PharmaSUG 2026 - Paper ET285

mkheader: An R Package for Automated Generation and Management of Program Headers in Clinical Trial Programming

Laura Frederick, Merck & Co., Inc., Rahway, NJ, USA
Gabriela Piasecki, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

In clinical trial programming, maintaining accurate and up-to-date program headers is essential for regulatory compliance, traceability, and audit readiness. Program headers capture critical metadata such as study name, input datasets, authorship, and revision history. As R gains traction in pharmaceutical programming, the lack of standardized tools for automated header generation within R leads to inefficient, error-prone, manual processes that increase compliance risks.

The mkheader R package addresses these challenges by automating the creation and management of program headers tailored specifically for clinical trial programming. It automatically extracts metadata - including study, author details, program date, and R version - and seamlessly integrates user inputs to generate consistent, comprehensive headers. Key features include interactive header creation and maintenance via RStudio Addins, automated header updates, and seamless integration with existing R programming workflows.

By eliminating manual header maintenance, mkheader enhances programming efficiency and ensures headers remain accurate, consistent, and audit-ready throughout the study lifecycle. This package offers a practical solution to improve documentation quality and regulatory compliance in clinical trial programming.

INTRODUCTION

Clinical trial programming involves complex, multi-contributor workflows, and stringent regulatory scrutiny. Program headers serve as the primary in-file documentation, articulating context and evidence needed for audits: study identifiers, program inputs and output, environment details, and change history. In practice, headers are often inconsistent, incomplete, or out of date, hindering traceability and increasing compliance risk.[1] In the regulatory context, clear audit trails and documentation are essential for submissions and the FDA recommends all programs include standard headers as part of good programing practice.[2,3]

The mkheader package is a comprehensive R solution designed to automate the creation, maintenance, and management of standardized program headers for clinical trial programming within R. Developed specifically to address the unique documentation requirements of pharmaceutical programming environments, mkheader transforms header management from a manual, time-consuming task into an automated, efficient process that ensures consistency and regulatory compliance.

DESIGN PRINCIPLES

The mkheader package is built on three fundamental principles. First, automation with intelligence: it minimizes manual data entry by pulling information from the code, file system, and existing headers, and by applying sensible defaults aligned to organizational conventions. Second, compliance by design: it structures headers to meet regulatory expectations through mandatory fields, standardized formatting, and complete revision histories. Third, seamless integration: it is integrated directly into the RStudio environment through addins, so programmers can create and maintain headers without interrupting their coding workflow.

PACKAGE ARCHITECTURE

The mkheader package consists of three core functions integrated into RStudio as addins, support functions, and defined header templates.

Core Functions:

- addin_create_header(): Creates a new header for the active script
- addin_update_header(): Updates the header for the active script
- addin_update_program_flow(): Refreshes the Program Flow section based on current code comments

mkheader integrates these functions seamlessly with RStudio through three registered addins: “Create New Header”, “Update Existing Header”, and “Update Program Flow", accessible from the Addins menu as shown in **Figure 1.**

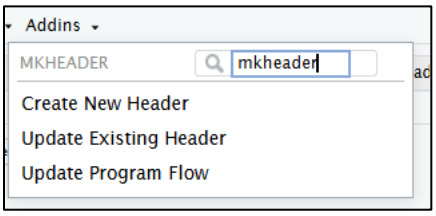

Figure 1. mkheader Addins

Key Support Functions Include:

- .apply_header(): Inserts new header to the beginning of active program
- .check_header(): Determines if content has existing header
- .parse_header(): Extracts information from existing header
- .shiny_header_dialog(): Displays a Shiny-based interface that captures user inputs
- .validate_and_prepare_document(): Identifies active program and retrieves content
- create_header(): Formats inputs into a standard header based on defined template

