PhUSE US Connect 2020

Paper SI04

Creating Simplified TS domain Using R Shiny App

Hanming Tu, Frontage Laboratories, Exton, PA, USA

ABSTRACT

This paper describes how to use R and R shiny design a web-based application to easily generate trial summary (TS) domain dataset to meet the FDA study data conformance requirements.

INTRODUCTION

Based on recent statistics from FDA, many companies or contract research organizations have failed to meet Technical Rejection Criteria (TRC) for Study Data and the Study Data Technical Conformance Guide simply due to missing trial summary (TS) domain dataset. When to include full TS file and when to provide simplified TS file has been clearly defined in the guide but not many companies know how to produce a TS file even a simplified one. Simplified TS Files are SAS Transport files which can be created using free and open-source software, including Python and R.

This paper shows how to use R and R shiny to develop a web-based application to easily generate TS domain dataset to meet the FDA study data conformance requirements. With this web application, sponsors can create Simplified TS Files (ts.xpt) to meet study data submission requirements to the Food & Drug Administration (FDA) Center for Drug Evaluation and Research (CDER) and the Center for Biologics Evaluation and Research (CBER). Simplified TS Files provide a Study Start Date for NDA/BLA/ANDA studies that began on or before December 17, 2016 and Commercial IND studies that began on or before December 17, 2017 and allow FDA to determine that study data is not required to be in a CDISC standardized format.

BUSINESS REQUIREMENTS

When to provide a full trial summary (TS) domain data set and when to just have a simplified TS file have been documented in two FDA guides:

Study Data Technical Conformance Guide[1]:

- Provide guidance on simplified and full ts.xpt in Trial Design Model (TDM)
- Define a list of parameters that should be submitted where relevant for clinical studies for a full ts.xpt

Technical Rejection Criteria for Study Data[2]:

- If a clinical study, submitted to CDER or CBER, started on or prior to December 17, 2016, for NDAs, BLAs, and ANDAs, and the study contains an xpt dataset (other than the ts.xpt), a simplified ts.xpt file should be submitted
- If a nonclinical study, submitted to CDER, started on or prior to December 17, 2016, for NDAs, BLAs, and ANDAs (or December 17, 2017, for Commercial INDs), whether or not the study contains an xpt dataset (other than the ts.xpt), a simplified ts.xpt file should be submitted

We will only discuss the requirements for a simplified trial summary domain file and how to develop a tool to create TS domain files.

[1] Study Data Technical Conformance Guide Technical Specifications Document, March 2019, by U.S. Department of Health and Human Services, Food and Drug Administration: https://www.fda.gov/media/122913/download

[2] Technical Rejection Criteria for Study Data, 10/19/2019, by FDA: https://www.fda.gov/media/100743/download

TECHNICAL REQUIREMENTS

FDA has published a technical recommendation[3] for creating simplified ts.xpt file using either Python or R as a development language. Further detailed are obtained from FDA and presented in the following sections as basic and enhanced technical requirements.

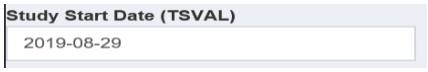
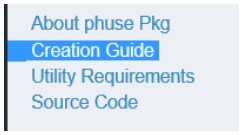
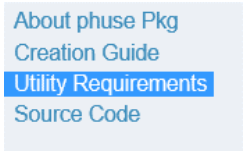
BASIC TECHNICAL REQUIREMENTS

