PharmaSUG 2026 - Paper TT-106

Automating BIMO Subject-level Data Line Listings with R

Weishan Song*, Vertex Pharmaceuticals Inc.;
Wanting Jin*, University of North Carolina at Chapel Hill;
Weiyu Zhou, Vertex Pharmaceuticals Inc.;
Margaret Huang, Vertex Pharmaceuticals Inc.

ABSTRACT

The U.S. Food and Drug Administration (FDA) requires a BIMO package for pivotal studies to support clinical site inspections. A key component is the subject-level data line listings by site, which provide site-specific safety and efficacy subject data. These listings are often produced by manual programming, requiring programmers to program and validate each individual output from SDTM/ADaM datasets. In large submission packages containing multiple pivotal studies and hundreds of BIMO listings, this approach becomes not only labor-intensive but also introduces the risk of variability in presentation and potential misalignment with the clinical study report (CSR). While the same data elements have already been validated and presented in CSR listings, this duplication of effort represents a significant operational burden that increases with submission complexity.

We developed an R tool that automates the generation of these subject-level listings by site using an outputs-to-outputs approach (creating BIMO listings directly from corresponding CSR listings). The tool ingests the finalized CSR listings, extracts content, page layout, and formatting and uses this information to reproduce the required *site-specific* BIMO listings without reprogramming. This 1- significantly improves efficiency, 2- standardizes presentation of the BIMO listings and 3- enhances accuracy and simplifies quality control.

INTRODUCTION

For pivotal studies supporting regulatory submissions, the BIMO package is a required component of the FDA review process. The package includes subject-level data line listings by site, a summary-level clinical site dataset, a define file for the site dataset, and optionally a BIMO Data Reviewer's Guide. Within the BIMO package, subject-level line listings are operationally critical because they are used by field investigators during site inspections and source data verification.

As shown in Figure 1, the site-level listings typically include ten subject-level categories: consented subjects, treatment assignment, discontinuations, study population, inclusion and exclusion criteria, adverse events, protocol deviations, efficacy endpoints, concomitant medications, and safety monitoring. The challenge is not simply producing these categories once, but generating them repeatedly in a site-oriented structure.

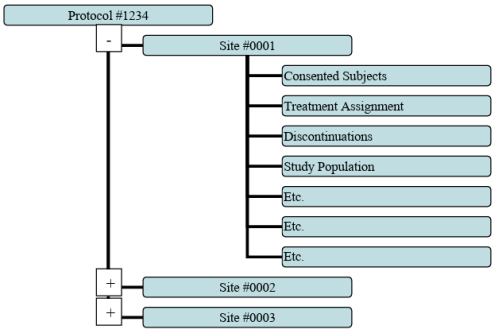


Figure 1 Structure of Subject-Level Data Line Listings By Site, By Listing

* These authors contributed equally to this work.

In large submission packages containing multiple pivotal studies, with each conducted across dozens of investigation sites, the total number of BIMO listings can easily reach hundreds of individual outputs. In many programming environments, the required content has already been generated for the clinical study report as formatted listings. These outputs contain the data needed, but they are not directly reusable as BIMO deliverables because the document structure, sectioning, and site-level partitioning differ from the CSR presentation model. As a result, teams often duplicate effort by reconstructing already validated content in a separate workflow. Consequently, automation and standardization of BIMO listings generation are essential to ensure efficiency, consistency, and scalability.

Here we present an R-based automation tool that addresses the duplication by treating existing RTF listings as machine-readable production assets. Instead of rebuilding each listing from raw analysis datasets, the workflow parses completed CSR outputs into structured objects, applies site-level transformation rules, and writes standardized RTF documents for BIMO submission.

In the following sections, we first present an overview of the automation workflow. We then describe the three core functions in detail and illustrate the application using an example study that has received FDA approval. Finally, we conclude by discussing the benefits of the approach, its broader applications, and directions for future work.