The folder structure for the mkheader package is shown in **Figure 2**. The package’s functions are logically grouped into R scripts (e.g., header_operations.R, addins.R) by their purpose, making the package contents easy to navigate and maintain. Header templates are defined in templates.R. These templates define the structure and formatting of program headers. This template system is designed to be extensible, allowing for customized header formats while maintaining the core functionality of the package.

```
mkheader/
├── R/
│   ├── addins.R                    # Core Addin Functions
│   ├── constants.R                 # Package Wide Constants
│   ├── create_header.R             # Header Creation Function
│   ├── create_header_interactive.R # Interactive Dialog Logic
│   ├── header_operations.R         # Header Parsing/Manipulation Functions
│   ├── rstudio_helpers.R           # RStudio Integration Utilities
│   ├── shiny_header_dialog.R       # Shiny Gadget Definition
│   ├── program_flow.R              # Program Flow Helper Functions
│   ├── templates.R                 # Header Template Definitions
│   └── utils.R                     # Helper Functions
├── inst/
│   └── rstudio/
│       └── addins.dcf      # Addin Registration
└── man/                    # Documentation
└── tests/                  # Package Testing
```

Figure 2. mkheader R Package Structure

KEY FUNCTIONALITY

STANDARDIZED HEADER CREATION

The primary function of mkheader is to create comprehensive program headers with minimal user input. For new R scripts, mkheader's "Create New Header" addin populates core fields automatically. When a user selects this addin, the associated core function first identifies the active program and retrieves its contents. It extracts the program name from the filename, identifies the study and protocol from directory conventions, captures the current user and date, and records the system environment (R version and platform). It also generates default output and log path locations based on naming standards.

A Shiny gadget-based window then displays all input fields with the extracted information automatically populated as shown in **Figure 3**. For entry fields without defaults, example values and additional instructions guide users on how to populate. Users then review the pre-filled information and fill in fields such as the program's purpose, inputs, keywords, and known limitations. The interface clearly marks required fields with an asterisk.