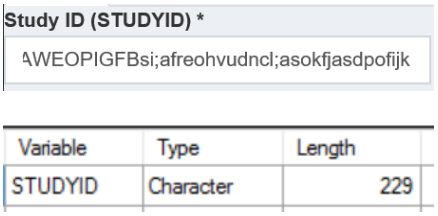
Req No.	Requirements	Acceptance Criteria
R-1	Allow users to enter a Study ID	• Field label: “Study ID (STUDYID)” • Free text field • Maximum 200 characters • Format variable value as character • Field should be independent from R-2 field
R-2	Allow users to select a Study Type	• Field label: “Study Type Parameter (TSPARMCD)” • Dropdown Selections: o "Clinical (SSTDTC)" o "Nonclinical (STSTDTC)" • Format variable value as character
R-3	Allow users to select a Study Start Date	• Field label: “Study Start Date (TSVAL)” • Include validation to format date and variable value as *YYYY-MM-DD* • Format variable value as character • R-3 and R-4 are mutually exclusive i.e. If R-3 is filled, then R-4 should be blank/disabled
R-4	Allow users to select an Exception Code	• Field label: “Exception Code (TSVALNF)” • Free text field • Format variable value as character • R-4 and R-3 are mutually exclusive i.e. If R-4 is filled, then R-3 should be blank/disabled
R-5	Allow users to view variable values	• StudyID Variable Value is what the user entered in the Study ID Field (R-1) • TSPARMCD Variable Value is what the user selected in the Study Type Field, SSTDTC or STSTDTC (R-2) • TSVAL Variable Value is what the user selected in the Study Start Date Field (R-3) • TSVALNF Variable Value is what the user selected in the Exception Code Field (R-4) • Variable values are visually associated with their variable names and field labels • Variable values are formatted as they will appear in the XPT file
R-6	Allow users to download and generate an XPT file containing the four variables entered the fields	• Button labeled: “Download” • *Nonfunctional requirements for generated XPT files included in worksheet 2*
R-7	Generated XPT file is titled, "ts.xpt"	N/A
R-8	Generated XPT file is in SAS Transport Format Version 5	N/A
R-9	Generated XPT file contains one dataset with the SAS name, "TS"	N/A

[3] Creating Simplified TS.XPT Files, November 2019, By FDA: https://www.fda.gov/media/132457/download.

R-10	Generated XPT file contains four variables in one row of information: STUDYID, TSPARMCD, TSVAL, TSVALNF	• StudyID Variable Value is what the user entered in the Study ID Field (R-1) • TSPARMCD Variable Value is what the user selected in the Study Type Field, SSTDTC or STSTDTC (R-2) • TSVAL Variable Value is what the user selected in the Study Start Date Field in the format of YYYY-MM-YY (R-3) • TSVALNF Variable Value is what the user entered in the Exception Code Field (R-4)
R-11	Generated XPT file contains all variable values formatted as characters	N/A

ENHENCED TECHNICAL REQUIREMENTS

Defects #/ Enhancement #	Defects / Enhancements	Screenshots
E-1	Study Start Date (TSVAL) and Exception Code (TSVALNF) should not be mutually exclusive. -User can add values to either fields without any inter- dependency	N/A
E-2	Default value in Study Start Date (TSVAL) field should be blank, not today's date.	
E-3	“Exception Code (TSVALNF)” should be a Dropdown with only one value "NA" Note - Currently it is a free text field	
E-4	In View tab, only shows one entry at a time. Remove show entry functionality.	
E-5	In View tab, remove search function	
E-6	In View tab, remove use of paging functionality	
E-7	Edit the link text from "Creation Guide" to "Simplified ts.xpt Creation Guide" and point to the Study Data for Submission to CDER and CBER Page URL - "https://www.fda.gov/industry/study-data-standards-resources/study-data-submission-cder-and-cber"	
E-8	Remove the Link "Utility Requirements"	

D-1	"Study ID (STUDYID)" field should have a character limit of 200	

TECHNICAL DESIGN

Since one of the authors has developed a R package and is more familiar with R language, so we use R language as a language to develop the tool. We initially developed the web application in R and embedded it in the phuse package as one of examples[4], and the latest version is published to the CRAN as well[5]. In order to make it easier to manage, the web application has been extracted out and a separate package called genTS is created for it[6]. The technical design and considerations are discussed in the following sections.

R AND R PROJECT

R is a free and open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. it compiles and runs on a wide variety of Linux, Windows and MacOS platforms.

The fundamental unit of shareable code in R is package. A **package** bundles together code, data, tests, examples, and documentation, and is easy to share with others. Packages are in a well-defined format and stored in Comprehensive R Archive Network (CRAN). The directory where packages are stored is called the library. R comes with a standard set of packages.

The integrated development environment is provided through **RStudio**. RStudio has built-in support for you to divide your work into multiple contexts. In RStudio, you can create a project to manage your work and your code. Each **RStudio project** has its own working directory, workspace, history, and source documents. The steps for creating a project in RStudio are documented in the paper presented in PHUSE conference in 2017[7].

R PACKAGES USED

R provides many shared packages and helps with web development. When developing the web application for simplified TS domain file, we find that we need to use a few packages. We have gathered the description about those packages from the Internet and will not include the references here.