AUTOMATION WORKFLOW

To demonstrate the functionality of the R tool, we provide an overview of the automation workflow (see Figure 2). The workflow begins with established CSR listings in RTF format as input and returns submission-ready subject-level data line listings as output.

The first step (the first arrow in Figure 2) is to parse each listing into structured R data frames that preserve the document information needed for reconstruction, including title text, footnotes, headers, spanning headers, body cells, and optional column-width metadata. Once these intermediate R data frames have been created, the second step (the second arrow in Figure 2) is to partition the listing content by site and write it iteratively using a standard formatting specification.

This workflow reduces study-specific reprogramming, supports deterministic layout across sites, and enables batch production of outputs while preserving the visual conventions of the originating report listings.

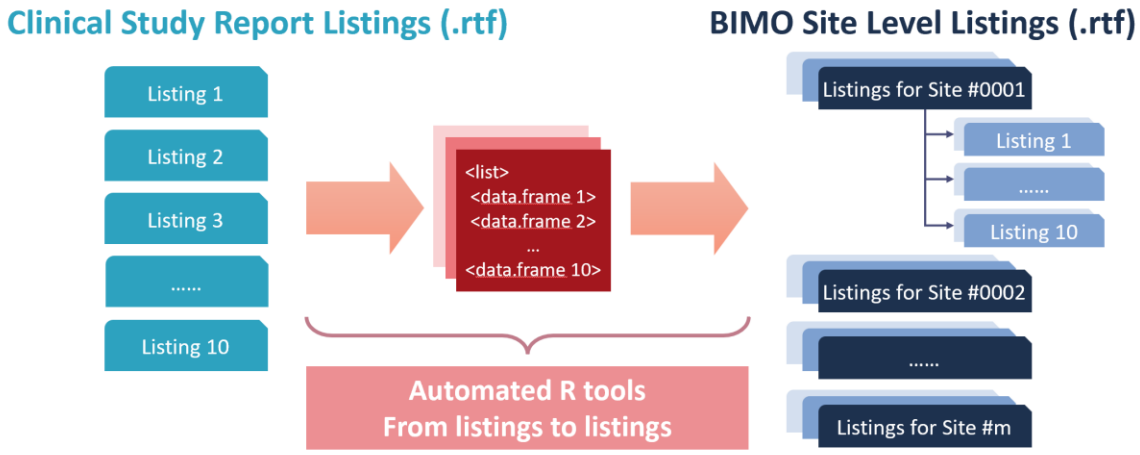


Figure 2 Subject-Level Data Line Listings Automation Workflow Overview

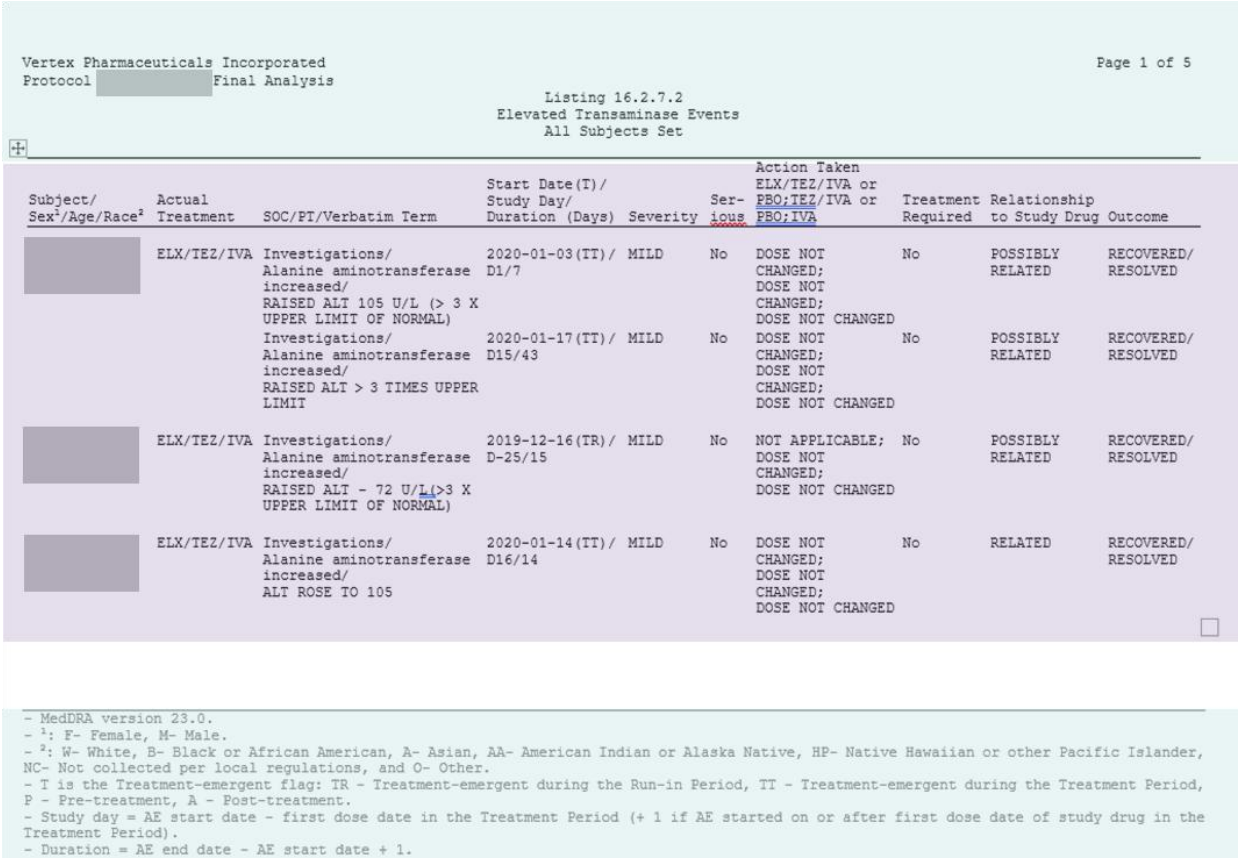
CORE FUNCTIONS IMPLEMENTATION

The workflow is realized with three core R functions: *r_tfl_rtfread()* for the first step, *r_tfl_rtfwrite()* for the second step, and generate_bimo_listings() as a study-level driver that integrates the first two functions. In the following subsections, we present the technical details of all three functions.

R_TFL_RTFREAD(): READING RTF INTO R

The r_tfl_rtfread() function converts an RTF table or listing into structured R data frames that allow downstream code to manipulate listing content without losing the structural elements required for document reconstruction. This function takes two arguments: a required RTF file path argument rtffile and an optional argument DisplayColWidths, that controls whether column-width information is captured. It returns a list with two data frames. The first, tfl, stores listing structure as variables for Site ID, subject ID, column headers, spanning headers, body cells, and, when requested, header and group widths. The second, tf, stores titles and footnotes with sequential identifiers and type labels.

Figure 3 shows an example of an RTF listing used as input to r_tfl_rtfread(). Figure 4 and Figure 5 show the resulting output objects. The function parses the purple-shaded region in Figure 3 into the tfl data frame shown in Figure 4 and the blue-shaded region into the tf data frame shown in Figure 5.



