PharmaSUG China 2024 - Paper AP-058

A R shiny app creating scripts for batch running R programs

Wayne Yang, Gloria Zhong, and Sukie Gao Merck Serono R&D Hub

ABSTRACT

This paper seeks to elucidate the utilization of R shiny apps in the generation of batch scripts, particularly focusing on the enhancement of clinical data analysis.

Potential users include R users who need to run batch of R programs for clinical data creation and outputs generation.

INTRODUCTION

As R programming is more and more popular in clinical data analysis. However, Statistical & Computational Environment (SCE) in most of organization is SAS-based and the infrastructure for R-based programming is not complete, e.g., the batch run of a large number of R programs in a specific scope. How to efficiently run the desired programs in current SCE is an important element to delivery.

Especially in some repositories, there is no automatic registration in the version control system of outputs generated by R programs, that requires all outputs must to be declared as parameters to the program. But the drawback is that it entails declaring and maintaining large number of file paths, which is inefficient.

In addition, given that there are various purposes requiring different batching scopes, this R shiny app is developed to ease the drawback and to create batch scripts which are in R language and based on either metadata specifying executing scope or a folder including R programs in an intuitive and interactive way.

Since there are numerous environment and situation which is hard to be exhausted, hopefully this paper can provide you with some inspiration.

PRIMARY BATCH RUN CODES

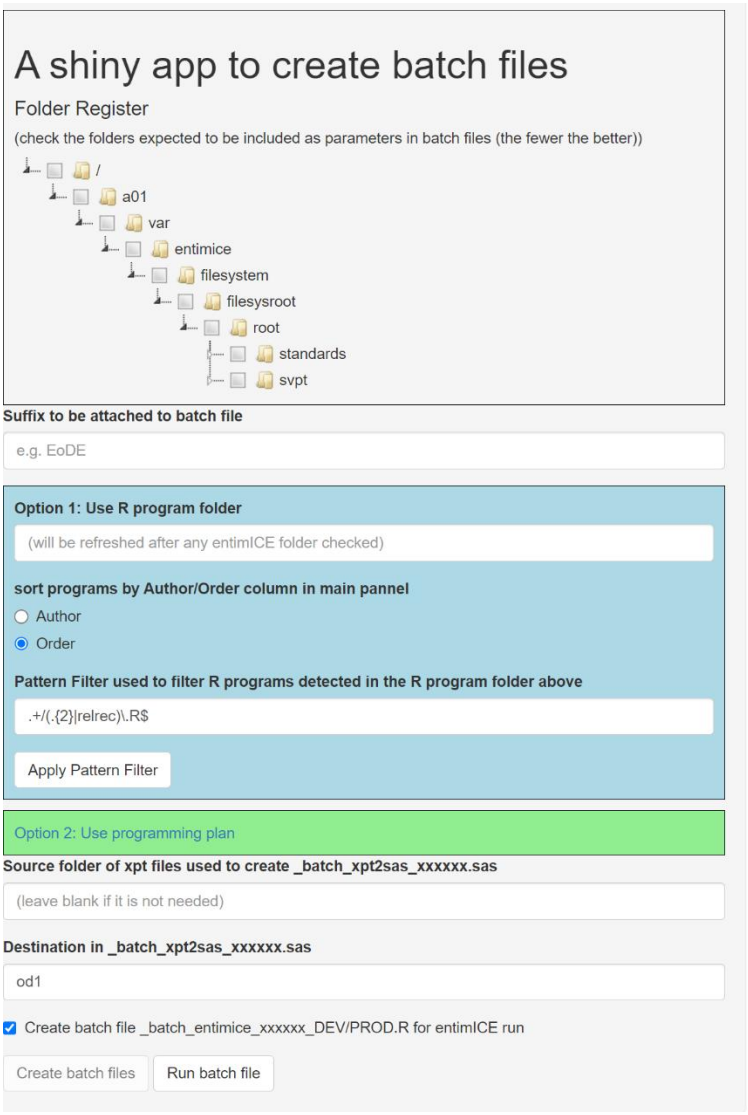
Basically, you can use following codes to batch run R programs in a shell of UNIX or Windows:

```
UNIX:     R CMD BATCH XXXX.R
Windows:  "XXXX\R.exe" CMD BATCH "XXXX.R"
```

But for some repositories which require programs to be run inside and do not have shell available for users, it entails a workaround like Display 15. An interactive interface and pre-programmed framework can significantly reduce the difficulty of generating such scripts for users.

