SESUG Paper ###-2025

Enhanced Reporting Traceability: An R Shiny Application for Live Dependency Tracking and SAS Code Execution

Laura Frederick, Tongda Che, and Peng Wan, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

In clinical trial analysis, managing complex dataset and code dependencies is crucial for ensuring traceability and compliance with regulatory guidelines. Traditional methods of tracking these dependencies often require significant manual effort, leading to potential errors and inefficiencies. This paper presents an innovative R Shiny application designed to streamline dependency tracking among SAS macro programs and output results.

The application provides a dynamic, real-time list of all TLF (table, listing, and figure) outputs associated with a given study protocol. Utilizing log files, it identifies and displays all relevant SDTM (Study Data Tabulation Model) datasets, ADaM (Analysis Data Model) datasets and macro programs linked to each output, thereby enhancing transparency throughout the development process. Additionally, the application uses file timestamps to automatically flag outdated files, significantly reducing the risk of errors and improving the accuracy of final outputs. With integrated SAS API (Application Programming Interface) functionality, users can effortlessly rerun any outdated TLF outputs and ADaM datasets with a single click.

Developed using base R, the Tidyverse, and the Shiny package, this application features a user-friendly interface for protocol path input and file selection, along with seamless integration with SAS. This tool enhances compliance with traceability requirements and optimizes clinical trial analysis efficiency.

INTRODUCTION

IMPORTANCE OF MANAGING DEPENDENCIES

Managing dependencies in clinical trial analysis is critical to ensuring data integrity, reproducibility, traceability, and regulatory compliance. Clinical trial analysis generates complex TLF (table, listing, and figure) outputs that often involve multiple levels of related programs and datasets. Figure 1 illustrates typical analysis and reporting relationships, showing the SDTM (Study Data Tabulation Model) and ADaM (Analysis Data Model) dataset and macro dependencies that are required to produce a TLF output.

Figure 1. Typical Analysis and Reporting Dependency Relationships

Further complexity can exist beyond the relationships shown in the figure. For example, the ADaM, TLF and Driver macros will often utilize additional secondary (utility) macros and ADaM datasets may rely on other ADaM datasets (most commonly ADSL). Due to these relationships, changes in one dataset or

program can directly impact the validity of outputs. Without effective dependency management, it becomes challenging to track these relationships, increasing the risk of errors, inconsistencies, and outdated outputs.

Regulatory agencies such as the FDA and the EMA require rigorous traceability of data and analysis workflows to ensure that clinical trial results are reliable and auditable. Proper dependency management supports this requirement by providing clear documentation of how outputs are generated, which datasets and programs contribute to each result, and when updates occur. This traceability is essential during audits and inspections, where sponsors must demonstrate that all outputs are reproducible and based on the most current data and validated code.

CHALLENGES WITH TRADITIONAL METHODS

Traditional methods for dependency tracking in clinical trial analysis often rely on manual documentation and spreadsheets. These approaches are time-consuming, error-prone, and increasingly difficult to maintain as trial complexity grows. Manual processes can lead to missed updates, outdated outputs, and an increased risk of non-compliance.

Automating dependency tracking and management addresses these challenges by providing real-time visibility into the relationships between datasets, macro programs, and outputs. It enables programmers to quickly identify outdated files, rerun necessary programs, and maintain consistency across all deliverables.

R AND SHINY FOR APPLICATION DEVELOPMENT

R is an open-source programming language designed for statistical analysis that offers extensive package libraries and tools for data analysis and visualization. Shiny is an R package that provides a powerful platform to build interactive web applications directly from R code. It allows developers to create dynamic, user-friendly interfaces without requiring extensive web development expertise. Shiny's reactive programming model automatically updates outputs in response to user inputs or changes in underlying data, making it particularly well-suited for applications that require real-time monitoring and interaction.

