PharmaSUG China 2026 - Paper DV-185

An Interactive Shiny Application for Streamlined ADaM Specification Integration and Compliant DefineXML v2.1 Generation

Dongxu Liu, Dizal Pharma

ABSTRACT

Define-XML v2.1 is the mandatory machine-readable metadata companion for clinical data submissions to regulatory agencies. Despite clear CDISC standards, manual production of Define-XML remains widespread, leading to reproducibility gaps, late-cycle defects, and submission risk. This paper presents an R-based Shiny application that automates the full Define-XML v2.1 lifecycle: metadata ingestion from structured Excel workbooks, automatic detection and dual support of both SDTM and ADaM standards, deterministic XML generation using the R `xml2` package, optional XSLT-driven PDF rendering, and a comprehensive two-tier validation framework covering over 100 metadata quality rules and the full published CDISC DD/OD rule catalogue.

The application is built entirely in R using `shiny`, `bslib`, `xml2`, `xslt`, `readxl`, `dplyr`, `stringr`, and `purrr`. It provides a multi-tab interactive UI for metadata browsing, validation, generation, and conformance checking, while preserving programmatic, headless execution capability for integration into submission pipelines. Key technical innovations include automatic SDTM/ADaM standard detection from workbook metadata, value-level metadata with flexible `WHERE_CLAUSE` and `WHERE_VARIABLE/OPERATOR/VALUE` syntaxes, ADaM `AnalysisResultDisplays` support, and structured validation findings tied to named CDISC rule IDs.

The framework demonstrates how modern R tooling can replace fragile document-based workflows with a reproducible, auditable, and submission-ready metadata production pipeline.

INTRODUCTION

The Regulatory Imperative

Regulatory submissions to global health authorities (FDA, PMDA, EMA) require structured metadata describing all analysis and tabulation datasets. Define-XML v2.1, published by the Clinical Data Interchange Standards Consortium (CDISC), provides the machine-readable format for this metadata, encoding dataset structures, variable definitions, derivations, controlled terminology, value-level conditions, and links to supportive documents.

The Define-XML standard serves as the "data dictionary" for regulatory reviewers, providing a comprehensive map of the submitted datasets. Without accurate and complete Define-XML, regulatory reviewers cannot efficiently navigate the complex data structures that underpin clinical trial submissions. As such, Define-XML is not optional -- it is a critical component of the electronic Common Technical Document (eCTD).

Background and Motivation

While the Define-XML standard is well-established, its operational implementation remains a significant bottleneck for biometrics departments. The traditional workflow often involves maintaining metadata in loosely structured, isolated Excel files, which are later manually transcribed or semi-automatically assembled using expensive commercial tools or ad-hoc SAS scripts.

In common ADaM programming practice, statistical programmers are each assigned ownership of specific analysis datasets, such as ADSL, ADAE, and ADLB. Each programmer maintains their dataset specification in a standalone Excel workbook. This distributed approach, while efficient for parallel development, poses several critical challenges when preparing for submission: cross-dataset conflicts,

predecessor traceability, validation delays, visual review bottleneck, etc. This disjointed process inevitably leads to metadata fragmentation: a scenario where the written specification, the physical dataset, and the final Define-XML document fall out of synchronization. The consequences are severe: submission delays, regulatory queries, and increased audit risk.

Define-XML v2.1 was published by CDISC to supersede Define-XML v2.0, introducing improvements in the representation of value-level metadata, analysis results metadata (ARM) for ADaM, and codelist provenance via the `def:Standards` element. Existing approaches to Define-XML production fall into three categories: manual authoring in XML editors, spreadsheet-to-XML utilities that require extensive manual mapping, and commercial tools that are often environment-specific and lack integration with analytical pipelines. An R-based approach is particularly attractive in pharmaceutical statistics teams where R is the primary programming language. A single, version-controlled Shiny application can serve as the canonical metadata production tool, eliminating environment dependencies and enabling scripted, reproducible execution in regulated contexts.

The Need for an Integrated Solution

To address these systemic inefficiencies, this paper introduces an R-based Shiny application designed to serve as a comprehensive "Metadata Hub". The application replaces document-centric, manual processes with a deterministic data pipeline. It is capable of autonomously harvesting metadata from scattered project folders, consolidating them into a unified specification, performing rigorous compliance checks, and instantly rendering submission-ready Define-XML artifacts—all within an interactive visual interface that facilitates real-time review and validation.

Unlike traditional command-line utilities or batch scripts, the Shiny-based approach provides immediate visual feedback, making it accessible to both technical programmers and non-programming stakeholders such as data managers, medical writers, and QC reviewers. The application serves not only as a document-generation engine but also as an interactive data visualization and reporting platform for metadata quality assessment.