THE R SHINY APP INTERFACE

This R shiny app has a side-by-side interface shown in Display 1. You can use left side to provide inputs, e.g., folders used as parameters which are dedicated for some repositories batching, suffix for the scripts created by this app, R program folder, metadata, etc. The right side reacts to your inputs, you can provide further inputs along with the information on the right side.



Display 1 Main Interface

THE FLOWCHART

To better explain the details of this R shiny app, you can refer to the Figure 1. The details will be described in the following sections.

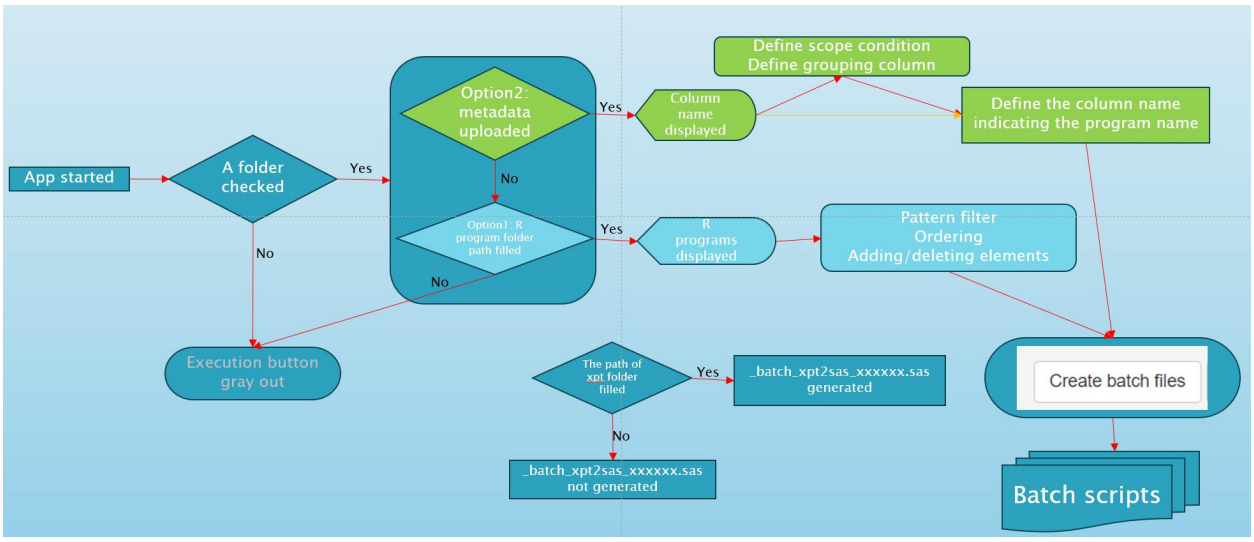


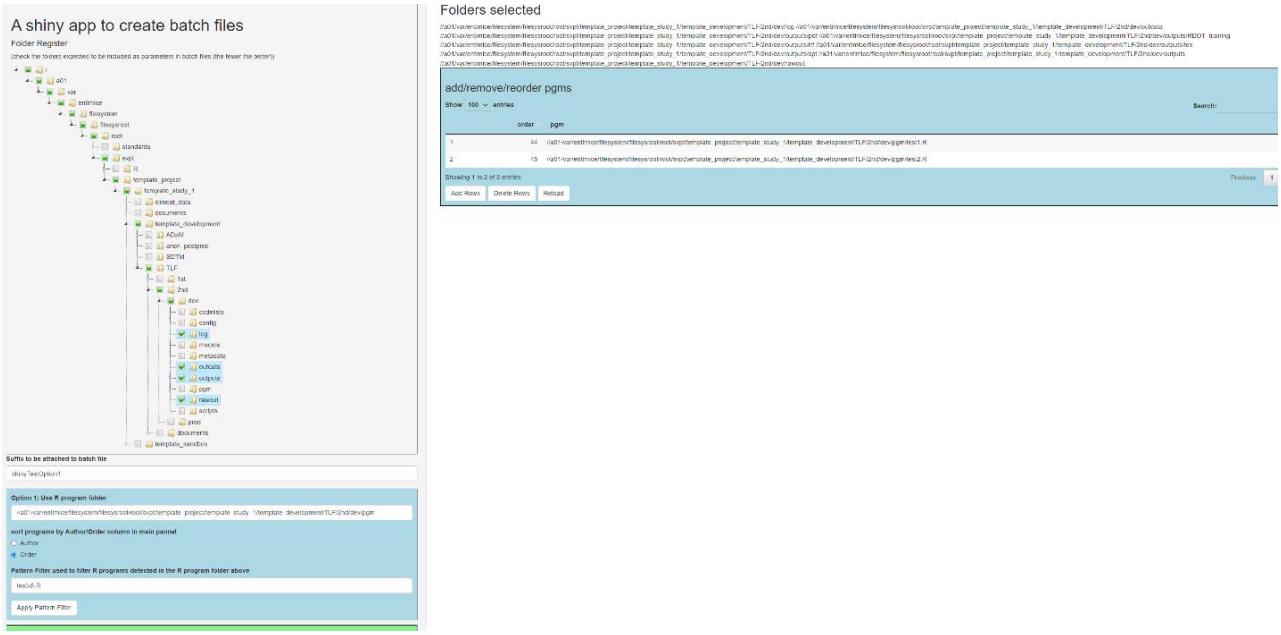
Figure 1 The Flowchart

THE USAGE

There are 2 options to create batch scripts. For tiny studies that do not have metadata/programming plan, you can use option1 by filling in a folder path storing R programs. For the studies having programming plan, option2 is recommended for a full functionality of this app.

OPTION1