- Package **shiny**: is an R package that makes it easy to build interactive web apps straight from R. You can also use Shiny server to set up a web server.
- Package **shinydashboard**: provides an easy way to create dashboard.
- Package **shinyjs**: lets you perform common useful JavaScript operations in Shiny apps that will greatly improve your apps without having to know any JavaScript
- Package **shinyBS**: Twitter Bootstrap Components for Shiny to add additional functionality and interactivity to your Shiny applications

[4] The phuse package on GitHub: https://github.com/TuCai/phuse

[5] The phuse package in CRAN: https://CRAN.R-project.org/package=phuse

[6] The genTS package on GitHub: https://github.com/TuCai/genTS

[7] Defining Script Metadata for Sharing: Using phuse R package as an example 2017 by Hanming Tu: https://www.phusewiki.org/docs/Conference%202017%20CT%20papers/CT12.pdf

- Package **SASxport**: to read and write 'SAS' 'XPORT' Files It includes functions for reading, listing the contents of, and writing 'SAS' 'xport' format files.
- Package **Hmisc**: contains many functions useful for data analysis, high-level graphics, utility operations, functions for computing sample size and power, importing and annotating datasets, imputing missing values, advanced table making, variable clustering, character string manipulation, conversion of R objects to LaTeX and html code, and recoding variables.
- Package **rhandsontable**: is a htmlwidget based on the handsontable.js library. Handsontable is a data grid component with an Excel-like appearance. Built in JavaScript, it integrates with any data source with peak efficiency. It comes with powerful features like data validation, sorting, grouping, data binding, formula support or column ordering.
- Package **phuse**: is a web application framework for phuse-scripts repository. It provides some useful functions that allow you to search and clone a repository.
- Package **DT**: provides an R interface to the JavaScript library DataTables. R data objects (matrices or data frames) can be displayed as tables on HTML pages, and DataTables provides filtering, pagination, sorting, and many other features in the tables.
- Package **V8**: is an R interface to V8: Google's open source JavaScript and WebAssembly engine. This package can be compiled either with V8 version 6 and up, a NodeJS shared library, or the legacy 3.14/3.15 branch of V8.
- Package **stringr**: is simple, consistent wrappers for common string operations There are four main families of functions in stringr:
    - Character manipulation: these functions allow you to manipulate individual characters within the strings in character vectors.
    - Whitespace tools to add, remove, and manipulate whitespace.
    - Locale sensitive operations whose operations will vary from locale to locale.
    - Pattern matching functions. These recognise four engines of pattern description. The most common is regular expressions, but there are three other tools.

DESIGN USER INTERFACE

A web-based R application usually has three components: user interface, server and run-time.

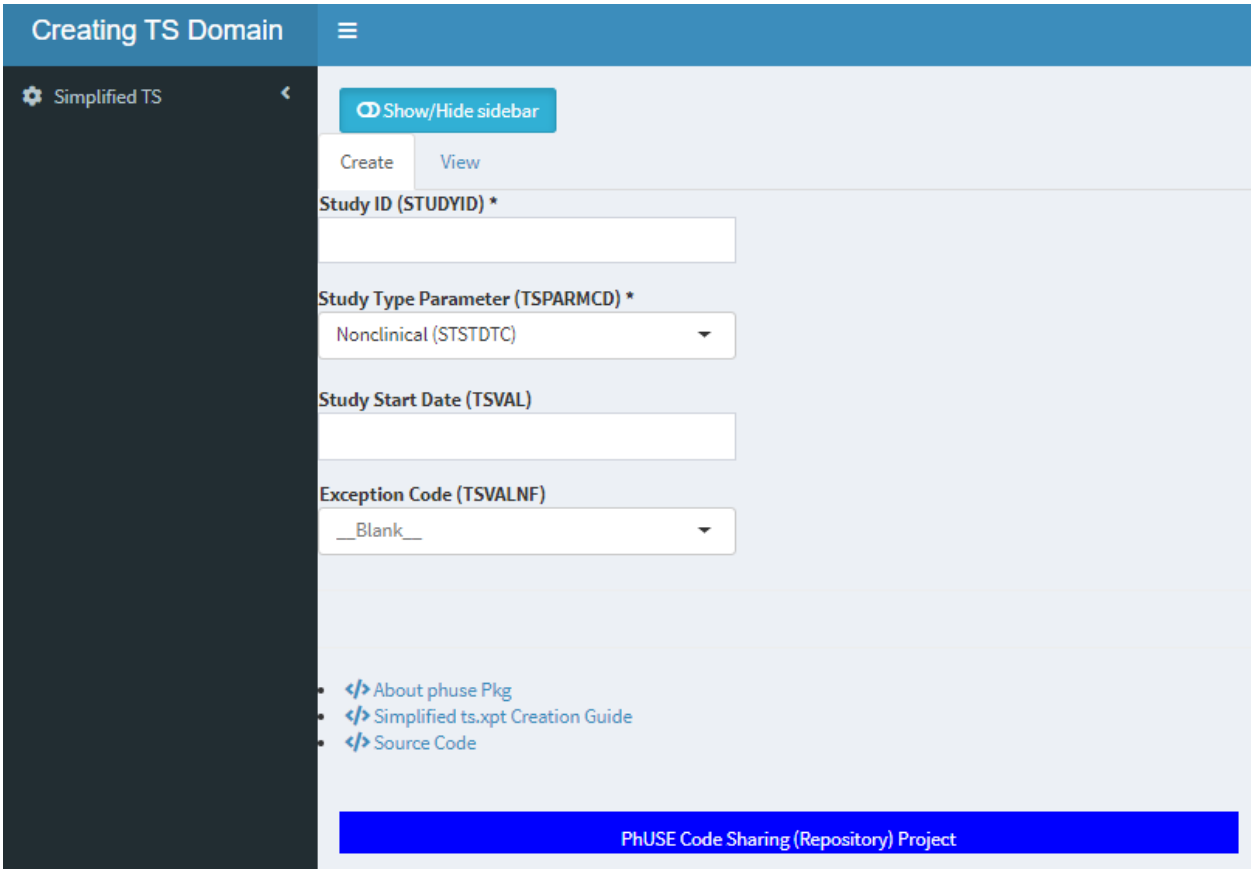
```
ui <- dashboardPage( … )

server <- function(input, output, session) { .. }

shinyApp(ui, server)
```