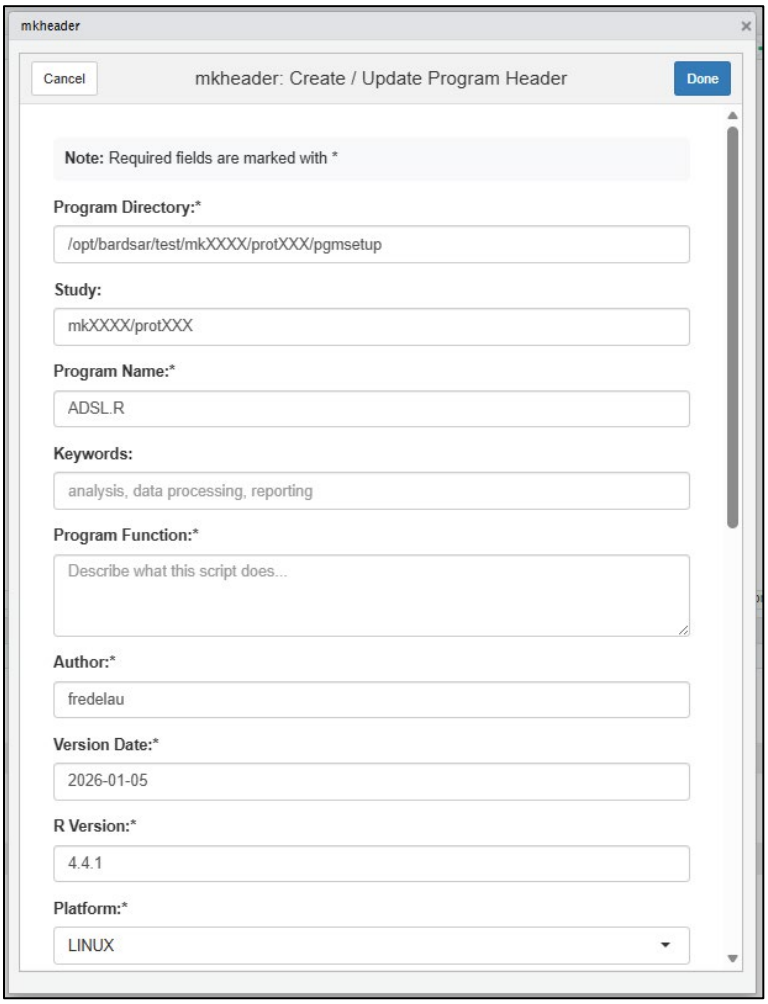

Figure 3. "Create New Header" Addin Window with Pre-Filled Defaults

Once all required fields have been populated, users can select ‘Done’ to insert their formatted header into the active program. **Figure 4** shows an example of the resulting standardized header added to an R script using “Create New Header”.

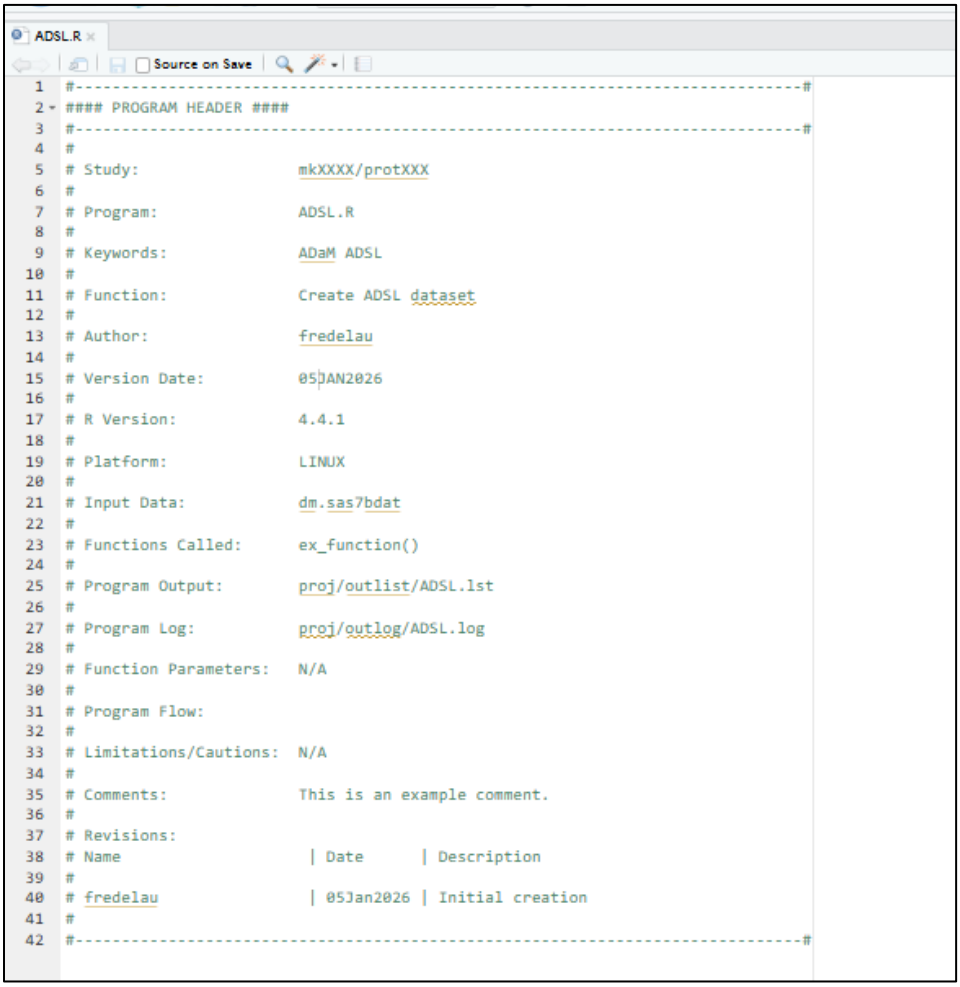

Figure 4. New Header Inserted into R script with “Create New Header” Addin

INTELLIGENT HEADER UPDATES

A critical challenge in clinical trial programming is maintaining header accuracy as programs evolve. The mkheader package addresses this through its "Update Existing Header" functionality. This function parses the current header, extracting all current field values and loading them into the interactive dialog. Users can then adjust any of the existing fields that need to be updated. The package preserves the full revision history in a readable format and replaces the header without touching any code below it. This approach ensures header formats remain standardized during updates, preventing formatting drifts from manual updates.