You can select the folders from the tree shown in Display 2 first. Once any folder is checked, a folder path where R programs are located is automatically filled, indicating option1 has been chosen. Meanwhile, the R programs in the folder are listed on right side with the same background color. You can apply pattern filter to the list and sort programs by either the author of each program or the column order in the list. You can delete rows, add rows and/or reorder rows. These operations are reflected in all batch scripts later.

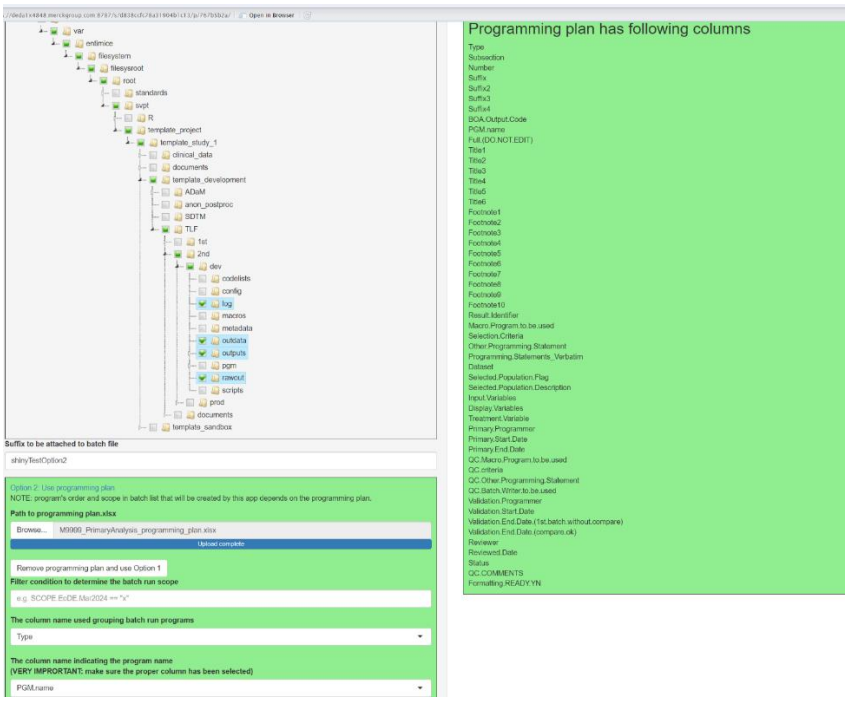


Display 2 option1

OPTION2

You can click the hyperlink “Option 2: Use programming plan” and upload a metadata/programming plan which should contain a column with unique program name, e.g., Display 4. In this case, option2 is activated and the area of option1 is hidden and overridden shown in Display 3.

You can go back to option1 by clicking the button of “Remove programming plan and use Option 1”.



Display 3 option2

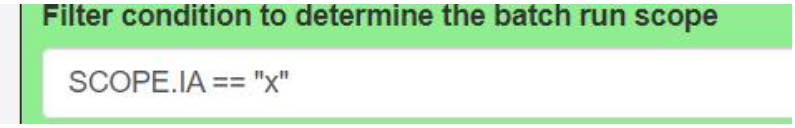
	A	B	C	D	E	F	G	H	I	
1	Type	Subsection	Number	Suffix	Suffix 2	Suffix 3	Suffix 4	BOA Output Code	PGM name	Fu (D ED
2	T	8	1	1	1			ANASET	t_disp_anaset	8.
3	T	8	1	1	2			SITE	t_disp_site	8.
	T	8	1	1	3			DISP1	t_disp_disp1	8.