Vertex Pharmaceuticals Incorporated
Protocol [redacted] Final Analysis

Page 1 of 5

Listing 16.2.7.2
Elevated Transaminase Events
All Subjects Set

Subject/ Sex[2]/Age/Race[2]	Actual Treatment	SOC/PT/Verbatim Term	Start Date(T)/ Study Day/ Duration (Days)	Severity	Ser- ious	Action Taken ELX/TEZ/IVA or PBO:TEZ/IVA or PBO:IVA	Treatment Required	Relationship to Study Drug	Outcome
[redacted]	ELX/TEZ/IVA	Investigations/ Alanine aminotransferase increased/ RAISED ALT 105 U/L (> 3 X UPPER LIMIT OF NORMAL)	2020-01-03(TT)/ D1/7	MILD	No	DOSE NOT CHANGED; DOSE NOT CHANGED; DOSE NOT CHANGED	No	POSSIBLY RELATED	RECOVERED/ RESOLVED
		Investigations/ Alanine aminotransferase increased/ RAISED ALT > 3 TIMES UPPER LIMIT	2020-01-17(TT)/ D15/43	MILD	No	DOSE NOT CHANGED; DOSE NOT CHANGED; DOSE NOT CHANGED	No	POSSIBLY RELATED	RECOVERED/ RESOLVED
[redacted]	ELX/TEZ/IVA	Investigations/ Alanine aminotransferase increased/ RAISED ALT - 72 U/L(>3 X UPPER LIMIT OF NORMAL)	2019-12-16(TR)/ D-25/15	MILD	No	NOT APPLICABLE; DOSE NOT CHANGED; DOSE NOT CHANGED	No	POSSIBLY RELATED	RECOVERED/ RESOLVED
[redacted]	ELX/TEZ/IVA	Investigations/ Alanine aminotransferase increased/ ALT ROSE TO 105	2020-01-14(TT)/ D16/14	MILD	No	DOSE NOT CHANGED; DOSE NOT CHANGED; DOSE NOT CHANGED	No	RELATED	RECOVERED/ RESOLVED

- MedDRA version 23.0.
- [1]: F- Female, M- Male.
- [2]: W- White, B- Black or African American, A- Asian, AA- American Indian or Alaska Native, HP- Native Hawaiian or other Pacific Islander, NC- Not collected per local regulations, and O- Other.
- T is the Treatment-emergent flag: TR - Treatment-emergent during the Run-in Period, TT - Treatment-emergent during the Treatment Period, P - Pre-treatment, A - Post-treatment.
- Study day = AE start date - first dose date in the Treatment Period (+ 1 if AE started on or after first dose date of study drug in the Treatment Period).
- Duration = AE end date - AE start date + 1.

Figure 3 Example of RTF Input

h1	h2	h3	tf_id	page	siteid	subjectid	c1	c2	c3
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	1	407			ELX/TEZ/IVA	Investigations/ Alanine
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	1	407				Investigations/ Alanine
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	1	407			ELX/TEZ/IVA	Investigations/ Alanine
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	1	411			ELX/TEZ/IVA	Investigations/ Alanine
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	2	411			ELX/TEZ/IVA	Investigations/ Asparta
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	2	416			TEZ/IVA	Investigations/ Alanine
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	2	416				Investigations/ Asparta
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	2	705			TEZ/IVA	Investigations/ Alanine
Subject/ Sex1/Age/Race2	Actual Treatment	SOC/PT/Verbatim Term	1	2	705				Investigations/ Asparta

Figure 4 Example of r_tfl_rtfread() Output TFL Data Frame

value	type	tf_id
Listing 16.2.7.2 Elevated Transaminase Events All Subjects Set	title	1
- MedDRA version 23.0.	footnote	1
- 1: F- Female, M- Male.	footnote	1
- 2: W- White, B- Black or African American, A- Asian, AA- A...	footnote	1
- T is the Treatment-emergent flag: TR - Treatment-emergen...	footnote	1
- Study day = AE start date - first dose date in the Treatmen...	footnote	1
- Duration = AE end date - AE start date + 1.	footnote	1

Figure 5 Example of r_tfl_rtfread() Output TL Data Frame

Implementation of the function can be divided into four main parts: loading the RTF file, extracting body cells, extracting column widths, and extracting titles and footnotes.

During RTF file loading, the `r_tfl_rtfread()` function collapses the file contents into a single text string, classifies the texts into title, header, cell, and footnote. These classifications drive the rest of the parsing logic.