APPLICATION ARCHITECTURE: A TWO-LAYER FRAMEWORK

The application is structured around three core functional layers, each implemented as a standalone R function that can be invoked either through the Shiny UI or programmatically in Rstudio:

Metadata Integration Layer (`create_adam_define_excel.R`)

XML Generation Layer (`create_define_xmlv21.R`) and Validation and Conformance Layer (`validate_metadata.R`)

This modular design ensures that each component can be tested, validated, and version-controlled independently, while still functioning seamlessly within the integrated Shiny application.

Display 1: Interface of R Shiny

LAYER 1: METADATA INTEGRATION (`CREATE_ADAM_DEFINE_EXCEL.R`)

FUNCTION OVERVIEW

The `create_adam_define_excel()` function is the heart of the metadata integration layer. This R function leverages modern tidyverse paradigms and functional programming principles.

Primary Responsibilities:

- Ingest multiple disparate Excel specification files

- Parse physical datasets (both XPT and SAS7BDAT formats) to extract actual variable attributes

- Map SDTM predecessors and inherit codelists from SDTM define spec if exists

- Generate a unified, Pinnacle 21-compatible ADaM define specification

MULTI-SOURCE METADATA RECONCILIATION

The function establishes a "Single Source of Truth" by extracting metadata from three distinct sources and reconciling conflicts using a hierarchical priority system:

Physical Datasets (Highest Priority): The `haven` package is used to read both XPT and SAS7BDAT files. Variable lengths, types, formats, and actual sequence orders are extracted directly from the data files. This ensures that the final Define-XML accurately reflects what reviewers will see in the submitted datasets.

Individual Specifications (Medium Priority): Each dataset's Excel specification (e.g., `adsl.xlsx`) provides derivation algorithms, comments, and value-level metadata. These are parsed using `readxl` and stored in memory.

CDISC Implementation Guides (Minium Priority): The function embeds references to CDISC ADaM IG, OCCDS IG, and standard controlled terminology. Variables are automatically cross-referenced against these libraries to populate standard attributes like core status and expected origins.

CRC32-BASED SMART CACHING

Clinical studies often contain dozens of datasets and thousands of variables. Re-processing all datasets when only one spec has changed is computationally wasteful. To optimize performance, the application implements a smart caching mechanism. It calculates a CRC32 hash (using the `digest` package) for

each source file (XPT and Excel spec). During subsequent runs, the application compares current hashes against a cached registry, selectively processing only the datasets that have been modified.

SDTM PREDECESSOR MAPPING

One of the most innovative features of the application is its ability to automatically map SDTM predecessors. If an SDTM Define-XML (or Excel representation) path is inferred or provided, the application parses it. When an ADaM variable lists an SDTM variable as its predecessor, the application automatically fetches the exact Codelist or Dictionary assignment from the SDTM metadata and applies it to the ADaM metadata. This eliminates manual cross-referencing and ensures unbroken end-to-end data traceability.

OUTPUT: PINNACLE 21 ENTERPRISE COMPATIBLE WORKBOOK

The output of this consolidation phase is a single, unified Excel workbook containing 11 standard metadata sheets (`Define`, `Datasets`, `Variables`, `ValueLevel`, `Codelists`, `Dictionaries`, `Methods`, `Comments`, `Documents`, `Analysis Displays`, and `Analysis Results`). This structured output is natively compatible with Pinnacle 21 Enterprise, acting as a perfect bridge between internal programming and external validation platforms.

LAYER 2: XML GENERATION AND VALIDATION AND CONFORMANCE (`CREATE_DEFINE_XMLV21.R` AND `VALIDATE_METADATA.R`)

Once the unified metadata workbook is generated, the application transitions to its XML Generation and Validation phase.

DETERMINISTIC XML CONSTRUCTION

The `create_define_xmlv21.R` function constructs the Define-XML tree programmatically. Utilizing the R `xml2` package, the script uses `xml_new_root()` to instantiate the root ODM element with all required namespaces (`odm`, `def`, `arm`, `xlink`). Child elements are appended deterministically. Because the tree is built natively in R's memory rather than through string concatenation, namespace handling and escaping are guaranteed to be syntactically perfect.

For ADaM submissions, the generator natively supports the Analysis Results Metadata (ARM) namespace, accurately translating the `Analysis Displays` and `Analysis Results` tabs into valid `arm:AnalysisResultDisplays` XML structures.