TECHNICAL DETAILS

DYNAMIC IDENTIFICATION OF DEPENDENCIES

This Shiny application takes a user-provided file path and identifies all TLF output file names within the folder location based on file extensions. It then uses the names of TLF outputs, along with standardized naming conventions and folder structures, to locate the corresponding log files for each output. Once the log files are located, the application parses them using regular expressions to extract key information, including the TLF output titles, all invoked macro programs, and all ADaM datasets used in the generation of each output. Once the ADaM datasets are identified by the application, their log files are further parsed to extract relevant SDTM datasets.

The application uses specific criteria to classify the macro programs into distinct categories: driver programs that initiate the processing, primary programs responsible for the main analysis, secondary programs that provide supplementary functions, and ADaM programs that generate the ADaM dependency datasets. This classification provides a clearer understanding of the role each program plays within the output generation workflow.

After identifying all dependencies, the application retrieves detailed file metadata for each one, including the last modification timestamp and the user ID associated with the most recent change. This metadata

along with the metadata for the TLF output is then used by the application to identify any dependency issues.

Example R code for dependency extraction:

```
# Read in log file
logfile <- paste0(file_path, log_name)
logfile_lines <- readLines(logfile)
logfile_txt <- paste(tolower(logfile_lines), collapse = " ")
# Function to extract ADaM dataset names from log text
extract_adam <- function(text) {
      # Regex pattern
      pattern <- "libname\\.([a-z]+)"
      # Find matches and extract matched substrings
      matches <- gregexpr(pattern, text, perl = TRUE)
      matched_strings <- unique(regmatches(text, matches)[[1]])
      # Remove the prefix to get only the desired text
      results <- sub("libname\\.", "", matched_strings)
      return(results)
}
adam_text <- extract_adam(text = logfile_txt)
```

FLAGGING DEPENDENCY ISSUES

To ensure outputs accurately reflect the most recent data and code changes, the application compares the timestamp of each TLF output with the timestamps of all its dependencies. If any dependency has a more recent timestamp than the output itself, the application automatically flags the outdated output. For example, if one of the ADaM datasets was rerun after the TLF output, the application highlights this discrepancy and identifies the ADaM dataset as the cause of the dependency issue. The application also checks if any of the ADaM datasets are outdated by comparing the dataset's timestamp against its primary macro and any SDTM/ADaM dataset dependencies. This automated flagging mechanism enables rapid identification of discrepancies, thereby reducing the risk of errors in the final deliverables.

Detailed list of dependency compliance checks:

- TLF Timestamp > Driver Macro Timestamp
- TLF Timestamp > Primary Macro Timestamp
- TLF Timestamp > All Secondary (utility) Macro Timestamps
- TLF Timestamp > All ADaM Dataset Timestamps
- TLF Timestamp > All ADaM Macro Timestamps
- ADaM Dataset Timestamp > ADaM Macro Timestamp
- ADaM Dataset Timestamp > ADSL Timestamp (when ADaM Dataset is not ADSL)
- ADaM Dataset Timestamp > All SDTM Dataset Timestamps

R code for comparing TLF output Timestamp against dependencies and create issue variable:

```
# Dependency_data is a tibble with one row per dependency per TLF
dependency_issues <- dependency_data |>
  mutate(issue = case_when(
   dep = "adam" & dep_tm > tlf_tm ~ "ADaM Timestamp > Output Timestamp",
   dep = "drvr" & dep_tm > tlf_tm ~ "Driver Timestamp > Output Timestamp",
   # repeat for other checks
 )
```

Note that tibbles are used throughout the application code. The tibble package is part of the Tidyverse suite. A tibble is an enhanced version of a traditional R data frame, designed to make data manipulation and analysis more intuitive and user-friendly than traditional data frames.

APPLICATION ARCHITECTURE

The application follows a modular framework inspired by the golem package and structures the Shiny application as a fully functional R package. This design allows for modular development, simplifies testing of individual components, promotes code reuse, and streamlines application deployment. Organizing the codebase in this manner enhances maintainability and scalability, facilitating the addition of new features.

