Formats and readiness

- Dataset-JSON v1.1; NDJSON for large files
- FDA acceptance expected 2026–2027
- Verify precision before switching
- Define-XML stays the metadata

Dataset-JSON replaces XPT for submission

cdisc®

Layer 2: Analytics

L2

Role in 2041

- Store and analyse integrated data
- Fast queries across many studies
- Feeds TLFs and exploratory work
- Pooled and cross-study analyses

Formats and readiness

- Parquet on disk, Arrow in memory
- Mature tools: Python, R, Spark, DuckDB
- Needs a shared CDISC metadata convention

Parquet for storage, Arrow in memory

cdisc®

clinfolio

Formats Processing Signals Roadmap GitHub

Open source · Python · Apache-2.0

Clinical data at scale, with its metadata intact.

clinfolio stores CDISC datasets in Parquet, Avro and HDF5, and keeps labels, codelists, origins and lineage attached through every filter, join and derivation.

```
$ pip install clinfolio  (coming soon)
```

```
advs.filter(PARAMCD == "SYSBP")
```

advs/ · 48.2 M rows · Parquet

USUBJID Unique Subject Identifier	PARAMCD Parameter Code	AVISIT Analysis Visit	AVAL Analysis Value	BASE Baseline Value	CHG Change from Baseline
101-001	SYSBP	Week 8	128	139	-11
101-001	PULSE	Week 8	71	74	-3
101-002	SYSBP	Week 8	141	146	-5
101-002	DIABP	Week 4	84	88	-4
102-004	SYSBP	Week 8	133	137	-4

Rows drop out. Labels, codelists and origins stay on every column; **green marks a derived variable.**



EEG_C3 · 256 Hz · HDF5 → linked to SDTM

Layer 3: Real-world data

L3

Role in 2041

- Bring EHR, claims and registry data in
- External controls, pragmatic trials
- Keep links to source records

Formats and readiness

- HL7 FHIR for exchange from health systems
- SDTM mapping not yet standardised
- CDISC and HL7 work under way

FHIR carries real-world data into submissions

Layer 4: Streaming

L4

Role in 2041

- Continuous wearable and eCOA data
- Validate records as they arrive
- Load incrementally, not in batches
- Absorb schema changes in long trials

Formats and readiness

- Avro for pipelines with changing schemas
- NDJSON for line-by-line JSON streams
- Converted to Parquet for analysis

Avro and NDJSON for sensor and device feeds

cdisc®

Layer 5: Reviewer tools

Role in 2041

- Query across datasets quickly
- No specialised software needed
- A whole study in one place
- Faster reviewer questions and QC

Formats and readiness

- SQLite as a single-file companion
- Generated from submission datasets
- Complements Dataset-JSON
- Needs an agreed table layout

SQLite gives reviewers one queryable file

Layer 6: Scientific data

L6

Role in 2041

- Raw signals, imaging and genomics
- Gigabytes to terabytes per study
- Summaries flow into SDTM and ADaM
- Signals linked to SDTM subjects

Formats and readiness

- HDF5: groups, metadata, compression
- Partial reads of very large arrays
- Specialised skills needed
- Summaries submitted as Dataset-JSON

HDF5 for data too large for tables

cdisc®

PART FOUR · LAYER 7 OF 7

Layer 7: Legacy

Role in 2041

- Read and resubmit historical studies
- Integrated summaries with older data
- A bridge while guidance changes

Formats and readiness

- SAS XPT v5 until fully retired
- Convert to Dataset-JSON on demand
- Keep validated conversion tools
- Archive originals for audit

XPT stays until the last legacy study retires

PART FOUR

What to do before 2041 arrives

- Plan for a portfolio of formats, not a single successor
- Adoption cost decides, not technical capability
- Separate the submission and internal layers deliberately
- Pilot Dataset-JSON now and check numeric precision
- Build internal pipelines on Parquet and Arrow
- Watch 2026–2027 for FDA acceptance decisions

The two-layer split is available today

Questions?

