Paper SM09

Batch Execution of CDISC SDTM/ADaM Programs with R

Denny Sebastian, Catalyst Flex, Kochi, India
Mohin K Mohanan, Catalyst Flex, Kochi, India

ABSTRACT

This abstract describes an R-based implementation that replicates the functionality of SAS batch processing for SDTM/ADaM domains. In clinical data programming and regulatory submissions, SDTM/ADaM datasets are often generated and validated in batch mode using SAS scripts. This paper demonstrates how R can be used to achieve similar SAS batch processing of SDTM domains using R scripts, parameterized functions, and dynamic execution. This functionality executes domain-specific scripts, and logs results for traceability. It supports scalability, automation, and reproducibility, key features for clinical trial programming. This functionality also generates a comprehensive error report by analyzing the log files produced during the execution of SDTM/ADaM domain programs using R batch mode. This solution enables clinical programmers to transition to R workflows without losing the robustness and familiarity of traditional SAS batch operations for SDTM dataset generation.

INTRODUCTION

The clinical programming field is evolving, and R has become increasingly prominent due to its open-source ecosystem, rapid innovation, and strong support for reproducible research practices. Many teams are exploring R-based alternatives for clinical data transformation, validation, and reporting. In the context of SDTM/ADaM, R is increasingly being used on the validation side to independently replicate and verify datasets developed in SAS. In most organizations, SAS batch processing remains the established approach for generating SDTM and ADaM domains: programs run in a predefined sequence, parameters are consistently controlled, and log files provide audit-ready evidence to support QC, traceability, and troubleshooting. As teams transition to R, they often require the same production-grade batch capability, not merely rewritten domain logic.

This paper presents an R-based framework that mirrors the batch-processing functionality traditionally achieved using SAS for SDTM/ADaM generation. The proposed approach dynamically executes domain-specific scripts, captures structured logs for traceability, and produces a consolidated error report through automated log analysis, supporting scalable, automated, and reproducible clinical programming workflows.

BACKGROUND AND RATIONALE

In SAS, a batch mode of execution refers to executing all programs for generating SDTM/ADaM domains in a specific, predefined order. These programs are initially developed and validated to ensure accuracy. Batch execution automates the process, enabling consistent, efficient transformation of raw clinical data into CDISC-compliant SDTM/ADaM datasets. R programming is emerging as a powerful tool in clinical data analytics, especially in the validation of clinical trial datasets. In the context of SDTM/ADaM, R is increasingly being used on the validation side to independently replicate and verify datasets developed in SAS. R can be used to achieve similar SAS batch processing of SDTM domains using R scripts, parameterized functions, and dynamic execution. R can be used to generate a comprehensive error report by analyzing the log files produced during the execution of SDTM/ADaM domain programs using R batch mode.

Beyond replication and validation, R offers advanced capabilities that enhance batch execution workflows. It supports parallel processing and distributed computing, significantly reducing execution time for large-scale datasets. Integration with version control systems such as Git ensures traceability and reproducibility of all batch runs. R's flexibility allows customization of validation rules beyond standard CDISC compliance, enabling tailored quality checks.

WHY R

R stands out as an ideal choice for clinical data validation and automation due to its versatility, open-source nature, and strong ecosystem of packages for data science. Its extensive libraries for data manipulation, statistical analysis, and visualization enable comprehensive validation of SDTM and ADaM datasets with minimal overhead. R's integration with modern automation frameworks and reporting tools ensures seamless execution of batch processes and dynamic error tracking. Furthermore, its growing adoption in regulatory environments underscores its capability to meet compliance requirements while supporting innovation in clinical analytics. By leveraging R, organizations can achieve a balance of efficiency, flexibility, and reproducibility, making it a strategic complement to traditional SAS-based workflows. R also facilitates integration with cloud platforms like AWS, Azure, and GCP, enabling scalable and secure execution of batch processes. Additionally, its interoperability with Python and other languages supports hybrid workflows, offering maximum flexibility for organizations adopting multi-tool strategies.

R AUTOMATION CAPABILITIES

R offers a wide range of automation capabilities that make it a powerful tool for data processing, analytics, and reporting in modern workflows. Its scripting flexibility allows repetitive tasks such as data cleaning, transformation, model execution, and visualization to be executed reliably without manual intervention. The integration of packages like *purrr*, *dplyr*, and *data.table* enables efficient batch processing over multiple datasets. For scheduled or event-driven workflows, R can be seamlessly integrated with cron jobs, task schedulers, or workflow orchestration tools such as Airflow, Nextflow, and GitHub Actions. R Markdown and Quarto extend automation to dynamic reporting, generating HTML, PDF, or Word outputs directly from scripts with embedded code. Additionally, R's ability to interact with APIs, databases, and external systems allows it to automate data ingestion and publishing processes. In regulated environments, automation frameworks can incorporate structured logging, error handling, and version control to ensure auditability and traceability. Automated validation dashboards can be built using Shiny, providing real-time monitoring of batch execution and error logs. Furthermore, R supports automated email notifications and alerts for process completion or error detection, improving operational efficiency and reducing manual oversight.