The core functions for dependency identification and issue flagging were created using a combination of base R and the Tidyverse suite. The Tidyverse suite provides powerful tools for efficient data manipulation and analysis of log file contents and file metadata. The Shiny package was used to deliver an interactive and responsive user interface, while the DT package enables dynamic tables that can be sorted and filtered to allow intuitive exploration of dependencies and flagged issues.

The application also leverages the future.apply package to enable parallel processing of multiple TLF outputs concurrently. This parallelization distributes the workload across available CPU cores, significantly accelerating parsing of log files for dependency extraction. This results in faster response times when displaying the dependency table, enhancing the overall interactivity of the application.

To enable seamless execution of SAS code, the application integrates an API (Application Programming Interface) that allows SAS programs to be rerun directly from within the Shiny application. This eliminates the need for manual code execution or switching between different tools. By combining R and SAS technologies, the application creates a unified platform that enhances workflow efficiency, traceability, and regulatory compliance.

SHINY APPLICATION INTERFACE

OVERVIEW

The Shiny application provides a clean, intuitive interface designed to facilitate efficient tracking and management of dependencies. The interface contains two tabs at the top: Home and Dependency Tracking. The Home tab includes a dropdown for navigating between the Welcome page, a Change Log page (containing the application's change history), and a Configuration page (containing application configuration details). Figure 2 shows the Welcome page, which serves as the landing page when users first access the application.

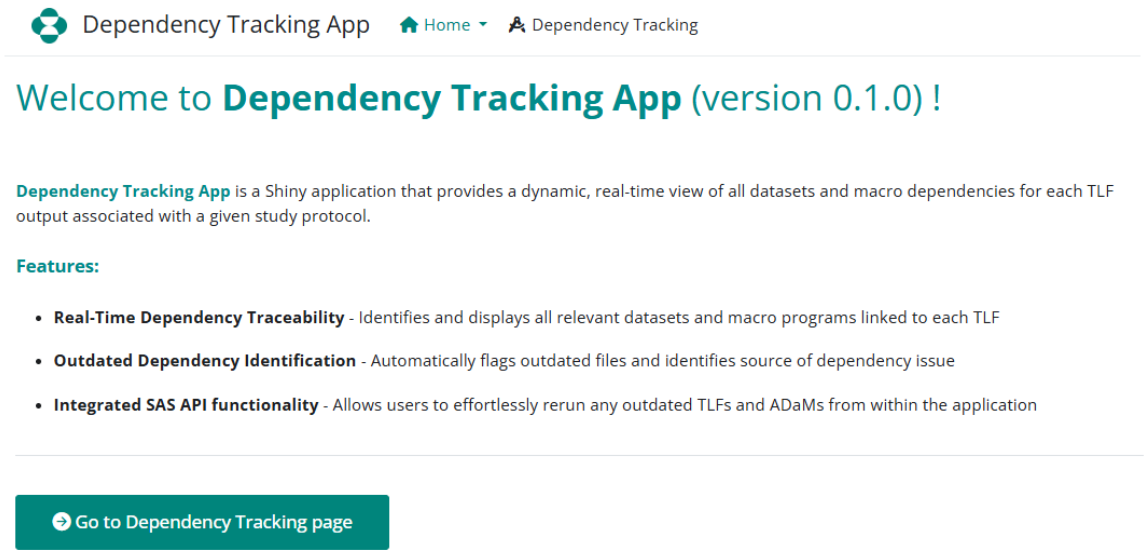

Figure 2. Welcome Page for Dependency Tracking Application

The second tab of the application is Dependency Tracking. This tab contains the primary page of the application. The Dependency Tracking page is organized into three main panels: Inputs, Dependency Table, and Dependency Details Table.

