SESUG Paper VR-97-2025

How to Create a Spreadsheet-driven Batch Job Using R and SASSY

David J. Bosak, r-sassy.org

ABSTRACT

It is very common to use spreadsheets to drive batch jobs in SAS. The spreadsheets allow you to easily control which programs are run, and in what order. They also provide a centralized place to store titles, footnotes, and other information used in the programs. Is it possible to create a similar system in R? Yes! This paper will demonstrate how to do it using a typical spreadsheet layout, and reports written with the SASSY set of R packages.

INTRODUCTION

Sometimes you have a lot of programs that you want to run in batch. In SAS, it is quite common to put all the program names in an Excel spreadsheet, and let the spreadsheet drive the batch job. It is also convenient to put some parameters for those programs in the spreadsheet. Putting parameters in the spreadsheet gives you a central place to manage those parameters values, which can make your work more efficient.

Having done this task in SAS, you may ask: Is there a way to do it in R? The answer is, of course, yes! This paper will explore one way to do it.

To create a spreadsheet-driven batch job in R, we will take the following steps:

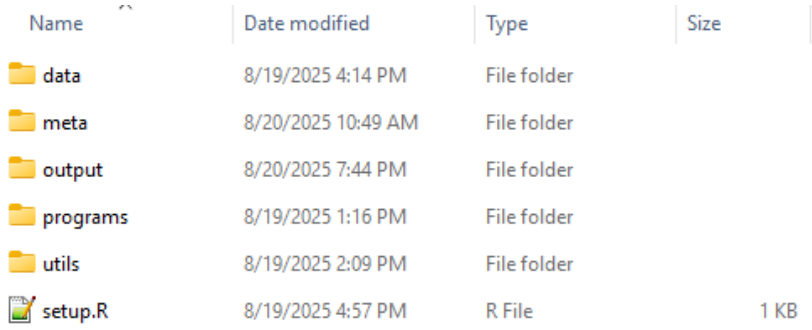
1) Create a project folder.
2) Define the batch program spreadsheet.
3) Define an R object to hold program information.
4) Write utility functions to extract program information from the spreadsheet.
5) Create some sample programs.
6) Write the batch execution routine.
7) Run the batch job.

Note that that the code in this paper has been greatly simplified to make the key elements of the project clearer.

The sample programs will also use some functions from the SASSY set of R packages. These functions provide some convenience, but ultimately are not required for the development of such a system.

PROJECT FOLDER

The project folder is defined as follows. This folder structure will hold the various files we create as part of the batch system:



Name	Date modified	Type	Size
data	8/19/2025 4:14 PM	File folder	
meta	8/20/2025 10:49 AM	File folder	
output	8/20/2025 7:44 PM	File folder	
programs	8/19/2025 1:16 PM	File folder	
utils	8/19/2025 2:09 PM	File folder	
setup.R	8/19/2025 4:57 PM	R File	1 KB

DEFINE SPREADSHEET

The second step in creating a spreadsheet-driven batch system is to define the spreadsheet. For this paper, we will use a simplified spreadsheet. The spreadsheet has the program name, description, titles, and footnotes. The spreadsheet also has an "Active" flag that will allow us to indicate whether to include a program in the batch job or not. Here is the spreadsheet:

	A	B	C	D	E	F	G	H
1	**Program**	**Description**	**Active**	**Title1**	**Title2**	**Title3**	**Footnote1**	**Footnote2**
2	l_dm	Demographics	Y	Listing 1.0	Subject Demographics	All Subjects	Program: l_dm.R	Data: DM.rds
3	l_ae	Adverse Events	Y	Listing 2.0	Subject Adverse Events	All Subjects	Program: l_ae.R	Data: AE.rds
4	l_ds	Disposition	Y	Listing 3.0	Subject Disposition	All Subjects	Program: l_ds.R	Data: DS.rds
5	l_vs	Vital Signs	N	Listing 4.0	Subject Vital Signs	All Subjects	Program: l_vs.R	Data: VS.rds

For this project, the spreadsheet will be named "study_info.xlsx", and placed in the "meta" folder.