FRAMEWORK DESIGN

The primary objective of the framework is to execute a set of SDTM R programs in batch mode without manual intervention. The design ensures that:

- Programs run in a predefined sequence
- Execution is independent and reproducible across runs
- Program specific console logs are created automatically
- Execution time is recorded for monitoring and documentation

This Framework adopts a method-development design to operationalize a reproducible batch execution framework for SDTM program runs in R. The batch runner is implemented in base R operating system invocation via *shell*(). This approach ensures that each SDTM program is executed in an independent R session, this isolation reduces the risk of carryover effects (e.g., objects in memory, altered options, or loaded packages) influencing subsequent domain runs.

IMPLEMENTATION APPROACH

The framework is implemented using base R and Windows command execution via *shell*(). The batch execution uses R CMD BATCH, which runs an R script in a non-interactive session and writes a transcript of console activity to an output file. This pattern enables automated SDTM execution without requiring an active R console session. Then the ordered execution can be done by a predefined list of SDTM programs (e.g., domain scripts) and the batch run function executed multiple times over this list using a for loop, ensuring that programs run in the required order.

In R, *shell*() is a system interface function that lets you execute an external operating system command from within an R session. Conceptually, it is a bridge from R to the command interpreter allowing to run scripts and command line utilities as if type them in a terminal.

What happens when you call *shell*(cmd, ...)

- R passes a command string (cmd) to the operating system.
- The OS launches a command interpreter (typically Windows cmd.exe)
- The interpreter executes the command, which may start one or more external processes, here R.exe.
- When the command completes, the control returns to the R session.
- R captures the exit status (return code) and optionally the console output, depending on the arguments used.

PROGRAM WORKFLOW

A predefined list of SDTM program files is maintained to enforce a deterministic execution order. The list typically contains program names (e.g., domain scripts) and their corresponding program directory. The full path to each program is constructed using *file.path*() to ensure platform consistent path handling.

For each program:

- Path Resolution: The program's absolute path is constructed from the inputs.
- Command Construction: A batch mode command is composed by referencing the resolved R executable path and the script path.
- Independent Invocation: The command is executed using *shell*(), launching a new R process for the script.
- Console Redirection: Console output is redirected to a dedicated log file (.Rout file) corresponding to the executed program.
- Runtime Capture: Execution time is measured by recording timestamps immediately before and after invocation.
- Program specific console log documenting execution messages, warnings, and errors

CODE & RESULTS

`rbatch_run` is a custom R function developed to enable automated batch execution of pre-defined R programs in a controlled sequence. It runs each script via R CMD BATCH, generates the corresponding console logs, and records execution time.

```
#' Execute SDTM R script in batch mode
#'
#' @param file name of an R script to be executed
#' @param progdir file directory of the R script to be executed
#'
#' @return Invisibly returns the path to the log file.
#'
```

```
#' @example rbatch_run(File = "dm.R", progdir = 'validation/program')
#'
rbatch_run <- function(File = "dm.R", progdir = 'validation/program'){

  # Check input
  Rinst <- file.path(R.home("bin"), "R")
  destdir <- progdir
  file <- file.path(progdir, File)

  domain <- gsub('.R$', '', basename(file))
  logfile <- paste0(domain, ".Rout")
  logfile <- normalizePath(file.path(destdir, logfile))
  infile <- normalizePath(file)

  cat("Processing sdtm script:", basename(file), "...!")
  execution_time <- system.time(
    shell(paste0(normalizePath(Rinst, mustWork = FALSE),
                 'CMD BATCH ', "--vanilla", ' "', infile,
                 '" "', logfile, '"'), wait = TRUE))

  run_time <- paste(round(execution_time[['elapsed']], 3), 'seconds')
  cat("\rExecution time for the domain `", basename(file), "` : ", run_time,
"\n", sep = "")

  if (any(grepl("^Warning", readLines(logfile))))
    warning(sprintf("%s Output file may contain warnings... check!", domain))

  if (any(grepl("^Error", readLines(logfile))))
    warning(sprintf("%s Output file may contain error... check!", domain))

  invisible(logfile)
}
```