The core RTF loading implementation is displayed below.

```
r_tfl_rtfread <- function(rtffile, DisplayColWidths = FALSE){

  ## Read plain-text representation
  line <- read_rtf(rtffile, row_start = "")
  rtf_file <- as.data.frame(line) %>%
    separate_longer_delim(line, delim = "\n\n") %>%
    mutate(part = "", page = NA, tf_id = NA)

  ## Load in raw rtf files
  rtf_content <- readLines(rtffile, encoding = "UTF-8")
  rtf_text <- paste(rtf_content, collapse = " ")
  rtf_text <- iconv(rtf_text, from = 'ISO-8859-1', to = "UTF-8")
  )

  for(i in 1:nrow(rtf_file)){
    ## Identify Title lines
    if(str_detect(rtf_file$line[i],"Vertex Pharmaceuticals|^Listing") & part==''){
      titlesplits <- trimws(unlist(str_split(rtf_file$line[i], "\n")))
      titlesplits <- titlesplits[!grepl("Vertex Pharmaceuticals|^Protocol|Vertex
Program", titlesplits)]
      title_cleaned <- paste0(titlesplits, collapse = "\n")
      rtf_file$line[i] <- title_cleaned
      page <- page+1
```

```
      if(identical(title,title_cleaned)){ ## Check if new tables
        part <- ''
      }else{
        part <- 'title'
        title <- title_cleaned
        tf_id <- tf_id + 1
        newtf <- 1
      }
    }

    ## Identify Table cell and headers
    if(grepl("\\s\\|\\s", rtf_file$line[i])){
      if(part %in% c('title','') | grepl("Subject\\/", rtf_file$line[i])) part <-
'header0'
      else part <- 'cell'
      if(grepl("Subject\\/|^Subject", rtf_file$line[i])) part <- 'header'
    }

    ## Identify footnote
    if(grepl("^\\-", rtf_file$line[i])) part <- 'footnote'

    # Identify blank lines
    if(trimws(rtf_file$line[i]) == "") {
      part <- ''
      next
    }

    rtf_file$part[i] <- part
    rtf_file$page[i] <- page
    rtf_file$tf_id[i] <- tf_id
  }
```

Program 1. Loading RTF file in r_tfl_rtfread()

After line classification, the function extracts body cells by selecting records marked as cell. The function splits delimited row text into column variables `c1` through `cn`., parses header rows into `h1` through `hn`, and parses spanning headers into `g1` through `gm`. All of these variables are stored in the returned `TFL` object. If no body cells are present, the function still creates an empty cell structure based on the detected headers so that subsequent write logic can operate consistently.

The implementation of body cells extraction is displayed below.

```
cells <- rtf_file %>%
  filter(part == "cell" & line != " | ") %>%
  mutate(
    line = str_replace(line, " \\| $", ""),
    Subject = str_match(line, "^(\\d+)-(\\d+)-(\\d+)"),
    siteid = Subject[,3],
    subjectid = Subject[,4],
    c = line
  ) %>%
  select(-c(Subject, line, part)) %>%
  fill(siteid, subjectid, .direction = "down") %>%
  separate_wider_delim(c, delim = " | ", names_sep = "")
```

Program 2. Body-cells Extraction in r_tfl_rtfread()

In the third step, if the `DisplayColWidths` argument is set to “TURE”, the function also calls the built-in function `r_tfl_rtfwidth()` to extract column boundaries from the RTF header blocks. The resulting widths are stored as `hw1` through `hwn` and `gw1` through `gwn` variables in the `TFL` data frame. This information can later be reused to preserve relative layout in regenerated output.