Display 4 an example of programming plan containing a column with unique program name

You can create various batch scripts based on different scope by adding dedicated columns into programming plan like Display 5 and filling in conditions accordingly in the app, e.g., Display 6.

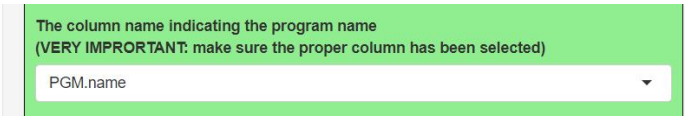
	A	B	C	D	E	F	G	H	I	J	
1	Type	Subsection	Number	Suffix	Suffix 2	Suffix 3	Suffix 4	BOA Output Code	PGM name	SCOPE IA	Fu (D ED
2	T	8	1	1	1			ANASET	t_disp_anaset	x	8.
3	T	8	1	1	2			SITE	t_disp_site		8.
4	T	8	1	1	3			DISP1	t_disp_disp1	x	8.
5	T	8	1	2	1			PPDEV	t_dev_ppdev_saf		8.

Display 5 an example of adding a scope column



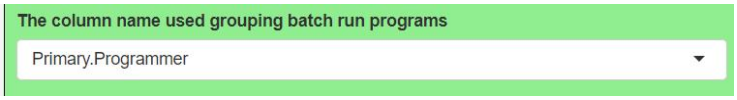
Display 6 filter condition

You need to select a column where program names are located, e.g., Display 7.



Display 7 selecting a column having program name information

You can group R programs together by selecting a column, e.g., Display 8. If the batch run order is critical, leave this input blank. Then the order will be the order in the metadata/programming plan.

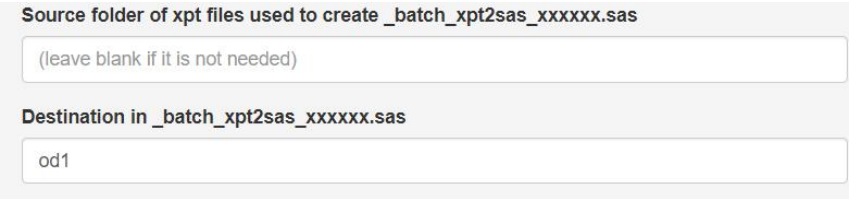


Display 8 selecting a column grouping programs

CREATING XPT2SAS

This app also provides a functionality to generate a SAS program you can use to convert the xpt files into sas7bdat format for QC purpose, given that the sas7bdat generated by R cannot be recognized by SAS and it entails involving xpt file as intermediate substance.

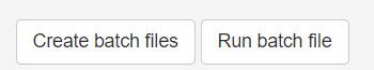
You can fill in the folder path where xpt files are located to activate this function, and specify the destination, in other words, the libname where you want to convert xpt files to.



Display 9 xpt convertion script setting

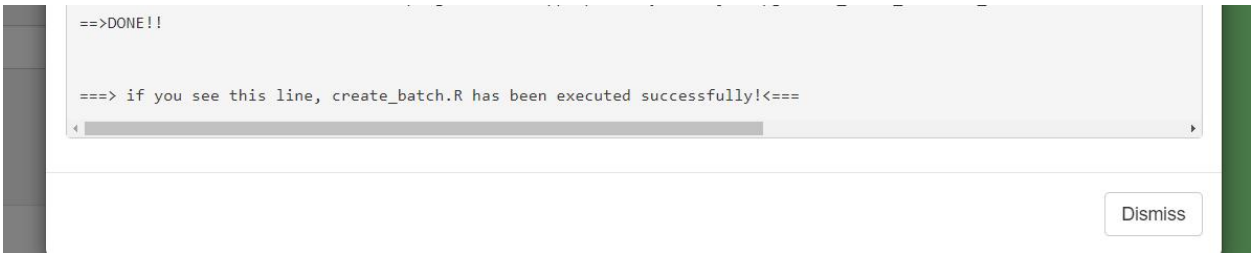
EXECUTING

Once you finish above settings, you can click button “Create batch files” to submit to server to generate batch scripts.



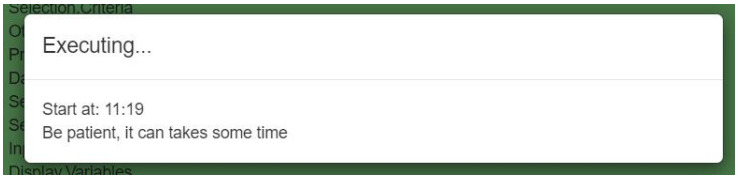
Display 10 execution button

And a log window shows up to confirm if any issues occur.

