

SM09

Batch Execution of CDISC SDTM/ADaM Programs with R

Denny Sebastian

Catalyst Flex

denny.sebastian@catalystcr.com

Mohin Mohanan

Catalyst Flex

mohin.mohanan@catalystcr.com

PHUSE APAC CONNECT

February 2026



Introduction

- R is rising in clinical programming due to open source, innovation, and reproducibility
- Teams use R for data transformation, validation, and SDTM/ADaM verification
- SAS is still the standard for SDTM/ADaM batch processing and audit-ready logs
- Transitioning to R requires SAS-like production batch capabilities
- This paper provides an R framework for SAS-style batch execution, logs, and error reporting



Background/Rationale/Motivation

- SAS batch mode runs SDTM/ADaM programs in a predefined sequence for consistent, automated dataset generation
- R is increasingly used to validate SDTM/ADaM datasets by independently replicating SAS-based outputs
- R can replicate SAS-style batch processing using scripts, functions, and dynamic execution
- R can analyze log files to produce consolidated error reports during batch execution
- R supports advanced capabilities like parallel processing, Git integration, and customizable validation rules to enhance clinical data workflows

Why Choose R

- **Versatility and Open Source**

R offers a versatile open-source platform with extensive packages for clinical data validation and automation

- **Comprehensive Data Validation**

R supports detailed validation of SDTM and ADaM datasets using libraries for data manipulation and statistical analysis

- **Automation and Compliance**

Integration with automation frameworks enabling smooth batch execution and dynamic error tracking

- **Cloud and Hybrid Integration**

R integrates with cloud platforms and interoperates with other languages for scalable and flexible clinical data workflows.





R Automation Capabilities

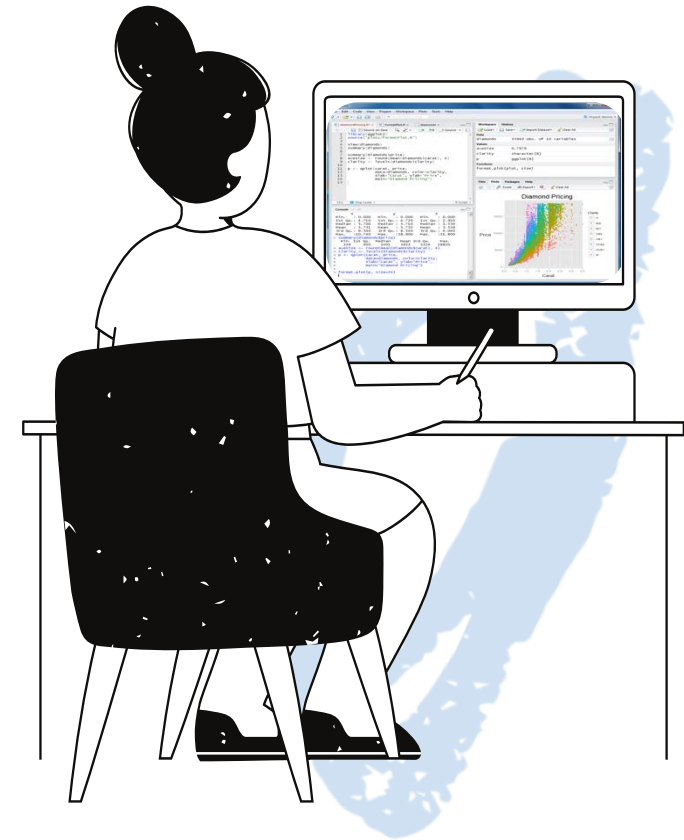
- R automates repetitive tasks such as data cleaning, transformation, model execution, and visualization through flexible scripting and powerful packages
- R supports scheduled and event-driven workflows via cron jobs, task schedulers, and orchestration tools like **Airflow**, **Nextflow**, and **GitHub** actions
- Tools such as **R Markdown** and **Quarto** automate dynamic reporting, while APIs and database connectivity streamline automated data ingestion and publishing
- In regulated workflows, R supports structured logging, error handling, Shiny dashboards, and automated alerts for monitoring batch execution and ensuring auditability





Framework Design

- 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
- The batch runner is implemented in base R operating system invocation via ***shell()***