PDF RENDERING VIA XSLT

To produce a human-readable submission artifact, the application applies the official CDISC `define2-1-0.xsl` stylesheet using the R `xslt` package to generate an HTML view. This HTML is subsequently converted into a formatted PDF using headless rendering tools, providing the sponsor with a complete artifact package (Excel, XML, and PDF) simultaneously.

METADATA QUALITY CONTROL

The `validate_define_metadata()` function accepts the parsed sheet list and the detected standard, and returns a findings data.frame with columns `SEVERITY`, `SHEET`, `RULE`, and `MESSAGE`. Severity levels are ERROR (blocks generation), WARNING (flagged for review), and INFO (informational).

The function executes checks across all eleven sheets. For each sheet it verifies: required sheets are present; required columns exist; key field values conform to allowed controlled vocabularies; cross-sheet referential integrity is maintained (e.g., variable `CODELIST` values resolve to entries in the `Codelists` or `Dictionaries` sheets); and CDISC-specific format rules are enforced (dataset names uppercase alphanumeric, maximum 8 characters; variable names uppercase, maximum 8 characters; `DATA_TYPE` from the allowed Define-XML v2.1 set).

Standard-aware rules include: SDTM `ORIGIN` accepts

Collected/Derived/Assigned/Protocol/Predecessor; ADaM restricts to Derived/Assigned/Predecessor. SDTM dataset `CLASS` values are checked against SDTM domain classes; ADaM against ADaM dataset classes. ADaM `Analysis Results` sheets receive additional checks for `SELECTION_CRITERIA` syntax, `REASON` and `PURPOSE` controlled vocabulary, and cross-reference between `DISPLAY` references and `Analysis Displays` IDs.

Listing 1. Representative metadata validation rule (DATA_TYPE check)

```
valid_dtypes <- c('text', 'integer', 'float', 'datetime', 'date', 'time', 'partialDate',
'partialTime', 'partialDatetime', 'incompleteDatetime', 'durationDatetime')
bad_dt <- vars$DATA_TYPE[ !is.na(vars$DATA_TYPE) & !(vars$DATA_TYPE %in% valid_dtypes) ]
if (length(bad_dt) > 0) {
add('ERROR', 'Variables', 'VAR_DTYPE_INVALID', paste0('Invalid DATA_TYPE: ',
paste(unique(bad_dt), collapse=', ')))
}
```

CDISC CONFORMANCE CHECKING

After generation, `check_define_conformance()` parses the define.xml using `xml2::read_xml()`, registers CDISC namespace prefixes (odm, def, xlink, arm), and executes a rule catalogue covering over 80 named checks drawn from the published CDISC Define-XML v2.1 validation rules.

Rule families include: ODM-level structural rules (OD0001–OD0005: root element, required attributes, FileType, ODMVersion, def:Context); Study/MetaDataVersion integrity (ST0001–ST0004, MD0001–MD0004); def:Standards completeness (SD0001–SD0005, requires IG and CT entries); ItemGroupDef rules (IG0001–IG0006, DD0053–DD0063: Purpose, ArchiveLocationID, Class, child ordering, key variables, split dataset aliases); ItemDef rules (DD0058–DD0150: label completeness, length/datatype consistency, SignificantDigits for float, SASFieldName, origin types, predecessor syntax, CRF page references); OID uniqueness (OID0001–OID0002, OD0030–OD0032, DD0012–DD0014); cross-reference integrity (DD0015–DD0017, DD0071, DD0075, DD0078–DD0084: document, method, comment, valuelist, codelist referential consistency); and special variable rules (DD0105–DD0110: study day variables must be Derived, DOMAIN/RDOMAIN/STUDYID/VISITNUM/USUBJID origin consistency).

Findings are reported with rule ID, severity, affected element OID, and a human-readable message including the count of violations and sample affected elements. This enables both analyst-level triage and automated pipeline gating.

Listing 2. Conformance check — duplicate OID detection across all element types

```
all_oids <- c()

for (xpath in c('.//odm:ItemGroupDef', './/odm:ItemDef', './/odm:MethodDef',
'.//def:CommentDef', './/def:WhereClauseDef', './/def:ValueListDef', './/odm:CodeList'))
    { nodes <- xml2::xml_find_all(mdv, xpath, ns)
      oids  <- xml2::xml_attr(nodes, 'OID')
      all_oids <- c(all_oids, oids[!is.na(oids)])
    }
dup_oids <- all_oids[duplicated(all_oids)]
if (length(dup_oids) > 0) {
    add('ERROR', 'OID0001', 'OID',
    paste0('Duplicate OID(s): ', paste(unique(dup_oids), collapse=', ')))
}
```