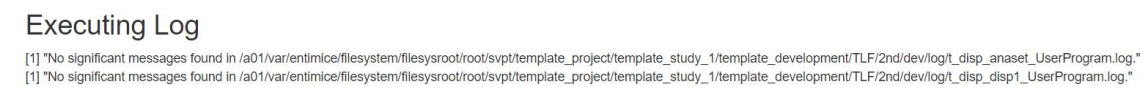


Display 11 log window of batch script generation

Once the batch scripts are generated, you can click button “Run batch file” to have a quick dry run Display 12 in current environment. And log check result Display 13 returns if it is configured in your environment.



Display 12 waiting window



Display 13 log of executing batch script

THE BATCH SCRIPTS

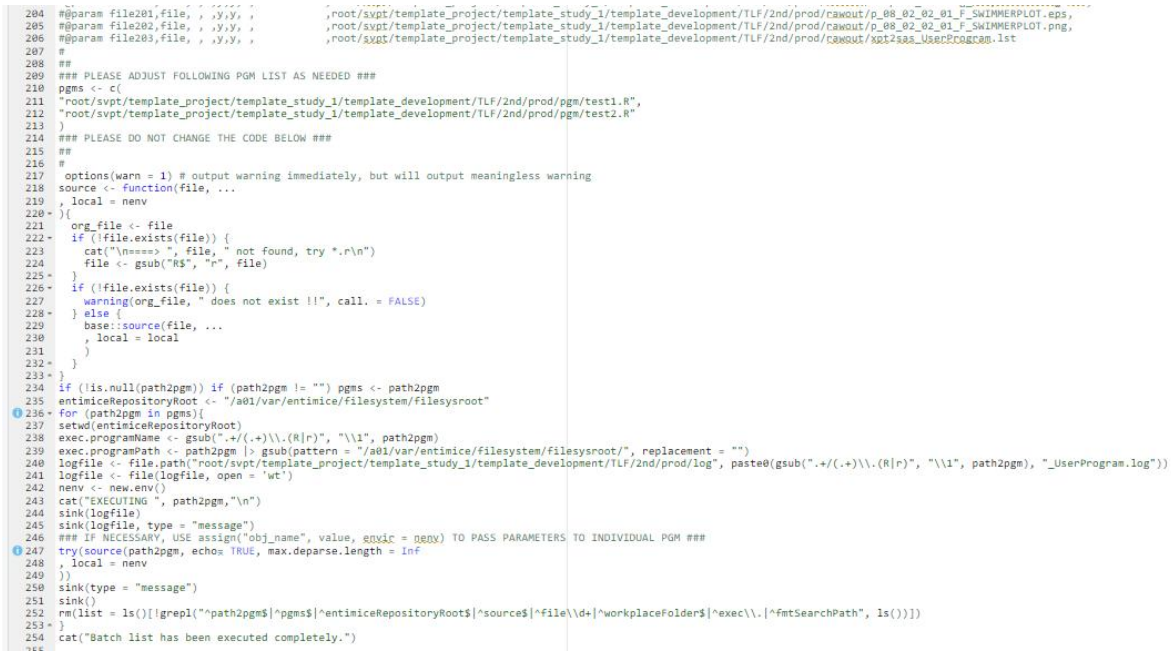
Following scripts in Display 14, Display 15 and Display 16 are generated by this app.

Display 14 is a R script for batch running in Linux.

```
1  .cmd <- "perl [redacted] "
2  .pgm_folder <- "[redacted]TLF/2nd/dev/pgm/"
3  ## Primary.Programmer: SG
4  system(intern = TRUE, paste0(.cmd, .pgm_folder, "l_dev_covid_saf.R"))
5  system(intern = TRUE, paste0(.cmd, .pgm_folder, "t_trtc_exaction.R"))
6  system(intern = TRUE, paste0(.cmd, .pgm_folder, "t_os_effos2.R"))
7  system(intern = TRUE, paste0(.cmd, .pgm_folder, "f_os_effkm.R"))
8  system(intern = TRUE, paste0(.cmd, .pgm_folder, "t_pfs_effpfs2.R"))
9  system(intern = TRUE, paste0(.cmd, .pgm_folder, "f_pfs_effkm.R"))
10 system(intern = TRUE, paste0(.cmd, .pgm_folder, "f_or_spiderplot.R"))
11 system(intern = TRUE, paste0(.cmd, .pgm_folder, "f_dor_effkm.R"))
12 system(intern = TRUE, paste0(.cmd, .pgm_folder, "t_dlt_dlt.R"))
13
14 ## Primary.Programmer: VR
15 system(intern = TRUE, paste0(.cmd, .pgm_folder, "l_discon.R"))
16
17 ## Primary.Programmer: WY
18 system(intern = TRUE, paste0(.cmd, .pgm_folder, "t_trtc_exdose.R"))
19 system(intern = TRUE, paste0(.cmd, .pgm_folder, "t_ae_teaesocpt_nons.R"))
```