INPUTS

Users begin by defining the protocol path through a series of dropdown menus that specify the environment, compound/indication, and protocol deliverable. This structured input ensures the application targets the correct folder hierarchy. The application automatically identifies all TLF outputs in the specified protocol location. Users can then select which outputs to check for dependencies by choosing either all available outputs or specific output files from a searchable list. Users can also filter the outputs displayed to show either all selected outputs or only those flagged with dependency issues. Once selections are made, clicking the “Display Dependencies” button loads relevant dependency data into the main table. Figure 3 shows the Inputs panel.

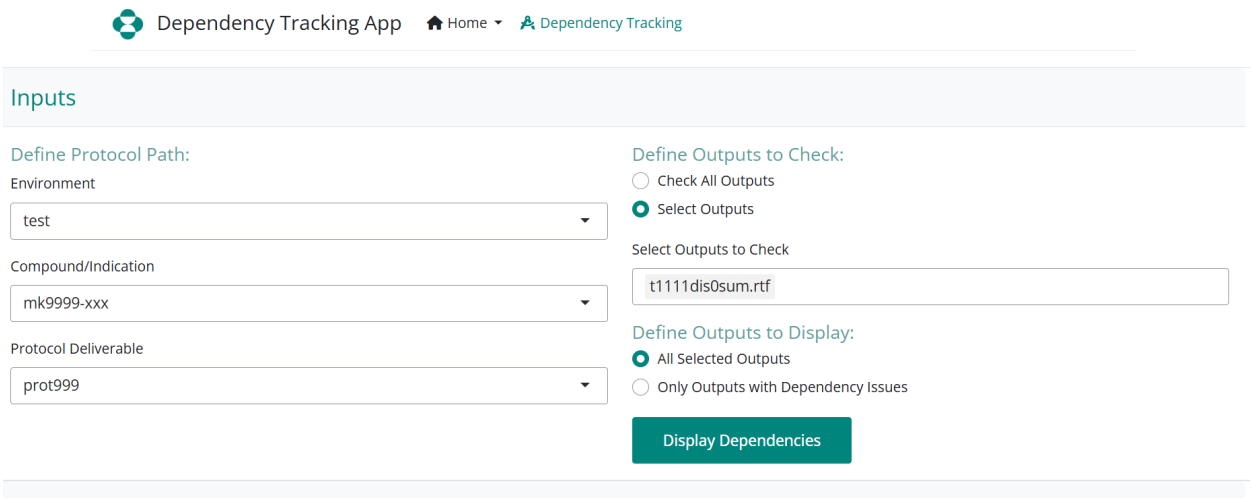

Figure 3. Inputs Panel for Protocol Path and Output Selection

DEPENDENCY TABLE

Below the input panel, the Dependency Table panel displays a table with one row per selected TLF output, providing a comprehensive summary of each output's dependencies and dependency issue status. For outputs with dependency issues, the table uses dynamic color-coded highlighting to draw attention to the dependencies responsible for the issue. By default, the table is sorted to display outputs with issues first. Users can sort and filter on any column to focus on outputs of interest. Buttons below the table allow exporting the dependency table to Excel and rerunning selected driver / ADaM programs. The table's interactive design supports efficient navigation and prioritization of outputs requiring attention. Figure 4 shows the Dependency Table.