THE ROLE OF INTERACTIVE VISUALIZATION

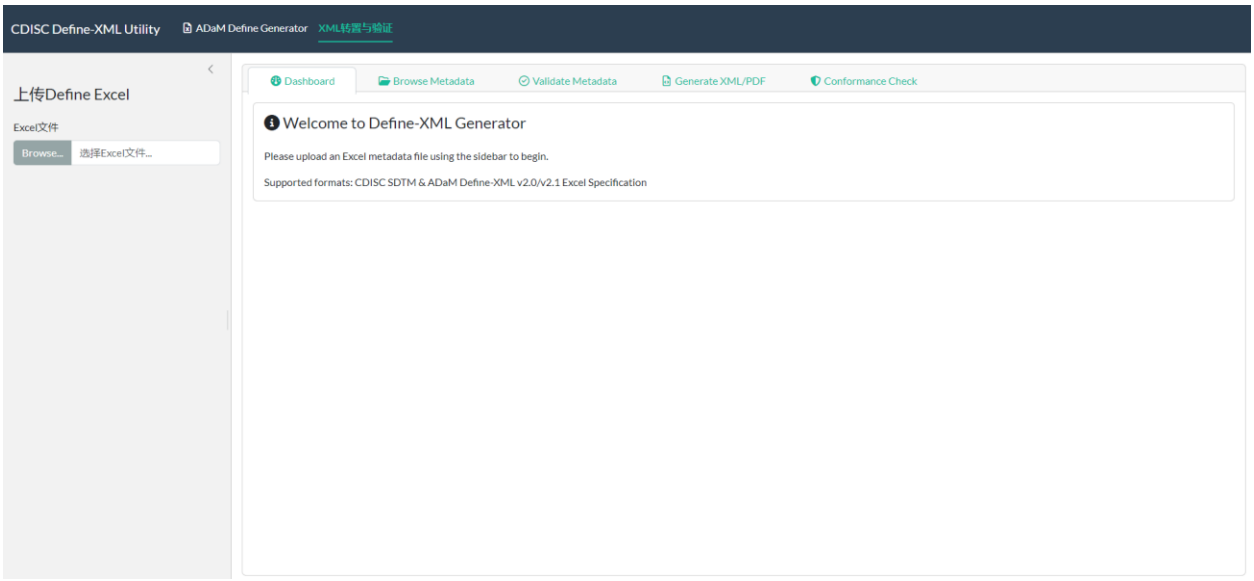
While the two core functional layers can be invoked programmatically via R scripts, the Shiny interface transforms these backend functions into an accessible, visual workflow. This section examines how interactive visualization addresses communication barriers inherent in traditional metadata production.

SHINY ARCHITECTURE: SERVER-SIDE REACTIVITY AND MODULAR UI

The application implements standard Shiny architecture with clear separation between `ui.R` (presentation layer) and `server.R` (business logic). The UI is structured as a `page_navbar()` with two primary navigation panels, each mapped to one of the three core functional layers:

Display 2: Interface of Define Genaration

ADaM Define Generator Tab: Wraps `create_adam_define_excel()` with file path inputs, collapsible parameter accordions, and real-time console output capture



Display 3: Interface of XML Genration/Validation

XML Generation & Validation Tab: Integrates `create_define_xmlv21()` and `validate_metadata()` with tabbed sub-panels for browsing, validating, and generating

PANEL 1: ADAM DEFINE SPECIFICATION GENERATOR

The first navigation panel provides a direct interface to the `create_adam_define_excel()` function described in Section 3. It features a sidebar-based parameter entry system with intelligent defaults:

Core Input: A file browser dialog allows users to select the ADaM specification excel file. Upon selection,

the application automatically infers related directory paths using the project folder convention: xptloc, adamloc, specloc, sdtmdefine. Besides, other parameters like protname, sdtmigv, adamigv, etc. were referenced automatically if not manually specified.

Execution & Feedback: Clicking the "Generate ADaM Define" button triggers `create_adam_define_excel()` in a reactive context. Console output is captured in real-time via `capture.output()` and displayed in a monospace `verbatimTextOutput` widget, providing live progress updates. Upon completion, a result summary displays key metadata counts (datasets, variables, codelists, methods, comments), and a download button becomes available.

Display 4: Interface of ADaM Define Specification Generator

This panel eliminates the need for command-line R scripting, making the metadata integration layer accessible to users without R programming experience.