Regulatory processes require robust traceability. mkheader enforces revision tracking from the start: the initial header includes an “Initial creation” entry with author and date. As shown in **Figure 5**, “Update Existing Header” automatically prompts for a new revision entry (with sensible defaults for user and date) ensuring the revision history is kept up to date. This promotes traceability and ensures documentation stays aligned throughout the program’s lifecycle. An example of the updated header revision history is shown in **Figure 6**.

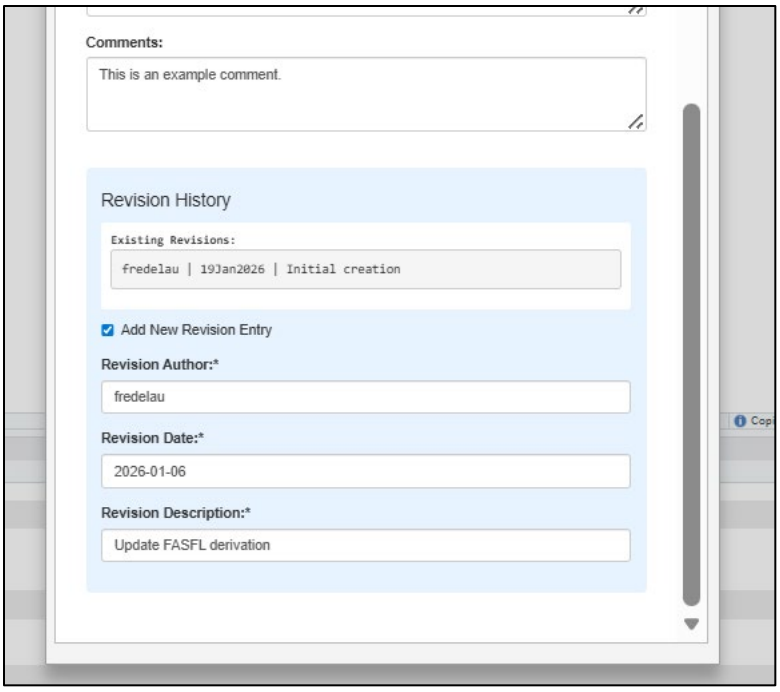

Figure 5. Revision History Section for "Update Existing Header" Addin

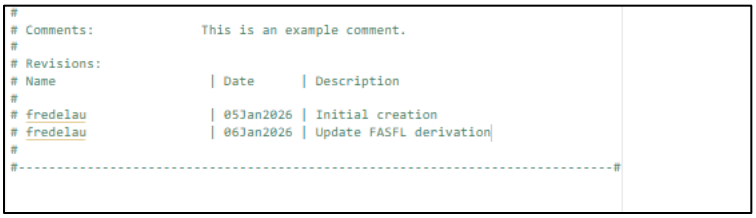

Figure 6. Updated Revision History using "Update Existing Header" Addin

AUTOMATED PROGRAM FLOW DOCUMENTATION

One of the most innovative features of mkheader is its ability to automatically generate and maintain program flow documentation. Programmers typically document their code with step comments (for example, "# Step 1: Load Packages"). The Program Flow section of the header contains a list of all steps performed throughout the program. Manually maintaining this section of the header is tedious and error-prone, and outdated program flow sections can undermine trust in documentation. mkheader automatically updates this section from structured step comments in the script. The "Update Program Flow" addin scans the code, ignores the header block, extracts step numbers and descriptions and replaces the Program Flow section with an accurately formatted, list of steps. This feature ensures that high-level program documentation remains synchronized with the actual code structure, eliminating the frequent problem of outdated program flow descriptions. **Figure 7** and **Figure 8** show an example of code with documented steps and the resulting updated Program Flow section of the header after running "Update Program Flow".