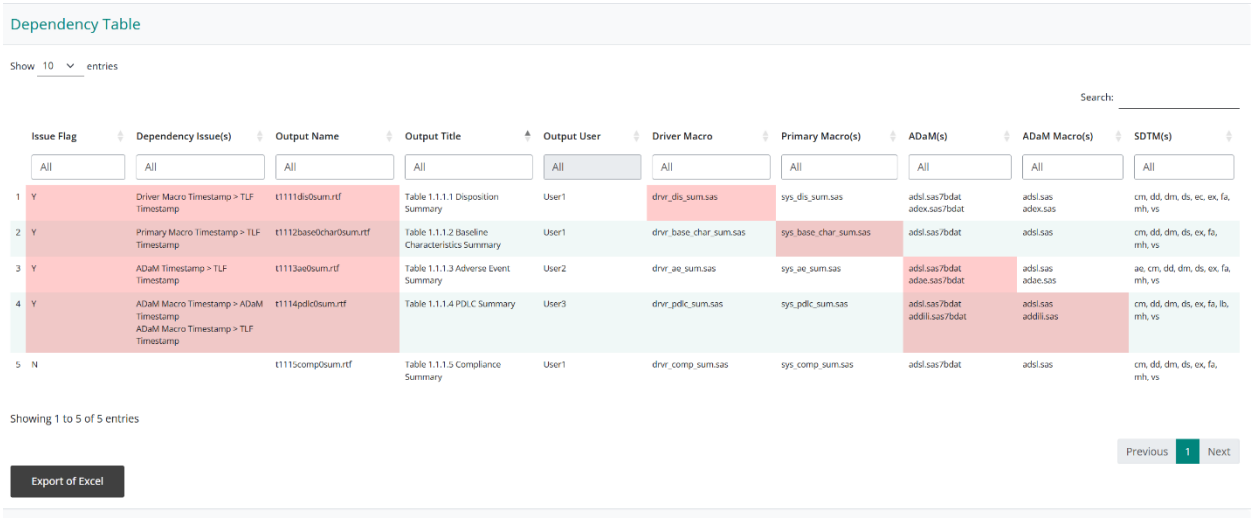

Figure 4. Dependency Table Displaying TLF Outputs and Their Issue Status

DEPENDENCY DETAILS TABLE

This panel dynamically contains a second table that automatically updates based on the rows selected in the main dependency table, showing detailed information for each dependency associated with the selected outputs. Each row represents a single dependency—such as driver programs, primary or secondary macro programs, or SDTM / ADaM datasets—and includes metadata like timestamps and modification user IDs. The Dependency Details Table supports the same interactive features as the main table, including sorting, filtering, dynamic highlighting, and Excel export. This detailed view enables users to drill down to the specific dependencies affecting output status. Figure 5 shows the Dependency Details Table panel.

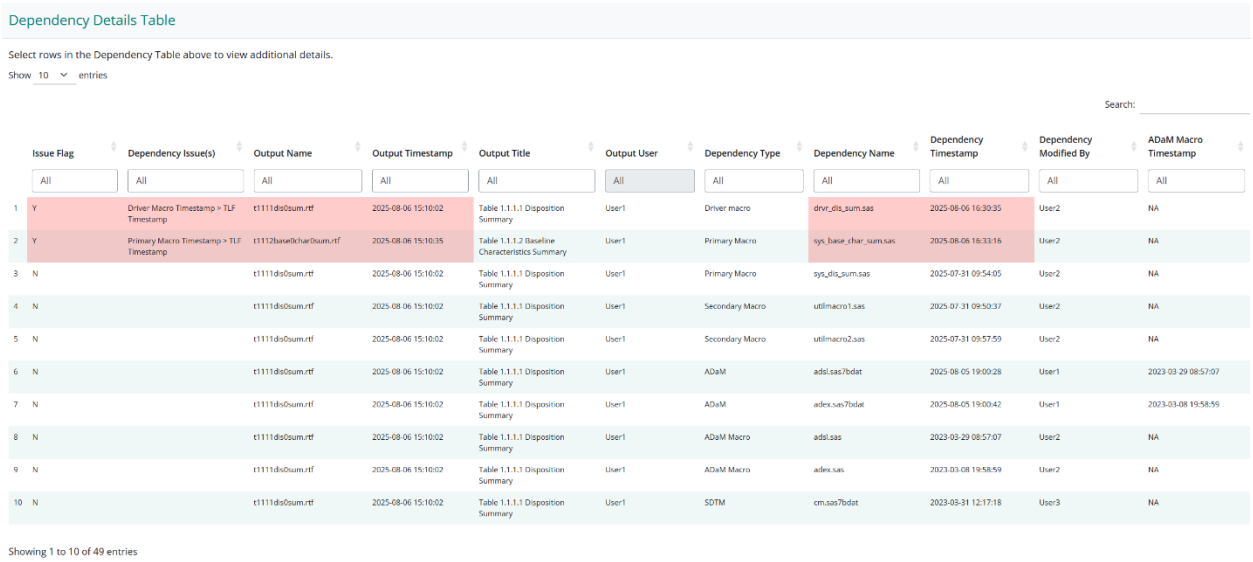

Figure 5. Dependency Details Table Showing Individual Dependencies for Selected Outputs