PANEL 2: XML GENERATION AND VALIDATION WORKSPACE

The second navigation panel is structured as a multi-tab workspace containing five functional sub-panels. This panel accepts the Master Specification Excel file (either generated from Panel 1 or uploaded directly) and provides end-to-end Define-XML production capabilities.

TAB 1: DASHBOARD

Upon uploading file, the Dashboard tab presents an at-a-glance metadata overview using `bslib::value_box()` components:

- Row 1: Primary metrics (Datasets, Variables, ValueLevels)

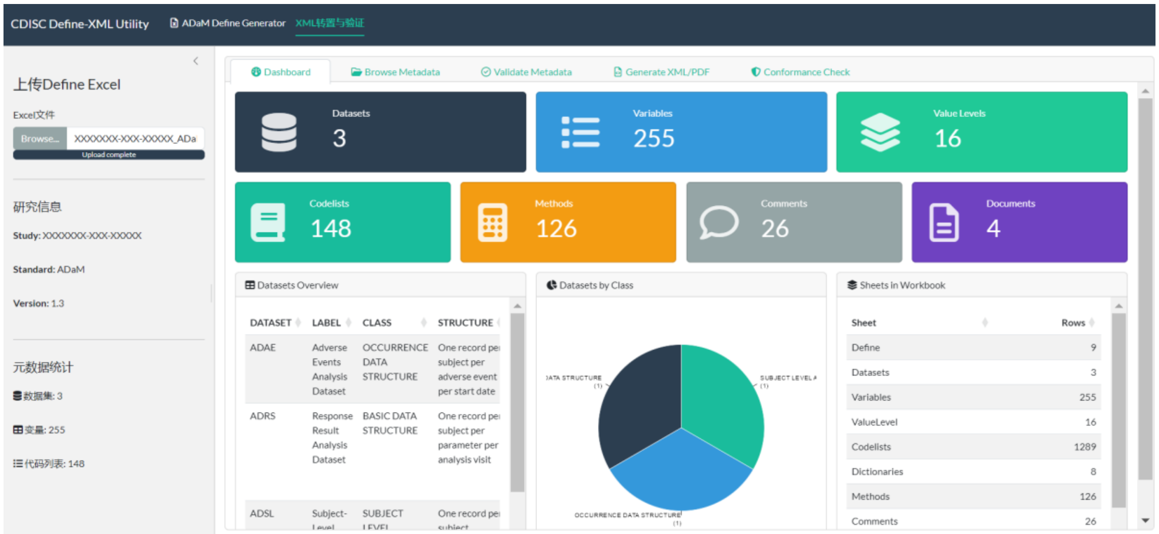
- Row 2: Supporting metadata (Codelists, Methods, Comments, Documents)

Below the value boxes, three interactive cards display:

Datasets Overview Table: Lists all datasets with their labels, classes, and variable counts

Datasets by Class Chart: A pie chart visualizing dataset distribution across CDISC classes (ADSL, BDS, OCCDS, OTHER)

Sheets in Workbook: Shows all Excel sheet names and row counts

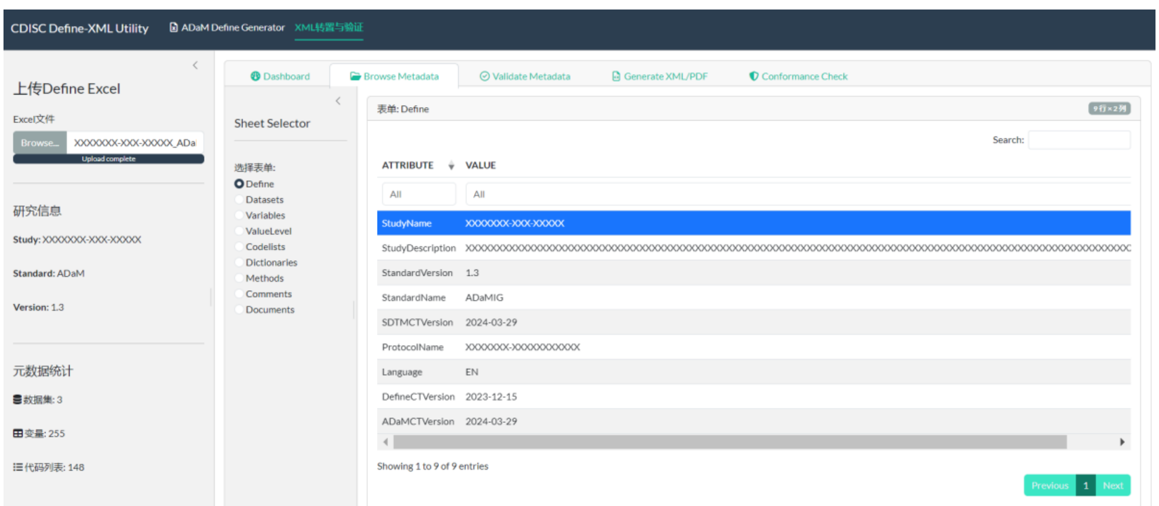


Display 5: Interface of Dashboard

The dashboard leverages `DT::datatable` for sortable, searchable tables and base R `pie()` charts rendered via `renderPlot()`. This tab provides quality control reviewers with immediate visibility into metadata completeness before XML generation.