```
45  # Step 1: Load Packages
46  library(dplyr)
47  library(pharmaversesdtm)
48
49  # Step 2: Read in Data
50  dm <- pharmaversesdtm::dm
51
52  # Step 3: Derive Treatment Variables (TRT0xP, TRT0xA)
53  adsl <- dm %>%
54    mutate(TRT01P = ARM, TRT01A = ACTARM)
```

Figure 7. Example Program Code with Steps Documented

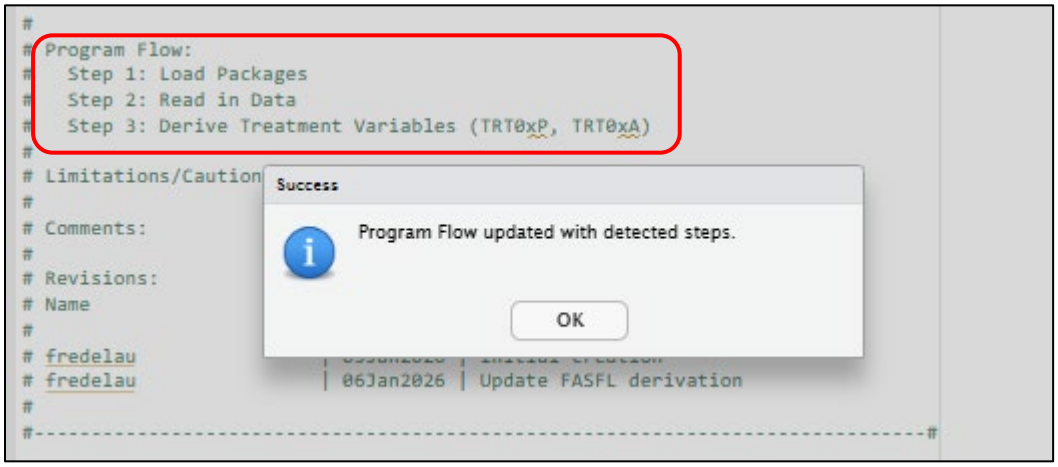

Figure 8. "Update Program Flow" Success Popup with Updated Header in Background

VALIDATION AND SAFETY CHECKS

The mkheader package implements a comprehensive validation framework to ensure data integrity and prevent errors throughout the header creation and maintenance process. Before any operation begins, the package performs essential pre-flight checks: verifying that RStudio is running, confirming an active R script is open, and checking whether the active program has been saved. If a program is unsaved, the user is prompted to save it before proceeding. When creating a new header, the package scans the script to detect existing headers; if one is found, the user is prompted to either update the existing header or cancel the operation, preventing accidental header duplication. Similarly, the "Update Existing Header" addin verifies that a header exists before proceeding, offering to create a new header if none is found. The "Update Program Flow" addin performs dual validation by first confirming a header exists and then scanning the code for step comments; if no steps are found, the user receives a clear message explaining how to add step comments to their code, and the program remains unchanged. Examples of two user prompts are shown in **Figure 9**.