There are many ways to define user interface. We choose a simple way – using one of many packages available in R to do it. The shinydashboard package provides an easy and simple way of laying out the header, sidebar and page of a dashboard.

```
header <- dashboardHeader(
  title = "Creating TS Domain"
)

sidebar <- dashboardSidebar(
  sidebarMenu(
    # Setting id makes input$tabs give the tabName of currently-selected tab
    id = "tab1",
    menuItem("Simplified TS", icon = icon("cog"),
      menuSubItem("Simplified ts.xpt Creation Guide", href = '…', newtab = FALSE)
    , menuSubItem("About phuse Package", href = '..', newtab = TRUE)
    , menuSubItem('Source Code',href='.. ', newtab = TRUE)

    )
  )
)

ui <- dashboardPage(
  header,
  sidebar,
  dashboardBody(
```

```
    useShinyjs()
    , extendShinyjs(text = 'shinyjs.hideSidebar = function(params) {
      $("body").addClass("sidebar-collapse");
      $(window).trigger("resize"); }')
    , extendShinyjs(text='shinyjs.showSidebar = function(params) {
      $("body").removeClass("sidebar-collapse");
      $(window).trigger("resize"); }')
    , bsButton("showpanel", "Show/Hide sidebar",icon = icon("toggle-off"),
               type = "toggle",style = "info", value = TRUE)
    , tags$head(
      tags$style(type="text/css"
        ,"label{ display: table-cell; text-align: right; vertical-align: middle; }
         .form-group { display: table-col;}")
    )
    , fluidRow(tabsetPanel(id='tabs'
                           , tabPanel("Create", uiOutput("tabP1"))
                           , tabPanel("View", uiOutput("tabP2"))
    ))
    , fluidRow(
      tabItem("tab1", hr()
              , menuItem('About phuse Pkg',icon=icon('code'),href='…')
              , menuItem('Simplified ts.xpt Creation Guide',icon=icon('code'),href='…')
              , menuItem('Source Code',icon=icon('code'),href='…')
              , hr()
      )
    )
    , tags$footer("PhUSE Code Sharing (Repository) Project"
                  , align = "center"
                  , style = "position:dynamic;
              bottom:0;
              width:100%;
              height:30px;   /* Height of the footer */
              color: white;
              padding: 10px;
              background-color: blue;
              z-index: 1000;"
    )
  )
)
```

Here is how the user interface looks like:



Here are the codes to define the two panels on the main page:

```
# -------------------- 1 tabPanel: Create  -------------------------------
  output$tabP1 <- renderUI({
    tabPanel("Create"
      , div(id = "form"
          , textInput("studyid", "Study ID (STUDYID) *" )
          , bsAlert("alert")
          , selectInput("tsparmcd", "Study Type Parameter (TSPARMCD) *"
             , choices = list("Clinical (SSTDTC)" = "SSTDTC"
                 , "Nonclinical (STSTDTC)" = "STSTDTC")
                 , selected = "STSTDTC")
             , dateInput("tsval", value = NA, label = "Study Start Date (TSVAL)"
                 , format = "yyyy-mm-dd" )
             , selectInput("tsvalnf", "Exception Code (TSVALNF)"
                 , choices = list("NA" = "NA", "__Blank__" = ""), selected = "")
          )
      , hr()
      , hidden(downloadButton('downloadData', 'Download'))
    )
  })

  # -------------------- 2 tabPanel: View  ---------------------------------
  output$DT2 <- renderDataTable({
    df <- ts_content();  str(df)
    datatable(df, options = list(dom = 't'));  # from DT package
  })
  output$tabP2 <- renderUI({
    tabPanel("View"
             , DT::dataTableOutput("DT2")
             , hr()
             , hidden(downloadButton('downloadData', 'Download'))
    )
  })
```

COLLECT CONTENT

Sponsors should submit a dataset named 'ts.xpt' with four variables (STUDYID, TSPARMCD, TSVAL, and TSVALNF) and one row of information in it. Here are the examples of TS.xpt files in clinical and nonclinical studies:

- For clinical studies:

For a study with a valid SSD:

STUDYID	TSPARMCD	TSVAL	TSVALNF
study ID in STF	SSTDTC	yyyy-mm-dd	

For a study without a valid SSD:

STUDYID	TSPARMCD	TSVAL	TSVALNF
study ID in STF	SSTDTC		As specified in the ISO 21090 Standard

- For nonclinical studies:

For a study with a valid SSD:

STUDYID	TSPARMCD	TSVAL	TSVALNF
study ID in STF	STSTDTC	yyyy-mm-dd	

For a study without a valid SSD:

STUDYID	TSPARMCD	TSVAL	TSVALNF
study ID in STF	STSTDTC		As specified in the ISO 21090 Standard

- STUDYID is a unique identifier for a study within the submission in the Study Tagging File (STF).
- TSPARMCD is trial summary parameter short name. The companion to TSPARM, TSPARMCD is limited to 8 characters and does not have special character restrictions. These values should be short for ease of use in programming, but it is not expected that TSPARMCD will need to serve as variable names. The value of TSPARMCD in a simplified TS is "SSTDTC" for clinic studies and "STSTDTC" for nonclinical studies. These two parameters are for study start date. Is there a standard in the Industry of how to determine the study start date for clinical studies? Is it the protocol finalized date, first subject in date or first initiation date? We gather the definition from PHUSE[8] and FDA[9] and summarize as the below.
    - SSTDTC is study start date in ISO 8601 format, and it should be the earliest date of informed consent among any subject that enrolled in the clinical study.
    - STSTDTC is study start date for nonclinical studies (SEND). It is the date on which the study protocol or plan is approved (signed) by the Study Director, also known as the study initiation date.
- TSVAL is parameter value.
- TSVALNF is parameter null flavor. Null flavor for the value of TSPARM, to be populated if and only if

[8] PHUSE Data Submissions FAQ team, 2018, https://www.phusewiki.org/wiki/index.php?title=SDTM_FAQ_Team_Responses

[9] SDTM Trial Summary Domain: Putting Together the TS Puzzle, PharmaSUG 2016 - Paper DS11, by Kristin Kelly, Jerry Salyers, and Fred Wood: https://www.pharmasug.org/proceedings/2016/DS/PharmaSUG-2016-DS11.pdf

TSVAL is null.

We developed two functions to store the variables for the content of the TS domain table:

```
  get_tsval <- reactive({
    if (is_empty(input$tsval)) {
      tsval <-  as.character("")
    } else {
      tsval <- as.character(input$tsval)
    }
    tsval
  })

  ts_content <- reactive({
    validate(
      need(input$studyid != "", "Please provide Study ID.")
    )
    validate(
      need(input$tsparmcd != "", "Please provide Study Type Parameter.")
    )
    req(input$studyid)
    req(input$tsparmcd)
    ts <-data.frame(STUDYID=input$studyid
                    , TSPARMCD=input$tsparmcd
                    , TSVAL=get_tsval()
                    , TSVALNF=input$tsvalnf
    );
    ts
  })
```

CONTROL INPUTS

R provides three functions to interact with user's inputs: reactive, observe and observEvent.