Display 14 batch file for Linux

Display 15 is a R script for some repository environment requiring to run programs inside and parameters (~ row 206). While running this script, an independent environment is created for each program in the batch list to avoid interaction among programs’ execution.



```
#@param file201,file, , ,y,y, ,          ,root/svpt/template_project/template_study_1/template_development/TLF/2nd/prod/rawout/p_08_02_02_01_F_SWIMMERPLOT.eps,
#@param file202,file, , ,y,y, ,          ,root/svpt/template_project/template_study_1/template_development/TLF/2nd/prod/rawout/p_08_02_02_01_F_SWIMMERPLOT.png,
#@param file203,file, , ,y,y, ,          ,root/svpt/template_project/template_study_1/template_development/TLF/2nd/prod/rawout/xpt2sas_UserProgram.lst
#
##
### PLEASE ADJUST FOLLOWING PGM LIST AS NEEDED ###
pgms <- c(
"root/svpt/template_project/template_study_1/template_development/TLF/2nd/prod/pgm/test1.R",
"root/svpt/template_project/template_study_1/template_development/TLF/2nd/prod/pgm/test2.R"
)
### PLEASE DO NOT CHANGE THE CODE BELOW ###
##
#
 options(warn = 1) # output warning immediately, but will output meaningless warning
source <- function(file, ...
, local = nenv
){
  org_file <- file
  if (!file.exists(file)) {
    cat("\n====> ", file, " not found, try *.r\n")
    file <- gsub("R$", "r", file)
  }
  if (!file.exists(file)) {
    warning(org_file, " does not exist !!", call. = FALSE)
  } else {
    base::source(file, ...
    , local = local
    )
  }
}
if (!is.null(path2pgm)) if (path2pgm != "") pgms <- path2pgm
entimiceRepositoryRoot <- "/a01/var/entimice/filesystem/filesysroot"
for (path2pgm in pgms){
setwd(entimiceRepositoryRoot)
exec.programName <- gsub(".+/(.+)\\.(R|r)", "\\1", path2pgm)
exec.programPath <- path2pgm |> gsub(pattern = "/a01/var/entimice/filesystem/filesysroot/", replacement = "")
logfile <- file.path("root/svpt/template_project/template_study_1/template_development/TLF/2nd/prod/log", paste0(gsub(".+/(.+)\\.(R|r)", "\\1", path2pgm), "_UserProgram.log"))
logfile <- file(logfile, open = 'wt')
nenv <- new.env()
cat("EXECUTING ", path2pgm,"\n")
sink(logfile)
sink(logfile, type = "message")
### IF NECESSARY, USE assign("obj_name", value, envir = nenv) TO PASS PARAMETERS TO INDIVIDUAL PGM ###
try(source(path2pgm, echo= TRUE, max.deparse.length = Inf
, local = nenv
))
sink(type = "message")
sink()
rm(list = ls()[!grepl("^path2pgm$|^pgms$|^entimiceRepositoryRoot$|^source$|^file\\d+|^workplaceFolder$|^exec\\.|^fmtSearchPath", ls())])
}
cat("Batch list has been executed completely.")
```

Display 15 batch file for some repository

Display 16 is a SAS script for converting xpt to sas7bdat.

```
%startup(debug=N);
%macro xpt2loc2(lib, xptfile);
  filename xptfile "&_g_root_path_./outputs/xpt/&xptfile.";
  %xpt2loc(libref= &lib.,filespec=xptfile);
%mend;
%xpt2loc2(od2, f_08_03_03_21_01_lbedish.xpt);
%xpt2loc2(od2, f_08_03_03_21_02_lbedish.xpt);
%xpt2loc2(od2, l_08_01_02_02_ppexcl.xpt);
%xpt2loc2(od2, t_08_01_01_01_disp1.xpt);
%xpt2loc2(od2, t_08_01_01_02_randvstrt.xpt);
%xpt2loc2(od2, t_08_01_01_03_anaset.xpt);
%xpt2loc2(od2, t_08_01_01_03_disp1.xpt);
```