IMPLEMENTAION SCRIPTS

The following script serves as the implementation scripts for the batch-run workflow using the `*rbatch_run()*` function with the required inputs (program paths, execution order, and output locations). It primarily sets up the program execution order in a dynamic way. Once the execution order is finalized, `*rbatch_run()*` is invoked within a *for* loop function to run each program in the predefined sequence. Overall, the script standardizes end-to-end execution by ensuring consistent log generation and timing capture across all scripts, thereby supporting transparent and reproducible processing.

```
# ================ Get list of programs ============================
dir.prod.prog <- 'validation/program'
all.progms <- list.files(dir.prod.prog, pattern = '.R$')
trials <- c("ta.R", "te.R", "ti.R", "ts.R", "tv.R")
primary <- c("dm.R", "sv.R", "se.R")
last <- c('co.R', 'relrec.R')
other <- setdiff(list.files(dir.prod.prog, pattern = '.R$'),
                 c(trials, primary, last))
```

```
# ================ Get batch run list ============================
run.List <- c(trials, primary, other, last)
run.List <- intersect(run.List, all.progms)
run.List <- intersect(run.List, paste0(tolower(specDomains), '.R'))
cat("\n## ==== Check Domain List Before Execution ==== ##\n",
    paste0(sprintf('%s. %s', sprintf("%02d", 1:length(run.List)), run.List),
collapse = ' \n '))
```

```
> cat("\n## ==== Check Domain List Before Execution ==== ##\n",
+     paste0(sprintf('%s. %s', sprintf("%02d", 1:length(run.List)), run.List),  collapse = ' \n '))

## ==== Check Domain List Before Execution ==== ##
 01. ta.R
 02. te.R
 03. ti.R
 04. ts.R
 05. tv.R
 06. dm.R
 07. sv.R
 08. se.R
 09. ae.R
 10. cm.R
 11. cv.R
 12. da.R
 13. dd.R
 14. ds.R
 15. ec.R
 16. eg.R
 17. ex.R
 18. ie.R
 19. lb.R
 20. mb.R
 21. mh.R
 22. pc.R
 23. pr.R
 24. rp.R
 25. rs.R
 26. ss.R
 27. vs.R
```

```
# ================ Execute ============================
for (i in run.List){
  rbatch_run(i, progdir = dir.prod.prog)
}
```

```
> # ==== Execute ====
> for (i in run.List){
+   rbatch_run(i, progdir = dir.prod.prog)
+ }
Execution time for the domain `ta.R` : 7.66 seconds
Execution time for the domain `te.R` : 2.96 seconds
Execution time for the domain `ti.R` : 2.84 seconds
Execution time for the domain `ts.R` : 2.95 seconds
Execution time for the domain `tv.R` : 2.74 seconds
Execution time for the domain `dm.R` : 6.28 seconds
Execution time for the domain `sv.R` : 4.54 seconds
Execution time for the domain `se.R` : 2.85 seconds
Execution time for the domain `ae.R` : 4.85 seconds
Execution time for the domain `cm.R` : 4.38 seconds
Execution time for the domain `cv.R` : 4.09 seconds
Processing sdtm script: da.R ...!
```

The pictures show snapshots of the console output. As the results indicate, we receive a status message such as ***"Processing SDTM script: domain…!"*** when the program is running. Once the execution of a domain is completed, the console displays the corresponding execution time in seconds, which confirms that the domain has finished running. If any program contains errors or warnings, the function returns a message like "***The program 'domain.R' contains errors/warnings; please check the log file."*** The above script demonstrates the batch execution of SDTM domains, and we can achieve similar batch execution for ADaM domains as well.

R V/S SAS

Running programs in R batch (R runs scripts automatically, creates log files) is like SAS batch (SAS runs programs automatically, creates SAS log and output). Both are used for scheduled, repeatable production runs.

Benefits of R batch

- Low cost: R is usually free
- Flexible: R has many packages for data handling, statistics, graphs, and machine learning.
- Easy automation: It works well with modern tools like Git, pipelines, and cloud servers.
- Reproducibility options: Using the *renv* package we can lock package versions so the same code gives the same results later.

Benefits of SAS batch

- Strong industry acceptance: SAS is widely used in clinical and regulated environments.
- Standard logs and procedures: SAS output and log formats are consistent and familiar to reviewers.
- Stable enterprise setup: Many companies already have validated SAS systems and established SOPs, so running batch jobs is straightforward.