TAB 2: BROWSE METADATA

This tab implements a sheet viewer with a radio-button sidebar for sheet selection. Users can inspect any worksheet from the uploaded Excel file in a filterable `DT::datatable` with column-level search. A dynamic badge displays current sheet dimensions (rows × columns). This tab is critical for QC reviewers who need to drill into specific metadata sections without opening Excel.



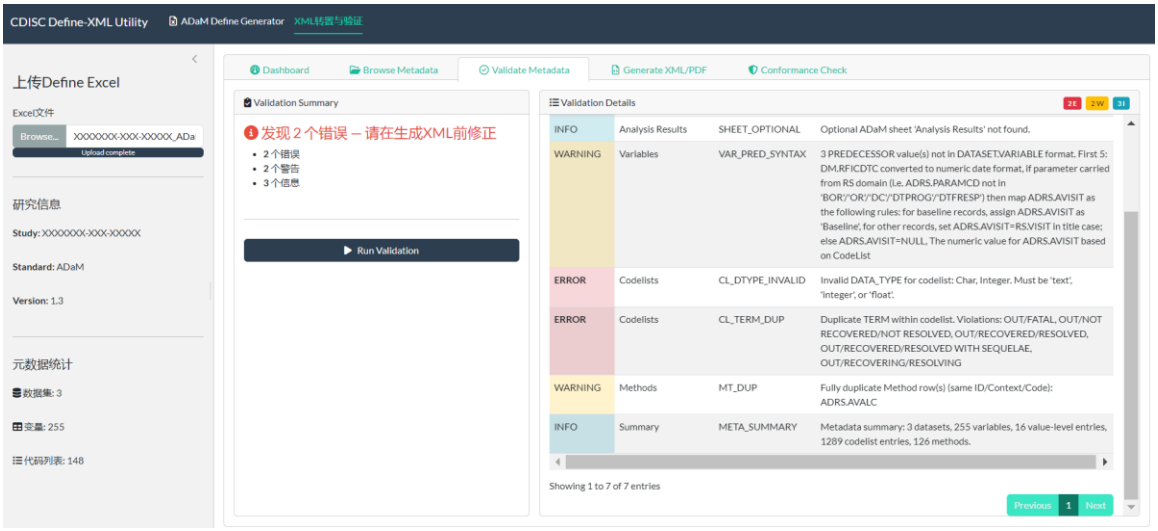
Display 6: Interface of Browse Metadata

TAB 3: VALIDATE METADATA

This tab surfaces the `validate_metadata()` function through a simple "Run Validation" button. Validation results are displayed in a two-panel layout:

- Left Panel: Summary card showing total error, warning, and info counts

- Right Panel: Full validation report rendered as a `DT::datatable` with severity-based color coding (errors in red, warnings in yellow, info in blue)



Display 7: Checking result of Validate Metadata

Validation runs automatically upon file upload, with the option to re-run manually. Results are downloadable as a CSV file for audit trail purposes.