Display 16 xpt2sas conversion program

SNIPPETS OF CODE

Following libraries are used in this app:

```
openxlsx_4.2.5
DT_0.24
dplyr_1.0.9
htmltools_0.5.3
shinyjs_2.1.0
shinyTree_0.2.7
shiny_1.7.2
```

Following code is used to prepare the contents in Display 14:

```
batchlist <- lapply(seq_along(bygroup),
    function(x) {
      c(
```

```
        if(.byauthor) {
          paste0("## Programmer: ", names(bygroup)[x])
        } else{
          NULL
        },
        paste0("system(intern =TRUE, 'perl
/NES/root/standards/programs/perl/runR.pl ", bygroup[[x]]$files ,"')"),
        ""
      )
    } ) |> unlist()
```

xpt files filtered against metadata/programming plan if Option2 is used:

```
if (!is.null(.programming_plan)){
  cat("\n===> NOTE:: .programming_plan is available.")
  cat("\n===> NOTE:: filtering xpts according to programming plan.")
  xpts <- lapply(spec, \(x) {
    detector  <-  x |>
      mutate(across(.cols = matches(c("Subsection", "Number")) |
starts_with("Suffix"), .fns = \(x) {stringr::str_pad(x, width = 2, pad =
"0")}),
             detector = Subsection,
             detector = ifelse(!is.na(Number), paste(detector , Number,
sep="_"), detector),
             detector = ifelse(!is.na(Suffix), paste(detector , Suffix,
sep="_"), detector),
             detector = ifelse(!is.na(Suffix2), paste(detector , Suffix2,
sep="_"), detector),
             detector = ifelse(!is.na(Suffix3), paste(detector , Suffix3,
sep="_"), detector),
             detector = ifelse(!is.na(Suffix4), paste(detector , Suffix4,
sep="_"), detector)

      ) |>
      pull(detector)

    grep(paste(detector, collapse = "|"), xpts, value = TRUE)
  })

  cat("\n===> NOTE:: ", length(xpts |> unlist()), " xpts left!!")
  .name_suffix_xpt <- paste0(.name_suffix, "_FilteredXpt")
} else {
```

```
  .name_suffix_xpt <- paste0(.name_suffix, "_AllExistingXpt")
}
```