SAS CODE EXECUTION

The application integrates SAS code execution, allowing users to rerun outdated driver programs and ADaM programs directly from the interface. When users select rows within the dependency table, two buttons become available: “Run Selected Driver Macros” and “Run Selected ADaM Macros.” Clicking one of these buttons triggers a confirmation window listing the programs to be executed, thereby preventing accidental execution. After confirmation, a secondary window prompts users to enter their SAS login credentials to execute the code. While the selected programs are running, a progress bar displays the running status, providing real-time feedback until completion. After rerunning the selected outputs, users can refresh the dependency table to ensure dependency issues have been resolved. This seamless integration reduces manual effort in switching between programming applications and ensures outputs are up to date. Figure 6 shows the two program execution buttons that appear below the dependency table once rows are selected. Figure 7 shows the window displayed prior to ADaM or driver program execution. Figure 8 shows the progress bar displayed during ADaM or driver program execution.

Figure 6. Program Execution Buttons

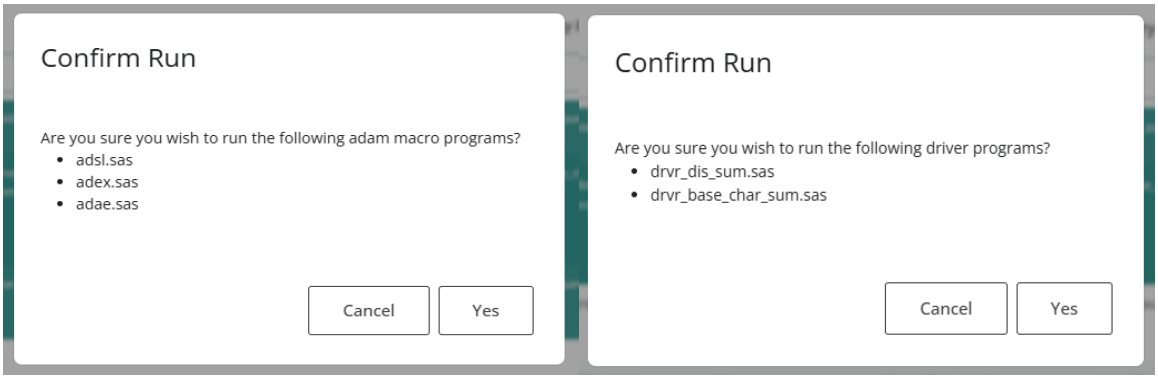

Figure 7. Program Execution Confirmation Windows

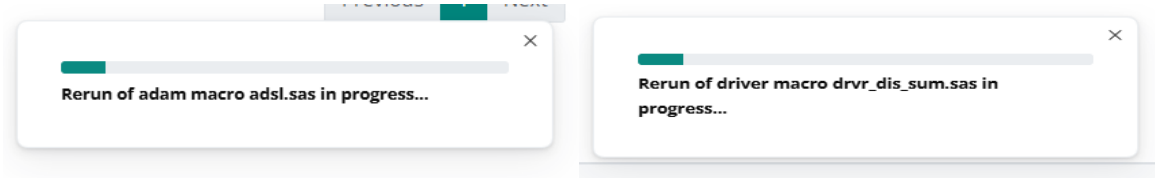

Figure 8. Program Execution Progress Bars

EXAMPLE USE CASES

DELIVERABLE TRACEABILITY AND COMPLIANCE

During the analysis and reporting process, the application can be used for real-time tracking of all TLF outputs generated for a specific study protocol, providing users with immediate visibility into the current status and completeness of deliverables. By consolidating output metadata and dependency information into a centralized dashboard, the tool enables stakeholders to monitor progress dynamically and identify any missing or incomplete outputs at any stage of the reporting process.