LIMITATIONS

- Package dependencies can change: R programs depend on many packages. If a package is updated (or removed), the same script may behave differently or fail unless versions are locked.
- Environment differences matter: Results can change between machines due to different R versions, package versions, OS differences (Windows vs Linux), and system libraries. This is a common issue in batch runs deployed across servers.
- Harder environment control if not governed: Without controls like renv, containers (Docker), or a managed server image, it is easy for "it works on my machine" problems to occur.
- Log standardization is not automatic: R logs can be produced (e.g., .Rout), but the content and format are not as standardized as SAS logs unless you enforce templates and conventions.
- Regulated expectations may require extra documentation: In clinical/regulatory settings, reviewers often expect strong traceability, validated processes, and consistent outputs. With R, you may need additional SOPs, validation evidence, package/version documentation, and audit trails to meet these expectations.
- Dependency on external system tools: Batch execution may rely on OS commands (shell()/system()), file permissions, network paths, and scheduler settings each can cause failures outside the R code itself.

R batch is powerful, but it needs stronger governance (version locking, controlled environments, standardized logs, and documentation) to be reliable in regulated production use.

FUTURE WORK

Parallel execution: A practical direction for future work is to extend the current sequential batch framework to support parallel execution for faster end-to-end SDTM processing. At present, programs are executed one after another to preserve a predefined order; however, many SDTM domain scripts are independent or only partially dependent, meaning they can be grouped and executed concurrently without compromising data integrity. Future enhancement can introduce a dependency aware scheduler that first classifies programs into (i) prerequisite domains, (ii) independent domains, and (iii) downstream domains that require upstream outputs. Within each independent group, multiple R sessions can be launched in parallel while still enforcing "wait points" between dependency groups. In addition, resource management (CPU, memory) must be handled to avoid performance issues when too many jobs run simultaneously. This enhancement is expected to reduce overall runtime, improve scalability for large studies, and enable more efficient integration with enterprise schedulers and cloud-based execution environments.

CONCLUSION

This article presented an R based batch execution framework for SDTM/ADaM program runs that supports non-interactive, repeatable processing in a controlled sequence. By invoking R CMD BATCH through *shell*() function each SDTM/ADaM script is executed in an independent R session, producing a corresponding console log file and enabling execution time measurement for operational monitoring. This approach improves traceability, reduces session carryover risk, and fits scheduled or command line environments. While environment governance and regulated documentation require additional controls, the framework provides a practical, scalable foundation for automated SDTM/ADaM pipeline execution.

Beyond the operational advantages demonstrated in this framework, the growing adoption of R in the clinical domain underscores its strategic importance in modern statistical computing. R offers an extensive ecosystem of validated packages for clinical trial data manipulation, statistical modeling, visualization, and regulatory-compliant reporting, making it highly adaptable to diverse study designs and therapeutic areas. Its open-source nature fosters transparency, reproducibility, and continuous community-driven innovation. In addition, modern tools within the R ecosystem such as *R Markdown, Shiny, Quarto, Git*, and *Posit Connect* provide powerful capabilities for automated reporting, interactive dashboards, version-controlled collaboration, and scalable deployment. Together, these technologies make R not only a robust analytical environment but also a comprehensive platform for end-to-end clinical data processing and reporting. As regulatory agencies increasingly support submissions built using open-source technologies, the role of R is set to expand further, enabling more efficient analytics pipelines, advanced data science methodologies, and scalable automation across the clinical development lifecycle. This positions R as a critical technology for future-ready clinical analytics and evidence generation.

REFERENCES

https://www.rdocumentation.org/packages/base/versions/3.3.3/topics/shell
https://adv-r.hadley.nz/

ACKNOWLEDGMENTS

We thank our colleagues, mentors, and reviewers for their guidance and constructive feedback throughout the development of this paper. Additionally, we acknowledge the assistance of AI tools, such as Copilot, in organizing ideas and refining content, which contributed to a more efficient writing process.

CONTACT INFORMATION

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

Author Name: Denny Sebastian
Company: Flex & Oncology Clinical Research Private Limited
Address: 8B Tower 1, TransAsia, Cyber Park Infopark, Phase 2, Kochi, Kerala 682303
Work Phone: 9539693707
Email: denny.sebastian@catalystcr.com
Website: https://catalystcr.com/

Co-Author Name: Mohin K Mohanan
Company: Flex & Oncology Clinical Research Private Limited
Address: 8B Tower 1, TransAsia, Cyber Park Infopark, Phase 2, Kochi, Kerala 682303
Work Phone: 9539499771
Email: mohin.mohanan@catalystcr.com
Website: https://catalystcr.com/