- Function **reactive**: The reactive function allows a user to monitor the status of an input or other changing variable and return the value to be used elsewhere in the code. The monitoring of a reactive variable is considered lazy. "Reactive expressions use lazy evaluation; that is, when their dependencies change, they don't re-execute right away but rather wait until they are called by someone else.
- Function **observe**: is similar reactive, the main difference is that it does not return any values to any other environment besides its own, and it is not lazy. The observe function continually monitors any changes in all reactive values within its environment and runs the code in its environment when these values are changed.
- Function **observeEvent**: is an event trigger. It watches one event or change in a variable, and then fires when the event happens. It is commonly used for watching input to buttons, which is a defined event in which things will happen after the button is pushed. The main difference between observeEvent and observe being the trigger, as observe runs anytime anything changes, and observeEvent waits for the trigger. Note that this environment is like observe in that it does not return non-reactive variables.

Beside the reactive function used for getting inputs, here are the observe and observeEvent further checking and reacting to user inputs:

```
# toggle the sidebar
observe({
    if(input$showpanel == TRUE) { js$showSidebar() } else { js$hideSidebar() }
  })

# whether required fields are filled for downloading
observeEvent(input$studyid, {
    if (input$studyid == "")
      hide("downloadData")
    else
      show("downloadData")
  })

# limit study id to 200 characters
# 11/04/2019: For FDA D-1 in PhUSE Utility feedback v4.0
```

```
observeEvent(input$studyid, {
    if(input$studyid != "") {
      if (str_length(input$studyid)>200) {
        updateTextInput(session, "studyid", value = str_sub(input$studyid, end=200) )
        createAlert(session, "alert", "StudyIDAlert", title = "StudyID Alert"
          , content = "Study ID must not exceed 200 characters and will be truncated."
          , dismiss = TRUE, style = "warning", append = FALSE)
        Sys.sleep(3)
      } else {
        closeAlert(session, "StudyIDAlert")
      }
    }
  })
```

DOWNLOAD TS FILE

The file format for the download is SAS transport file (XPT) so we use write.xport function from the SASxport package then use R downloadHandler to allow user to download and save the file on his or her local computer. The downloaderHandler allows content from the Shiny application to be made available to the user as file downloads. Both filename and contents can be calculated dynamically at the time the user initiates the download. Assign the return value to the downloadData on output in the sever function and make “Download” available to the user on the “Create” tab in the UI.

```
output$downloadData <- downloadHandler(
    filename = function() { paste("ts", ".xpt",sep="") },
    content  = function(file = filename) {
      str(file)
      # get dataframe with the data
      studyData <- ts_content()
      # set to characters not factors
      studyData$STUDYID   <- as.character(studyData$STUDYID)
      studyData$TSPARMCD  <- as.character(studyData$TSPARMCD)
      studyData$TSVAL     <- as.character(studyData$TSVAL)
      studyData$TSVALNF   <- as.character(studyData$TSVALNF)
      # Set length for character fields
      # Label list
      Hmisc::label(studyData$STUDYID) <- "Study Identifier"
      Hmisc::label(studyData$TSPARMCD)<- "Trial Summary Parameter Short Name"
      Hmisc::label(studyData$TSVAL)   <- "Parameter Value"
      Hmisc::label(studyData$TSVALNF) <- "Parameter Null FLavor"
      # place this dataset into a list with a name
      aList = list(studyData)
      # name it
      names(aList)[1]<-"TS"
      # and label it
      attr(aList,"label") <- "TRIAL SUMMARY"
      # write out dataframe
      write.xport(
        list=aList,
        file = file,
        verbose=FALSE,
        sasVer="7.00",
        osType="R 3.5.2",
        cDate=Sys.time(),
        formats=NULL,
        autogen.formats=TRUE
      )
    }
  )
```

CONCLUSION

The tool for creating simplified trial summary file can be developed quickly and made available to all due to the four reasons: 1) The tool embedded in the phuse package is developed under the Code Sharing (Repository) project in Standard Analysis and Code Sharing working group in PHUSE and hosted in GitHub repository and accepted by Comprehensive R Archive Network (CRAN). 2) GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. 3) The CRAN is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries. 4) R is released under the GNU General Public License (GPL), version 2. The license must not restrict anyone from making use of the program in a specific field of endeavor.

CONTACT INFORMATION

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

Author Name: Hanming Tu
Company: Frontage Lab
Address: 700 Pennsylvania Drive
City / Postcode: Exton, PA 19341
Work Phone: 484-202-6479
Fax: 610-232-0101
Email: htu@frontagelab.com
Web: http://www.frontagelab.com