The built-in export to Excel functionality allows users to extract details of output status and dependencies. These exports can be used by programmers to update any static analysis and reporting tracking spreadsheets, thereby improving the accuracy and efficiency of manual record-keeping. This capability supports compliance with regulatory requirements by ensuring that traceability documentation is both comprehensive and up to date, reducing the risk of discrepancies during audits or inspections. Additionally, the application empowers programmers to proactively identify and resolve dependency issues throughout the TLF development process. By flagging outdated outputs based on automated timestamp comparisons, the tool helps to ensure that all deliverables reflect the most current validated data and code. This continuous monitoring capability reinforces adherence to Good Clinical Practice (GCP) and regulatory guidelines.

IDENTIFY OUTPUTS IMPACTED BY UPDATES

This application enables programmers to immediately identify which TLF outputs are affected by updates to underlying programs or datasets. This targeted identification is particularly valuable in minimizing unnecessary reruns. For instance, following a database lock (DBL), if an update is required for an ADaM macro program, the application can precisely identify all TLF outputs that depend on the corresponding ADaM dataset. Users can then rerun the impacted outputs from within the application – promoting efficiency. This selective rerun capability ensures that only impacted outputs are regenerated, preserving the timestamps of unaffected deliverables.

FUTURE IMPROVEMENTS

This application is currently a proof-of-concept and will require validation before it is deployed for study use. In addition, opportunities exist to enhance the application's functionality.

Examples of potential future enhancements include:

- Additional dependency checks for ADaM datasets
    - ADaM Dataset Timestamp > Dependent Secondary Macro Timestamps
    - ADaM Dataset Timestamp > Dependent ADaM Timestamps (not just ADSL)
- Creation of an ADaM-level dependency details table
- Extending code execution capabilities to support programs written in languages beyond SAS (e.g., R and Python)

CONCLUSION

This paper presents a novel R Shiny application that significantly advances dependency management among TLF outputs, macro programs, and ADaM datasets in clinical trial analysis. By automating the identification and tracking of these dependencies through log file parsing and metadata comparison, the application enhances traceability, reduces manual effort, and mitigates the risk of errors associated with outdated or inconsistent outputs. The integration of SAS code execution within the Shiny interface further

streamlines workflows, enabling users to promptly update outputs and maintain compliance with regulatory requirements. Developed with a modular architecture, it offers a scalable and user-friendly solution that can be expanded to further enhance traceability and efficiency. Overall, this application optimizes the clinical data analysis processes ensuring data integrity and supporting regulatory compliance through improved dependency management.

REFERENCES

Wickham, Hadley. 2021. *Mastering Shiny*. 1st ed. O'Reilly Media, Inc. Available at https://mastering-shiny.org/.

University of California Berkeley, Department of Statistics. "Parallel Processing using the future package in R." Accessed August 7, 2025. Available at https://computing.stat.berkeley.edu/tutorial-dask-future/R-future.html.

ACKNOWLEDGMENTS

The authors would like to thank our colleagues Hui Liu and Danfeng Fu for their support in the development of this Shiny application. We also thank our managers and the Merck late-stage statistical programming leadership team for their valuable feedback during the preparation of this paper.

CONTACT INFORMATION

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

Laura Frederick
Merck & Co., Inc., Rahway, NJ, USA
laura.frederick1@merck.com

Tonda Che
Merck & Co., Inc., Rahway, NJ, USA
tong.da.che@merck.com

Peng Wan
Merck & Co., Inc., Rahway, NJ, USA
peng.wan@merck.com