The implementation of column widths extraction is displayed below.

```
if (DisplayColWidths) {
  col_widths <- r_tfl_rtfwidth(rtf_text)

  headers_w <- unlist(
    col_widths[sapply(col_widths, length) == ncol(headers) - 1]
  )
  headers_w <- as.data.frame(t(headers_w))
  names(headers_w) <- paste0("hw", seq_along(headers_w))
  headers <- cbind(headers, headers_w)

  if (nrow(g_headers) != 0) {
    g_headers_w <- unlist(
      col_widths[sapply(col_widths, length) == ncol(g_headers) - 1]
    )
    g_headers_w <- as.data.frame(t(g_headers_w))
    names(g_headers_w) <- paste0("gw", seq_along(g_headers_w))
    g_headers <- cbind(g_headers, g_headers_w)
  }
}
```

Program 3. Column-width Extraction

Finally, the `tf` data frame is created to store titles and footnotes. Titles are extracted from title segment identified during classification while footnotes are extracted from the footer blocks in the raw RTF text.

The final return objects, `TFL` and `TL`, therefore contain both the scientific content of the listing and the metadata needed to reconstruct it as a formatted document. The extracted content is standardized so that grouped columns such as subject and site identifiers remain available for downstream filtering even when repeated values are visually suppressed in the source listing.

R_TFL_RTFWRITE(): WRITING STRUCTURED CONTENT BACK TO RTF

The `r_tfl_rtfwrite()` function accepts the structured listing content returned by the function `r_tfl_rtfread()`. and rebuilds page geometry, title blocks, multi-level headers, body cells, and footer content. As a result, it can render an RTF that is identical to the one ingested by `r_tfl_rtfread()`. The function accepts six arguments: the listing data frame, `tfl`; the title-footnote data frame, `tf`; the number of grouping columns, `GroupCols`; an optional page heading, `heading`; an optional title note, `titlenote`; and the output RTF path, `rtfout`. The function output will be generated as RTF file in the specified RTF path.

The implementation of the function can be divided into three main parts: extracting of each output components, no-data handling and spacing, and RTF output construction.

The first step is to decompose the input into each output component: body cells, main headers, grouped headers, width variables, titles, and footnotes so that each component can be written with its own formatting logic. The core setup is shown below.

```
r_tfl_rtfwrite <- function(tfl, tf, GroupCols,
                           heading = "Vertex Pharmaceuticals Incorporated\n",
                           titlenote = "",
                           rtfout){

  cells <- tfl %>%
    select(matches("^c\d+"))

  headers <- tfl %>%
    select(matches("^h\d+")) %>%
    distinct()

  g_headers <- tfl %>%
    select(matches("^g\d+")) %>%
```

```
    distinct()

  headers_w <- tfl %>%
    select(matches("^hw\d+")) %>%
    distinct()

  title <- tf %>% filter(type == "title") %>% distinct() %>% select(value)
  footnote <- tf %>% filter(type == "footnote") %>% distinct() %>% select(value)
}
```

Program 4. Extraction of Each Output Components in r_tfl_rtfwrite()

The function then handles the no-observation case. It creates a one-column placeholder table and inserts the message "No data met the criteria for this listing." If data are present, it applies grouping logic to suppress repeated values within the requested grouping columns and inserts blank spacer rows between displayed rows. This improves readability while preserving the grouped structure.

The no-data and spacing logic are displayed below.

```
if(nrow(cells)==0 | ncol(cells)==0){
  cells <- setNames(data.frame(matrix("", nrow = 6, ncol = 1)), "c1")
  cells[4,1] <- "No data met the criteria for this listing."
  nodata <- TRUE
  cells_spaced <- cells
} else {
  nodata <- FALSE

  cols_to_fill <- cells %>%
    summarise(across(everything(), ~any(. != ""))) %>%
    unlist() %>% which()
  group_cols <- names(cols_to_fill[1:GroupCols])

  cells <- cells %>%
    group_by(c1) %>%
    mutate(c1 = if_else(row_number() == 1, c1, "")) %>%
    ungroup()

  blank_row <- cells[1,]
  blank_row[] <- ""
  cells_spaced <- cells[rep(1:nrow(cells), each = 2),]
  cells_spaced[seq(2, nrow(cells_spaced), 2),] <- blank_row
}
```