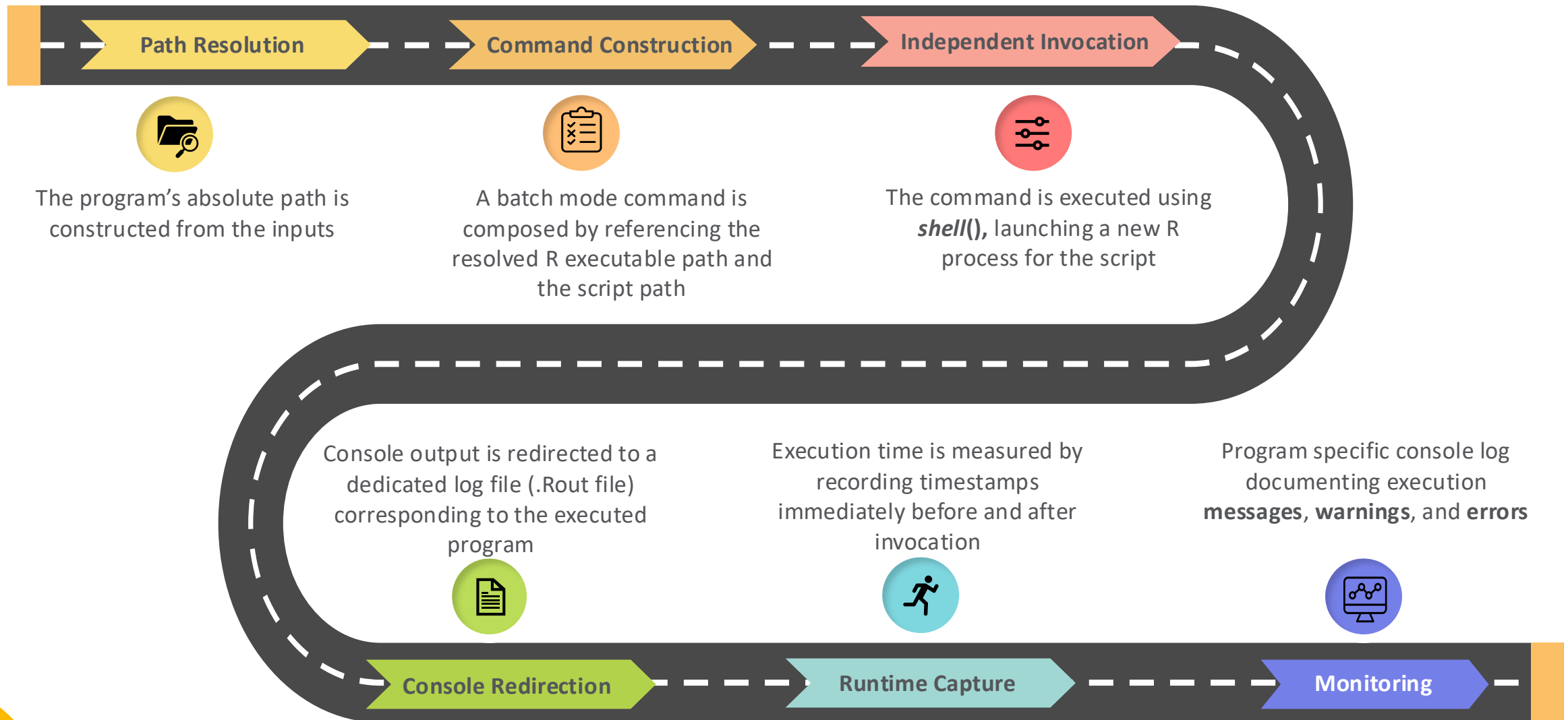


Implementation Approach

- The framework uses **base R** and Windows command execution
- **R CMD BATCH** executes each domain script in a **non-interactive R session**
- A **predefined ordered list** of SDTM/ADaM programs ensures that scripts execute in the correct sequence using a simple for loop
- The **shell()** function acts as a bridge between R and the operating system
- When shell() is called, the OS launches **cmd.exe**, runs the command (including R.exe), and returns control to R once execution finishes
- R receives the **exit status** and, depending on settings, can also capture console output for monitoring and error handling



Program Workflow



Core Function

rbatch_run is a custom R function developed to enable automated batch execution of pre-defined R programs in a controlled sequence

```
rbatch_run <- function(File = "dm.R", progdir = 'validation/program'){  
  logfile <- normalizePath(file.path(progdir,  
                                     paste0(gsub('.R$', '', basename(file)), ".Rout")))  
  execution_time <- system.time(  
    shell(paste0(normalizePath(file.path(R.home("bin"), "R"), mustWork = FALSE),  
                 'CMD BATCH ', "--vanilla", ' "', file.path(progdir, File),  
                 '" "', logfile, '"'), wait = TRUE))  
  run_time <- paste(round(execution_time[['elapsed']], 3), 'seconds')  
  cat("\rExecution time for the domain `", basename(file), "` : ", run_time, "\n", sep = "")  
  invisible(logfile)  
}
```



Implementation Scripts

- A script is required to serve as the main implementation layer for the batch-run workflow using the ***rbatch_run()*** function, primarily responsible for preparing the execution order by identifying and organizing the program paths
- Once the sequence is finalized, each program is executed in order via a for loop calling ***rbatch_run()***
- The workflow ensures standardized log creation and consistent timing capture for every script
- This approach enables transparent, reproducible, and uniform end-to-end batch processing



Implementaion Scripts - II

```
# ===== Get list of programs =====  
all.progms <- list.files('validation/program', 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)  
  
# ===== Execute =====  
for (i in run.List){  
  rbatch_run(i, progdir = dir.prod.prog)  
}
```

Results

```
> # ==== 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 ...!
```



Future Work



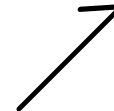
Enable parallel execution to speed up overall SDTM processing instead of running programs strictly sequentially.



Identify domain dependencies to classify scripts into prerequisite, independent, and downstream groups.



Execute independent domains concurrently using multiple R sessions.



Parallel Execution



Introduce “wait points” to ensure downstream domains run only after required upstream outputs are available.



Implement resource management (CPU, memory) to prevent overload when running multiple jobs at once.



Improve scalability, reduce runtime, and support integration with enterprise schedulers and cloud environments.



Conclusion

- The framework provides an **R-based batch execution system** for SDTM/ADaM that supports non-interactive, repeatable, and ordered program execution using R CMD BATCH via shell()
- Each domain runs in an independent R session, enabling **clean log generation, execution time tracking, and reduced session carryover risk**
- The approach is suitable for **scheduled, automated, and command-line workflows**, forming a scalable foundation for SDTM/ADaM pipeline automation
- R's expanding role in clinical programming is supported by its **rich ecosystem of validated packages**, open-source reproducibility, and tools like **R Markdown, Shiny, Quarto, Git, and Posit Connect**
- With growing regulatory acceptance of open-source submissions, R is becoming a **future-ready platform for clinical analytics**, enabling efficient pipelines, advanced methodologies, and scalable automation

THANK YOU

QUESTIONS?