Following code is used to prepare the contents in Display 15:

```
shell_code <- "# Parameter Definition - Example adapt as needed " |>
    append("#@      |name   |type| | |i|o|related to |descript  |default") |>
    append("#@param path2pgm,file,,,y,y,,,") |>
    append(sapply(seq_along(regi_files), function(x) paste0("#@param file", x, ",file, , ,y,y, ,           ,", regi_files[x]) )|> paste(collapse = ",\n")) |>
    append('#') |>
    append('##') |>
    append('### PLEASE ADJUST FOLLOWING PGM LIST AS NEEDED ###') |>
    append('pgms <- c(') |>
    append(paste0( paste(lapply(seq_along(bygroup),
           function(x) {
             c(
               # paste0("## Programmer: ", names(bygroup)[x]),
               if (is.null(.programming_plan)) {
                 if(.byauthor) {
                   paste0("## Programmer: ", names(bygroup)[x])
                 } else{
                   NULL
                 }
               } else if (!is.null(.groupby)){
                 # paste0("## ", .groupby, ": ", unique(x[[.groupby]]))
                 paste0("## ", .groupby, ": ", names(bygroup)[x])
               } else{
                 NULL
               },
               paste0('\"', bygroup[[x]]$files |>
                        gsub(pattern = "/*/xxx/xxx/xxxxxx/xxxxxxxx/xxxxxxxx/", replacement = ""),'\"')
             )
           } ) |> unlist()|> paste(collapse = ",\n")))
    ) |>
    append(")") |>
    append("### PLEASE DO NOT CHANGE THE CODE BELOW ###") |>
    append("##") |>
```

```
    append("#") |>
    append(" options(warn = 1) # output warning immediately, but will
output meaningless warning") |>
    append('source <- function(file, ...               ') |>
    append(', local = nenv') |>
    append('){               ') |>
    append('  org_file <- file                                        ')
|>
    append('  if (!file.exists(file)) {                               ')
|>
    append('    cat("\\n====> ", file, " not found, try *.r\\n")         ')
|>
    append('    file <- gsub("R$", "r", file)                         ')
|>
    append('  }                                                       ')
|>
    append('  if (!file.exists(file)) {                               ')
|>
    append('    warning(org_file, " does not exist !!", call. = FALSE) ')
|>
    append('  } else {                                                ')
|>
    append('    base::source(file, ...                  ') |>
    append('    , local = local                ') |>
    append('    )                ') |>
    append('  }                                                       ')
|>
    append('}                                                         ')
|>
    append('if (!is.null(path2pgm)) if (path2pgm != "") pgms <- path2pgm')
|>
    append('xxxxxxxxxRepositoryRoot <- "/xxx/xxx/xxxxxx/xxxxxxxx/xxxxxxxx"')
|>
    append('for (path2pgm in pgms){') |>
    append('setwd(xxxxxxxxxRepositoryRoot)') |>
    append('exec.programName <- gsub(".+/(.+)\\\\.(R|r)", "\\\\1",
path2pgm)') |>
    append('exec.programPath <- path2pgm |> gsub(pattern =
"/xxx/xxx/xxxxxx/xxxxxxxx/xxxxxxxx/", replacement = "")') |>
    append(paste0('logfile <- file.path(\"', log|>
                    gsub(pattern = "/xxx/xxx/xxxxxx/xxxxxxxx/xxxxxxxx/",
replacement = ""),
```

```
                  "\", paste0(gsub(\".+/(.+)\\\\.(R|r)\", \"\\\\1\",
path2pgm), \"_UserProgram.log\"))")) |>
    append("logfile <- file(logfile, open = 'wt')") |>
    append("nenv <- new.env()") |>


    ## pass global objects to the new environment
    # append("objInGlob <- ls()") |>
    # append('objInGlob <- objInGlob[! objInGlob %in% c("nenv",
"objInGlob")]') |>
    # append("sapply(objInGlob, \\(objInGlob_) assign(objInGlob_,
get(objInGlob_), envir = nenv))") |>


    append('cat("EXECUTING ", path2pgm,"\\n")') |>
    append("sink(logfile)") |>
    append("sink(logfile, type = \"message\")") |>
    append("### IF NECESSARY, USE assign(\"obj_name\", value, envir = nenv)
TO PASS PARAMETERS TO INDIVIDUAL PGM ###") |>
    append(paste0(if (.stop_error) {"tryCatch"} else {"try"},
"(source(path2pgm, echo= TRUE, max.deparse.length = Inf")) |>
    append(", local = nenv") |>
    append("))") |>
    append("sink(type = \"message\")")|>
    append("sink()") |>
    append('rm(list =
ls()[!grepl("^path2pgm$|^pgms$|^xxxxxxxxxRepositoryRoot$|^source$|^file\\\\
d+|^workplaceFolder$|^exec\\\\.|^fmtSearchPath", ls())])') |>
    append("}") |>
    append("cat(\"Batch list has been executed completely.\")")
```

Following code is used to prepare a tree-like list for renderTree:

```
car <- function(l) head(l,n=1)
cdr <- function(l) tail(l,n=-1)
path_split <- function(pth) {
  return(strsplit(pth,split="/")[[1]] |>
           gsub(pattern = "^$", replacement = "/"))
}
tree_builder <- function(tree, vec) {
    ## nothing to add, return l unchanged
    if (length(vec) == 0)
      return(tree)
```

```
    hd <- car(vec)
    tl <- cdr(vec)

    if (is.null(tree[[hd]])) {
      ## first time hd is seen

      if(length(tl)==0) {
        ## add hd and done because there is no tail
        tree[[hd]]  <- list()

      } else {
        ## add hd and the tail recursively
        tree[[hd]] <- Recall(tree[[hd]], tl)
      }

    } else {
      ## hd has already been seen, walk deeper
      tree[[hd]] <- Recall(tree[[hd]], tl)
    }

    return(tree)
}
df <- list.dirs("/xxx/xxx/xxxxx/xxxxx/xxxxx/root")
list_paths <- lapply(df, path_split)
res1 <- Reduce(tree_builder, list_paths, list())
```

CONCLUSION

When there is a large number of programs to batch run based on various purposes and scopes, developing a tool that generates dedicated batch scripts for specific environments can definitely improve efficiency thereafter. This paper demonstrates 2 approaches (Option1: detecting programs in certain folder + pattern filter, manual adding/deleting/reordering operations; Option2: metadata-based) to create batch scripts for Linux and some repositories.

ACKNOWLEDGMENTS

I would like to acknowledge Jose Manuel Cornes (Merck Germany) who helped me on the renderTree part.

CONTACT INFORMATION

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

Wayne Yang
Merck
Wayne.Yang@merckgroup.com

Gloria Zhong
Merck
Gloria.zhong@merckgroup.com

Sukie Gao
Merck
Sukie.Gao@merckgroup.com