Program 5. No-data Handling, Grouping, And Row Spacing

After the content has been prepared, the function constructs the formatted output using the `r2rtf` package. The final output construction is shown below.

```
rtf_table <- cells_spaced %>%
  rtf_page(orientation = "landscape",
           margin = c(0.375, 0.375, 1, 0.375, 0.375, 0.375),
           col_width = 10.25,
           nrow = 30) %>%
  rtf_page_header(text = "Page \\pagenumber of \\pagefield",
                  text_font_size = 9,
                  text_font = 9) %>%
  rtf_title(c(heading, title$value, titlenote),
            text_font_size = 9,
            text_font = 9,
            text_justification = c("l", "c", "l"))

if(nrow(footnote) != 0){
  rtf_table <- rtf_table %>%
    rtf_page_footer(footnote, text_font_size = 9,
```

```
                    text_justification = "l", text_font = 8)
}

rtf_table %>%
  rtf_encode() %>%
  r2rtf::write_rtf(rtfout)
```

Program 6. RTF Output Construction

This implementation makes `r_tfl_rtfwrite()` more than a simple export routine. It is a document-construction function that combines listing content, title and footnote metadata, grouping rules, and page-layout controls into a submission-conformant RTF file.

GENERATE_BIMO_LISTINGS(): STUDY LEVEL DRIVER FUNCTION

To streamline production operations with an intuitive, user-friendly approach, we developed a study-level driver function `generate_bimo_listings()`. This function formalizes the workflow and exposes only three study-facing inputs: `rtf_names`, `GroupCols`, and `study_text`.

The function body can be interpreted in three major parts: initialization, inputs reading, and production generation.

During initialization, the function performs input validation and environment setup. In particular, it parses `study_text` with a regular expression. For a value such as “VXxxx-yyy”, the code extracts the Compound and Study ID components and uses them to retrieve site name information, construct the page heading and create output filenames.

The initialization pattern is displayed below.

```
# xxx and yyy are used as dummy compound and study number

generate_bimo_listings <- function(
    rtf_names,
    GroupCols,
    study_text = "VXxxx-yyy"
) {
  # ---------------------------
  # basic checks
  # ---------------------------
  if (length(rtf_names) != length(GroupCols)) {
    stop("`rtf_names` and `GroupCols` must have the same length.")
  }

  # ---------------------------
  # paths derived from study_text
  # ---------------------------
  base_dir <- "."
  listings_dir <- file.path(base_dir, study_text, "listings")
  output_dir <- file.path(base_dir, study_text, "R_listings")

  if (!dir.exists(listings_dir)) {
    stop("Listings directory does not exist: ", listings_dir)
  }

  if (!dir.exists(output_dir)) {
    dir.create(output_dir, recursive = TRUE)
  }

  # ---------------------------
  # parse study_text for dynamic metadata
  # example: VXxxx-yyy -> Compound xxx, Study ID yyy
  # ---------------------------
```

```
m <- regexec("^VX?(\\d+)-(\\d+)$", study_text)
parts <- regmatches(study_text, m)[[1]]

if (length(parts) < 3) {
  stop("`study_text` must look like 'VXxxx-yyy'.")
}

compound_num <- parts[2]
protocol_num <- parts[3]

site_info_path <- file.path(
  "S:/cdm/Development/Current",
  compound_num,
  protocol_num,
  "sys_site.sas7bdat"
)

heading <- sprintf(
  "Vertex Pharmaceuticals Incorporated\nVertex Program: %s BIMO",
  study_text
)

# ---------------------------
# required packages
# ---------------------------
required_pkgs <- c("striprtf", "stringr", "dplyr", "tidyr", "r2rtf", "haven")
missing_pkgs <- required_pkgs[
  !vapply(required_pkgs, requireNamespace, logical(1), quietly = TRUE)
]
if (length(missing_pkgs) > 0) {
  stop("Please install required package(s): ", paste(missing_pkgs, collapse = ", "))
}

# ---------------------------
# read site metadata dynamically
# ---------------------------
if (!file.exists(site_info_path)) {
  stop("Site metadata file not found: ", site_info_path)
}
site_info <- haven::read_sas(site_info_path)

if (!all(c("SITENUM", "SITENAME") %in% names(site_info))) {
  stop("`site_info` must contain SITENUM and SITENAME.")
}
```