TAB 4: GENERATE XML/PDF

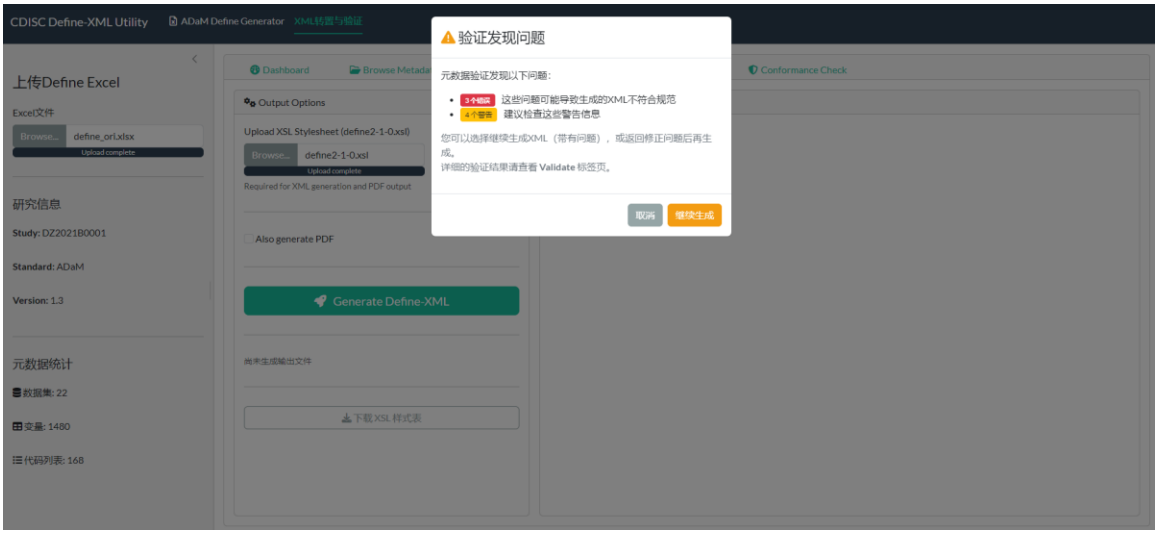
This tab orchestrates the `create_define_xmlv21()` function. Users upload the required XSL stylesheet (`define2-1-0.xsl`) and optionally enable PDF generation. A modal dialog intercepts generation requests when validation errors exist, prompting users to confirm before proceeding.

Upon clicking "Generate Define-XML", the application:

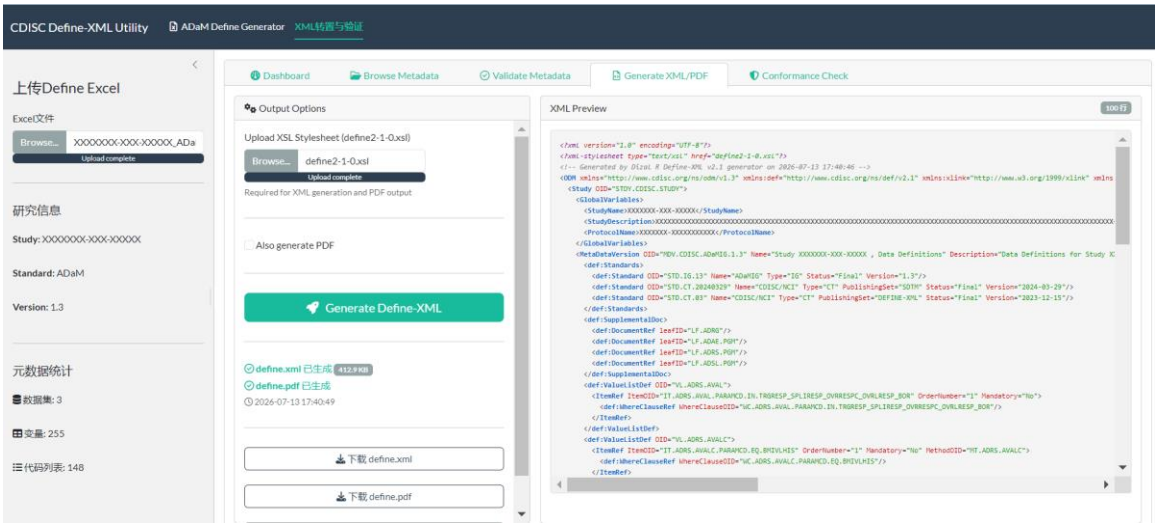
1. Calls `create_define_xmlv21()` with user-specified options