DEFINE PROGRAM INFORMATION OBJECT

The next step in our project is to define an R object to hold information about a program. This object will contain all the fields in one row of the spreadsheet. In R, we can define such an object as follows:

```
# An S3 object definition for program information
pgm_info <- function(program, description = "", active = "",
                     titles = c(), footnotes = c()) {

  # Define pgm_info class
  x <- structure(list(), class = c("pgm_info", "list"))

  # Populate properties
  x$program <- program
  x$description <- description
  x$active <- active
  x$titles <- titles
  x$footnotes <- footnotes

  return(x)
}
```

The above code defines an S3 object of class "pgm_info". In R, it is very easy to create such an object. The `structure()` function defines the class. The class is derived from a list. Then the list is populated with the function parameters, and returned.

This code will be placed in a file called *utils.R*, and located in the "utils" project folder.

WRITE UTILITY FUNCTIONS

Get Program Information

Now that we have a class to hold our program information, let's write a utility function to retrieve it. This function will accept a path to the spreadsheet, and the name of the program for which we want information:

```
# Get program info from spreadsheet
get_info <- function(file_path, program) {

  if (!file.exists(file_path))
    stop("file_path does not exist")

  # Read in file
```

```
  inf <- readxl::read_xlsx(file_path)

  # Get requested program
  rw <- subset(inf, Program == program)

  # Find titles
  tpos <- grepl("title", names(rw), ignore.case = TRUE)
  ttls <- rw[tpos]

  # Find footnotes
  fpos <- grepl("footnote", names(rw), ignore.case = TRUE)
  fnts <- rw[fpos]

  # Instantiate pgm_info object
  ret <- pgm_info(program = rw[["Program"]],
                  description = rw[["Description"]],
                  active = rw[["Active"]],
                  titles = c(ttls[!is.na(ttls)]),
                  footnotes = c(fnts[!is.na(fnts)]))

  return(ret)
}
```

In the above code, observe that titles and footnotes have each been collapsed into a single vector. These vectors are then passed as parameters to our `pgm_info()` function defined above. The result is then returned as a fully populated program information object.

Get Programs

Next, we need another utility function to get the names of the programs in the spreadsheet. These names are required for our batch job:

```
# Get program names from spreadsheet
get_programs <- function(file_path, all = FALSE) {

  if (!file.exists(file_path))
    stop("file_path does not exist")

  # Read in file
  inf <- readxl::read_xlsx(file_path)

  # Subset for active
  if (!all) {
    inf <- subset(inf, Active == 'Y')
  }

  # Get program names
  ret <- inf$Program

  return(ret)

}
```

As with the `get_info()` function, `get_programs()` also accepts the path to the spreadsheet as the first parameter. For the second parameter, however, `get_programs()` has a parameter to indicate

whether you want all of the programs returned, or only the active programs. By default, only the active programs are returned. The return value is a vector of program names to run.

The `get_programs()` function is also located in the *utils.R* file.

CREATE SAMPLE PROGRAMS

Setup File

We have reached the point where we need some programs to run. For this paper, we will create very simple listing programs. The listing programs have some shared information. So first, we will need to create a setup file to store the shared information.

```
# Global variables
sponsor_name <- "Archytas"
study_name <- "ABC"

# Working Directory
study_dir <- "C:/packages/Testing/batch_reports"
setwd(study_dir)

# Other Directories
program_dir <- "./programs"
data_dir <- "./data"
output_dir <- "./output"

# Paths
study_info_path <- "./meta/study_info.xlsx"
batch_utils_path <-  "./utils/batch_utils.R"

# Load Scripts
source(batch_utils_path)
```

The above setup code accomplishes a few things:

1) Populates global variables.
2) Sets working directory.
3) Defines paths relative to working directory.
4) Identifies location of spreadsheet and script files.
5) Loads the scripts into memory.

The setup ties together the various parts of the system. It should be run prior to running the batch script.