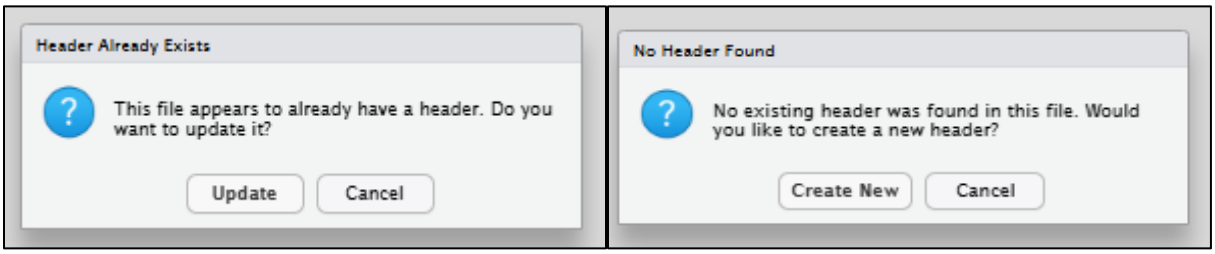

Figure 9. 'Header Already Exists' and 'No Header Found' Dialog Boxes

Field-level validation ensures completeness and accuracy of header metadata. Required fields—including Program Directory, Program Name, Program Function, Author, Version Date, and R Version—are validated upon form submission, with all fields checked for non-empty values after trimming whitespace. Date fields undergo additional validation to ensure they are in valid format and not set to future dates. When updating an existing header, if the user opts to add a revision entry (enabled by default), the revision author, date, and description become required fields with the same validation rules applied. As shown in **Figure 10**, if any validation errors are detected, a dialog displays all errors in a clear, bulleted list, and the user remains in the form to make corrections. The package follows a fail-safe philosophy: no changes are made to the program until all validations pass, users can cancel operations at any point without consequence, and all successful operations are logged in the revision history to maintain a complete audit trail. This multi-layered validation approach ensures that headers are always complete, accurate, and compliant with documentation standards while providing clear guidance to users when issues are detected.

Figure 10. Example 'Validation Error' Dialog Box

FUTURE IMPROVEMENTS

This package is currently undergoing additional validation and testing before it is published. In addition, opportunities exist to enhance the application's functionality.

Examples of potential future enhancements include:

- Additional functionality to auto-detect inputs/outputs and dependencies
- A separate 'Update Revision History' function / addin
- Addition of more standard program header templates
- More robust step comment extraction for program flow
- Support for SAS and Python headers

CONCLUSION

The mkheader package represents a significant advancement in clinical trial programming documentation practices. By automating header creation and maintenance while ensuring consistency and regulatory compliance, it addresses critical pain points in pharmaceutical programming workflows. The package's intelligent automation, robust revision tracking, and seamless RStudio integration make it a practical solution for improving documentation quality and efficiency in clinical trial programming environments.

REFERENCES

[1.] Liping Sun. May 2023. "Experiences of CDER Statistical Analysts with NDA/BLA Reviews: Some Helpful Tips for Sponsors". PharmaSUG 2023 Conference Proceedings: FDA-G004. https://pharmasug.org/proceedings/2023/FDA/PharmaSUG-2023-FDA-G004.pdf

[2.] Good Programming Practice Project Team. September 2021. "Programming Practice – Headers". PHUSE Advance Hub. https://advance.hub.phuse.global/wiki/spaces/WEL/pages/26805922/Good+Programming+Practice+Headers

[3.] Paul Schuette and Weiya Zhang. March 2018. "FDA Study Data Technical Conformance Guide Updates for Software Programs Submission". Poster at 2018 PhUSE Computational Science Symposium. https://www.lexjansen.com/css-us/2018/PP33.pdf

[4.] Posit. January 2026. "RStudio Addins" RStudio User Guide Release 2026.01.0. https://docs.posit.co/ide/user/ide/guide/productivity/add-ins.html

CONTACT INFORMATION

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

Laura Frederick
Senior Scientist, Statistical Programming
Merck & Co., Inc., Rahway, NJ, USA
E-mail: laura.frederick1@merck.com

Gabriela Piasecki
Senior Scientist, Statistical Programming
Merck & Co., Inc., Rahway, NJ, USA
E-mail: l.gabriela.piasecki@merck.com

DISCLAIMER

The views and opinions presented in this paper are our own and do not necessarily reflect the official position of the company.