2. Renders an XML preview (first 500 lines) in a scrollable code block

3. Provides download buttons for both XML and PDF outputs



Display 8: Error handling prompts before XML generation

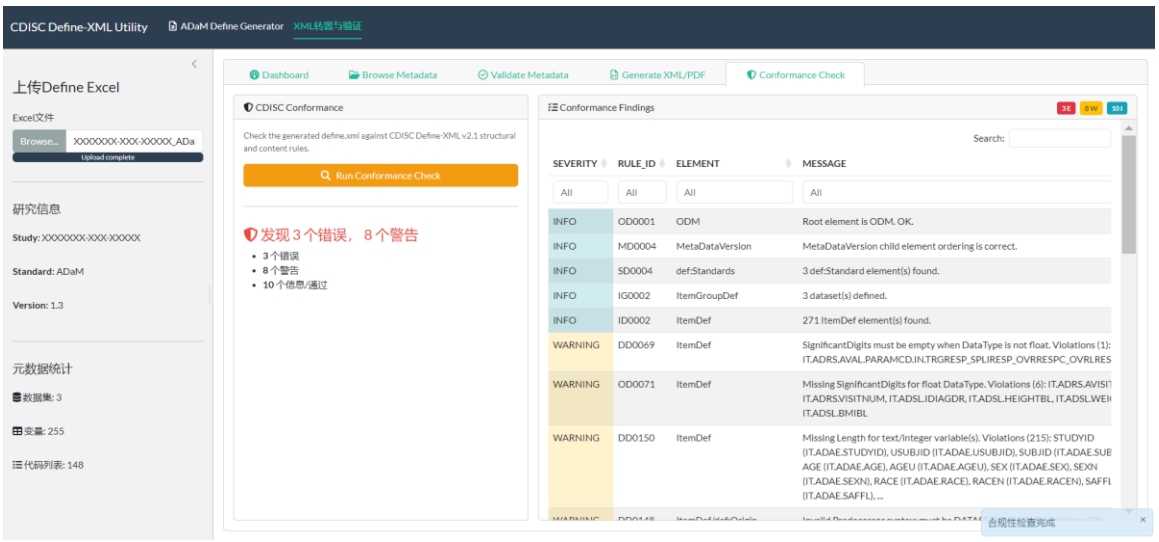


Display 9: XML generation result of Generate XML/PDF

The XML preview uses HTML tags with monospace styling, allowing users to visually inspect the generated structure before downloading.

TAB 5: CONFORMANCE CHECK

This tab runs post-generation CDISC conformance checks, validating the XML against Define-XML v2.1 structural and content rules. Results mirror the Validate Metadata tab layout, with a summary card and detailed findings table. This final quality gate ensures the XML meets submission standards before package assembly.



Display 10: Result of Conformance Check

TECHNICAL IMPLEMENTATION DETAILS

UI Framework: The application uses Bootstrap 5 via `bslib::bs_theme(bootswatch = "flatly")`, providing a modern, professional appearance with built-in responsive design.

Data Tables: All tabular displays use `DT::datatable()` with server-side processing disabled (`server = FALSE`) to enable client-side filtering and sorting without re-querying R.

File Handling: The `tcltk::tk_choose.files()` function provides cross-platform file browser dialogs, with automatic path normalization (`gsub("\\\\", "/", path)`) for Windows compatibility.

Download Handlers: `downloadHandler()` functions create on-demand file downloads without storing files in temporary directories, improving security and reducing disk I/O.

CSS Customization: A custom CSS block (`custom_css`) defined in `global.R` provides application-specific styling for badges, progress bars, and alert boxes.

The combination of reactive programming, Bootstrap components, and direct integration with the three core functions creates a cohesive user experience that masks the underlying complexity of Define-XML generation while maintaining full programmatic control.

CONCLUSION

This paper has presented a production-ready, R-based Shiny application for automated generation of ADaM Define Spec and Define-XML v2.1. By combining automatic SDTM/ADaM standard detection, deterministic XML construction via `xml2`, two-tier validation with over 100 named rules, and optional XSLT/PDF rendering, the application substantially improves upon manual and semi-automated approaches.

The contribution of this work is not only the application itself but the demonstration that R's ecosystem is mature enough to support the full Define-XML production lifecycle — from metadata ingestion to submission-ready artifact delivery — within a single, maintainable codebase. Teams can adopt this approach to reduce submission risk, improve metadata consistency, and accelerate regulatory review cycles.

For organizations seeking to modernize their Define-XML production processes, this open-source, R-based approach offers a transparent, cost-effective, and submission-proven alternative to commercial tools. The integration of interactive visualization and real-time validation exemplifies the power of the Data Visualization and Reporting paradigm in pharmaceutical data management. All source code is implemented in R and the application is deployable on any platform supporting R and Shiny, including enterprise Posit Connect environments.

REFERENCES

CDISC Define-XML Team. "CDISC Define-XML Specification Version 2.1 (Final)". July 6, 2021. https://www.cdisc.org/standards/data-exchange/define-xml/define-xml-v2-1

CDISC Define-XML Team. “Conformance Rules for Define-XML v2.1”. Mar 30, 2021. https://www.cdisc.org/standards/foundational/define-xml/conformance-rules-define-xml-v2-1

Define-XML-Validation Rules-Pinnacle 21. https://standards.pinnacle21.certara.net/validation-rules/Define-XML

ACKNOWLEDGMENTS

Thanks to all statistical programming team members at Dizal for their help to facilitate the success.

CONTACT INFORMATION

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

Name: Dongxu Liu
Enterprise: Dizal Pharma
E-mail: Dongxu.Liu@Dizalpharma.com