Sample Listing Program

Now we are ready to write a sample listing program. This listing code below is from the program *l_dm.R*. All the other listing programs are basically the same. Only the program name and the input dataset used are different:

```
library(reporter)

# Find the nearest setup file
fl <- common::file.find(pattern = "setup.R", up = 5, down = 0)[1]

# Source the setup file
source(fl)
```

```
# Program name
pgm <- "l_dm"

# Output path
out_pth <- file.path(output_dir, pgm)

# Get program info
info <- get_info(study_info_path, pgm)

# Get listing data
dat <- readRDS(file.path(data_dir, "DM.rds"))

# Create table object
tbl <- create_table(dat) |>
  define(USUBJID, id_var = TRUE)

# Create report object
rpt <- create_report(out_pth, font = "Courier", output_type = "RTF") |>
  page_header("Sponsor: " %p% sponsor_name, "Study: " %p% study_name) |>
  titles(info$titles) |>
  add_content(tbl, align = "left") |>
  footnotes(info$footnotes) |>
  page_footer(Sys.time(), "CONFIDENTIAL", "Page [pg] of [tpg]")

# Write report to file
write_report(rpt)
```

Note the following about the above sample listing:

- The program finds and sources the *setup.R* file, which loads several variables used in the program. It uses the `file.find()` function from the common package to search for the setup. Variables used from the setup include the `sponsor_name`, `study_name`, and location of the spreadsheet held in `study_info_path`.
- The program pulls a "pgm_info" object from the spreadsheet using the `get_info()` function defined earlier. The titles, and footnotes from the "pgm_info" are then passed to the reporting functions.
- The program uses the reporter package to create the listing. This package handles creation of the header and footer, titles and footnotes, plus the body of the report. The **reporter** package will automatically insert horizontal and vertical page breaks, which makes listings very easy.

It is also worth noting that the program can be run independently. That is to say, the listing program can be run all by itself, outside the batch system. This design makes it easy to write and debug programs that will be part of the batch process.

The above listing program and all other programs in the batch run are stored in the "programs" project directory.

WRITE BATCH EXECUTION ROUTINE

The last part of our system is the batch execution routine itself. All the other components of the system feed into the batch routine. Let's see how it works:

```
# Routine to execute the batch run
run_batch <- function(file_name = "./meta/study_info.xlsx") {
```

```
  # Assign base programs directory
  base_dir <- program_dir

  # Get batch programs
  pgms <- get_programs(file_name)

  # Execute programs
  for (pgm in pgms) {

    # Construct program name
    fl <- file.path(base_dir, paste0(pgm, ".R"))

    # Execute program
    source(fl, local = TRUE)

  }
}
```

The batch routine has a very simple logic:

1) Get a vector of active program names from the spreadsheet.
2) Loop through the program names and construct a full path for each.
3) Source each program, setting the `local` parameter to TRUE.

The `local` parameter ensures that each program will be run in a local environment instead of the global environment. Running in a local environment is important to avoid one program clashing with the next via shared variable or function names.

The above batch routine is contained in the *utils.R* script file in the "utils" project directory.

EXECUTE BATCH RUN

Now is the moment we've been waiting for. Let's execute the batch run!

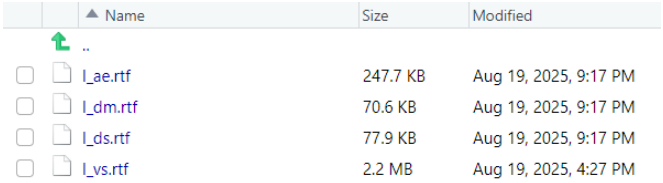
Run Batch Routine

Given all the work we have done previously, executing the batch run is very easy. There are only two steps:

1) Run the *setup.R* to initialize the environment and load the batch routine.
2) Execute the batch routine, either from the console or another script file. You will execute the batch script like this:

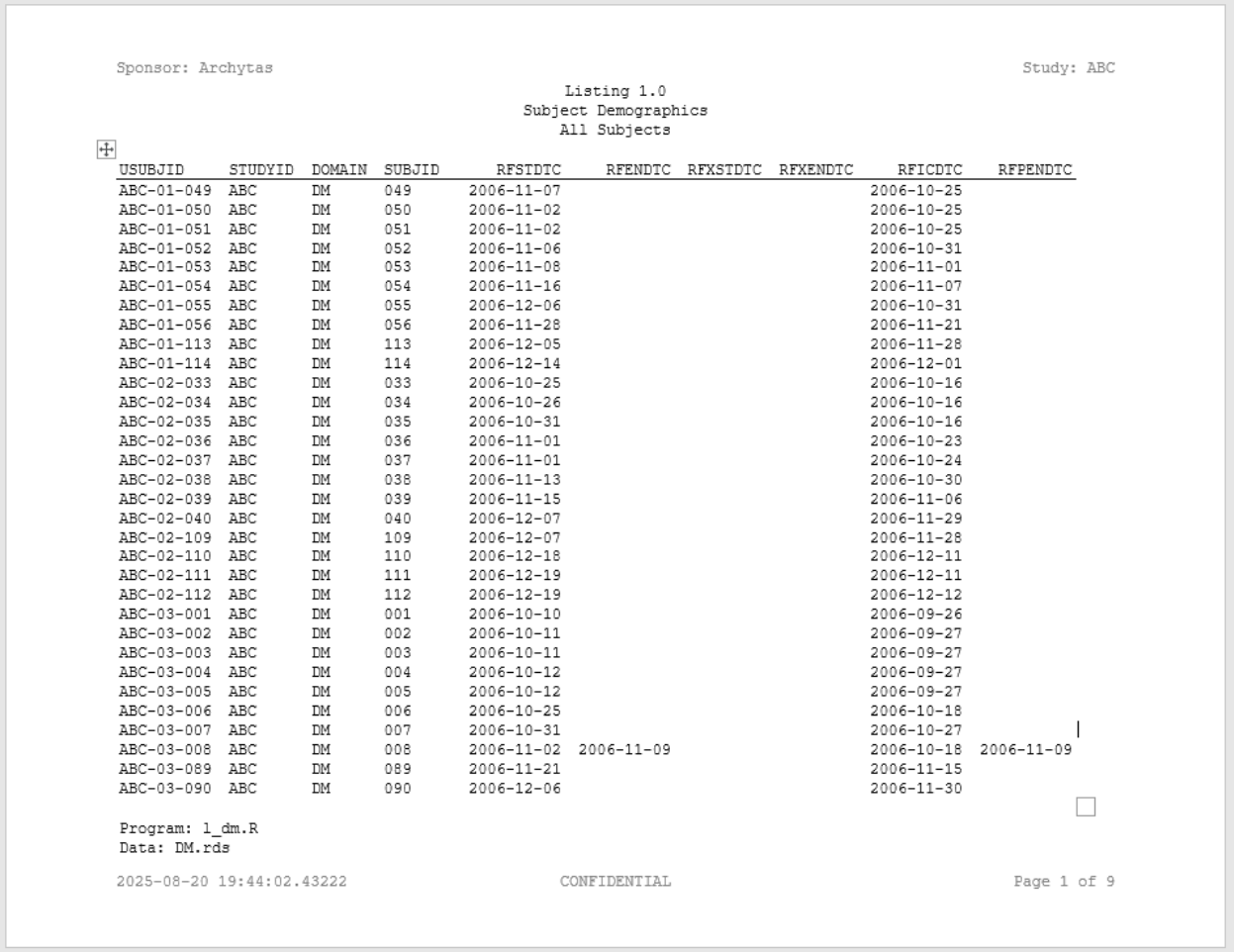
```
# Execute the batch run
run_batch()
```

That's it! The batch routine will now run each program that is activated in the spreadsheet. The listing output can be seen in the "output" directory:



View Sample Listing

And the sample listing will look like this:



Sponsor: Archytas — Study: ABC

Listing 1.0
Subject Demographics
All Subjects

USUBJID	STUDYID	DOMAIN	SUBJID	RFSTDTC	RFENDTC	RFXSTDTC	RFXENDTC	RFICDTC	RFPENDTC
ABC-01-049	ABC	DM	049	2006-11-07				2006-10-25	
ABC-01-050	ABC	DM	050	2006-11-02				2006-10-25	
ABC-01-051	ABC	DM	051	2006-11-02				2006-10-25	
ABC-01-052	ABC	DM	052	2006-11-06				2006-10-31	
ABC-01-053	ABC	DM	053	2006-11-08				2006-11-01	
ABC-01-054	ABC	DM	054	2006-11-16				2006-11-07	
ABC-01-055	ABC	DM	055	2006-12-06				2006-10-31	
ABC-01-056	ABC	DM	056	2006-11-28				2006-11-21	
ABC-01-113	ABC	DM	113	2006-12-05				2006-11-28	
ABC-01-114	ABC	DM	114	2006-12-14				2006-12-01	
ABC-02-033	ABC	DM	033	2006-10-25				2006-10-16	
ABC-02-034	ABC	DM	034	2006-10-26				2006-10-16	
ABC-02-035	ABC	DM	035	2006-10-31				2006-10-16	
ABC-02-036	ABC	DM	036	2006-11-01				2006-10-23	
ABC-02-037	ABC	DM	037	2006-11-01				2006-10-24	
ABC-02-038	ABC	DM	038	2006-11-13				2006-10-30	
ABC-02-039	ABC	DM	039	2006-11-15				2006-11-06	
ABC-02-040	ABC	DM	040	2006-12-07				2006-11-29	
ABC-02-109	ABC	DM	109	2006-12-07				2006-11-28	
ABC-02-110	ABC	DM	110	2006-12-18				2006-12-11	
ABC-02-111	ABC	DM	111	2006-12-19				2006-12-11	
ABC-02-112	ABC	DM	112	2006-12-19				2006-12-12	
ABC-03-001	ABC	DM	001	2006-10-10				2006-09-26	
ABC-03-002	ABC	DM	002	2006-10-11				2006-09-27	
ABC-03-003	ABC	DM	003	2006-10-11				2006-09-27	
ABC-03-004	ABC	DM	004	2006-10-12				2006-09-27	
ABC-03-005	ABC	DM	005	2006-10-12				2006-09-27	
ABC-03-006	ABC	DM	006	2006-10-25				2006-10-18	
ABC-03-007	ABC	DM	007	2006-10-31				2006-10-27	
ABC-03-008	ABC	DM	008	2006-11-02	2006-11-09			2006-10-18	2006-11-09
ABC-03-089	ABC	DM	089	2006-11-21				2006-11-15	
ABC-03-090	ABC	DM	090	2006-12-06				2006-11-30	

Program: l_dm.R
Data: DM.rds

2025-08-20 19:44:02.43222 — CONFIDENTIAL — Page 1 of 9

Note that the titles and footnotes are populated dynamically from the parameters in the spreadsheet. Also note the header values are populated dynamically from the setup. In this way, you can execute a large number of programs, and ensure consistency across the entire set.

CONCLUSION

This paper illustrates one way to create a spreadsheet-driven batch system in R. The system described allows you to share values across programs using a setup file. The system also consolidates titles, footnotes, and other parameters into the batch spreadsheet. A valuable feature of the design is that it allows you to write, debug, and run each program independently, yet still execute all programs in batch. While the system presented is somewhat simplified, you will be able to easily adopt the code above to create your own R batch framework.

REFERENCES

Bosak, David J. 2025. “Why SASSY?” *Proceedings of PharmaSUG*. Available at https://pharmasug.org/proceedings/2025/AP/PharmaSUG-2025-AP-263.pdf

Bosak, David J. “The SASSY System.” 2025. Available at https://r-sassy.org/.

CONTACT INFORMATION

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

David J. Bosak
Archytas Clinical Solutions, LLC
dbosak01@gmail.com
www.r-sassy.org