

PREPRINT

Data Sheet Structure (DSS) Specification Version 1.0

Vincenzo Manto

May 31, 2026

DOI: <https://doi.org/10.5281/zenodo.19659516>

<https://github.com/Datastripes/DataSheetStructure>

Abstract

This document specifies the Data Sheet Structure (DSS) version 1.0, an application-layer, plain-text data serialization format optimized for representing multi-sheet, sparse tabular datasets. DSS is designed to maintain native human readability and maximize compatibility with line-oriented version control systems (e.g., Git) by eliminating arbitrary formatting padding and treating spreadsheets as a collection of coordinate-mapped data matrices.

Status of This Memo

This Internet-Draft is submitted in full conformance with RFC guidelines. Distribution of this document is unlimited. Technical feedback and implementation reports are invited.

Table of Contents

- 1. Introduction and Rationale
- 2. Core Architectural Principles
- 3. Protocol Syntax & Technical Specification
 - 3.1. Metadata Frontmatter
 - 3.2. Sheet Declarations
 - 3.3. Coordinate Anchor Notation (@)
- 4. Reference Syntax Example
- 5. Formal Grammar (EBNF)
- 6. Technical Matrix & Comparative Analysis
- 7. Implementation Requirements & State Machine
- 8. IANA Considerations & Media Type Registration

1. Introduction and Rationale

Modern distributed version control systems encounter structural inefficiencies when processing industry-standard tabular data formats. The Comma-Separated Values (CSV) format lacks native multi-sheet encapsulation and introduces substantial line-diff noise via comma-padding in sparse datasets. Conversely, compressed binary architectures such as Office Open XML (XLSX) and OpenDocument Spreadsheet (ODS) are opaque to line-based differential parsing, generating monolithic merge conflicts and requiring specialized runtime environments.

The Data Sheet Structure (DSS) specification bridges this architectural gap by defining a coordinate-based plain-text encoding standard that maps multi-sheet sparse matrix abstractions directly to

deterministic, line-buffered data blocks.

2. Core Architectural Principles

Implementations of the DSS specification **MUST** adhere to the following foundational design constraints:

- **Human-Centric Transparency:** Data payloads **MUST** be natively readable and editable within basic ASCII/UTF-8 terminal environments without secondary decoding layers.
- **Differential Determinism (Git-Native):** Discrete cellular alterations within a dataset **MUST** map to predictable, single-line deltas within upstream tracking structures.
- **Sparse Optimization:** Volumetric constraints are optimized by excluding unallocated or null-value cellular space. Spatial coordinate positioning dictates rendering, rather than physical character padding.
- **Algorithmic Parsimoniousness:** Compliant parser implementations must maintain low cognitive and computational overhead, executing deterministically in fewer than 100 logical lines of code.

3. Protocol Syntax & Technical Specification

3.1. Metadata Frontmatter

A DSS data stream **MAY** initiate with an optional block of YAML-compliant key-value pairs bounded by triple-hyphen delimiters (ASCII 0x2D). If present, the frontmatter **MUST** occupy the absolute apex of the file stream.

```
---
project: Global Economy Simulation
version: 1.0.0
encoding: UTF-8
---
```

3.2. Sheet Declarations

Logical sheets are encapsulated using bracketed identifiers. A DSS instance may partition an arbitrary number of distinct sheet boundaries.

Syntax: [Sheet Name] **Constraints:** Sheet identifiers **MUST** be unique within the global file scope. Duplicate declarations result in undefined parser state behavior unless handled as explicit structural errors.

3.3. Coordinate Anchor Notation (@)

Rather than handling values as sequential streams, DSS uses Coordinate Anchors based on standard spreadsheet alphanumeric (A1) layout notation.

Syntax: @ <Coordinate> (e.g., @ A1, @ C10)

Behavior: The declaration of an anchor re-indexes the internal cursor state to the defined absolute structural row/column coordinate. All subsequent comma-separated data vectors are mapped into the execution grid via incremental relative offsets from this origin point.

Collision Resolution (Last-Man-Wins):In instances where subsequent, distinct coordinate blocks define overlapping cell matrices, the sequence defined closest to EOF (End-of-File) **MUST** overwrite prior allocations.

4. Reference Syntax Example

The following block demonstrates a standard, structurally valid DSS payload containing multiple sheets, sparse spacing, and embedded anchors:

```
[Financials]
@ A1
"Period", "Revenue", "Margin"
"Q1", 10500.50, 0.22
"Q2", 12000.00, 0.25

@ E1
"Status", "Final"
"Audited", true

[Metadata_Private]
@ A1
"Internal_ID", 99823
```

5. Formal Grammar (EBNF)

The syntax of Data Sheet Structure (DSS) v1.0 is formally defined by the following Extended Backus-Naur Form (EBNF) production rules:

```
<file>          ::= [ <metadata> ] <sheet_list>
<metadata>      ::= "---" <newline> { <key_value> <newline> } "---" <newline>
<sheet_list>    ::= { <sheet_block> }
<sheet_block>   ::= "[" <name> "]" <newline> { <anchor_block> }
<anchor_block>  ::= "@" <coordinate> <newline> <csv_data>
<coordinate>    ::= [A-Z]+ [1-9][0-9]*
<csv_data>      ::= { <row> <newline> }
<row>           ::= <value> { "," <value> }
<key_value>     ::= [A-Za-z0-9_]+ ":" <value>
<value>         ::= <string> | <number> | <boolean>
```

6. Technical Matrix & Comparative Analysis

The following matrix contrasts structural capabilities across standard data formats:

Functional Vectors	CSV	XLSX / ODS	DSS v1.0
Multi-Sheet Capabilities	No	Yes	Yes
Sparse Matrix Optimization	No	Yes	Yes
Diff Engine / Git Optimization	Yes	No	Yes
Plain-Text Decoupling	Yes	No	Yes
Native Extensible Metadata	No	Yes	Yes

7. Implementation Requirements & State Machine

Compliant parsers **MUST** manage cell allocations by executing a 3-state lexical scanner utilizing an internal key-value document object map:

1. **Instantiation:** Initialize a multi-dimensional associative dictionary array (SheetMap).
2. **Token Evaluation Loop:**
 - **Sheet Match** (`^[<name>]$`): Construct or alter the target pointer context to the declared sheet identifier.
 - **Anchor Match** (`^@<coord>$`): Parse the A1 pattern alphanumeric identifier into distinct integer metrics. Map characters to `BaseCol` indices and integers to `BaseRow` indices. This initializes the localized block coordinate offset.
 - **Data Vector Selection:** Tokenize the row line sequence by parsing column characters using comma rules (complying with standard escaping conventions). For every field sequence evaluated at programmatic offset `i`, commit the execution assignment: `SheetMap[ActiveSheet][BaseRow + CurrentLine][BaseCol + i] = Value`.
3. **Comment Subroutines:** Any line sequence