Program 7. Initialization of generate_bimo_listings()

The second part reads the inputs, where each file in `rtf_names` is passed to `r_tfl_rtfread(..., DisplayColWidths = TRUE)`. A unique set of site numbers is then identified and saved in the `sites` object for per-site output generation in the next step. The core pattern is displayed below.

```
# ---------------------------
# read all RTF listings
# ---------------------------
tfl_list <- vector("list", length(rtf_names))
tf_list  <- vector("list", length(rtf_names))

for (j in seq_along(rtf_names)) {
  rtffile <- file.path(listings_dir, rtf_names[j])

  if (!file.exists(rtffile)) {
    stop("RTF file not found: ", rtffile)
  }
```

```
  tl_list <- r_tfl_rtfread(rtffile, DisplayColWidths = TRUE)
  tfl_list[[j]] <- tl_list$tfl
  tf_list[[j]]  <- tl_list$tf
}

# ---------------------------
# identify distinct sites
# ---------------------------
sites <- unique(dplyr::bind_rows(tfl_list)$siteid)
sites <- sites[!is.na(sites) & nzchar(as.character(sites))]
```

Program 8. Reading part of generate_bimo_listings()

The third part generates the production outputs. A nested by-site-by-listing loop, first calls `r_tfl_rtfwrite()` to produce temporary RTF files for each site-listing combination, then calls a built-in function `combine_rtf_sections()` to read the temporary files, insert page breaks between sections, create the by-section page numbers, and write one integrated BIMO output per site. The temporary files are then deleted.

The core production generation pattern is displayed below.

```
# ---------------------------
# helper: combine RTF sections
# ---------------------------
combine_rtf_sections <- function(input, output) {
  rtf <- lapply(input, readLines)
  n <- length(rtf)

  start <- c(
    1,
    vapply(
      rtf[-1],
      function(x) max(grep("fcharset", rtf[[1]])) + 2,
      numeric(1)
    )
  )

  end <- vapply(rtf, length, numeric(1))
  if (n > 1) {
    end[-n] <- end[-n] - 1
  }

  for (i in seq_len(n)) {
    rtf[[i]] <- rtf[[i]][start[i]:end[i]]
    if (i < n) {
      rtf[[i]] <- c(rtf[[i]], "\\sect\\pgnrestart\n")
    }
  }

  rtf <- lapply(rtf, function(x) gsub("NUMPAGES", "SECTIONPAGES", x))
  rtf <- do.call(c, rtf)

  write_rtf(rtf, output)
}


# ---------------------------
# generate one BIMO file per site
# ---------------------------
for (j in seq_along(sites)) {
  site <- as.character(sites[j])

  site_name <- site_info |>
    dplyr::filter(as.character(SITENUM) == site) |>
```

```
      dplyr::pull(SITENAME)

    if (length(site_name) > 0 && !is.na(site_name[1])) {
      titlenote <- sprintf("Site: (%s) %s \n", site, site_name[1])
    } else {
      titlenote <- sprintf("Site: (%s) \n", site)
    }

    temp_files <- character(length(tfl_list))

    for (k in seq_along(tfl_list)) {
      tfl <- tfl_list[[k]] |>
        dplyr::filter(as.character(siteid) == site)

      if (nrow(tfl) == 0) {
        tfl <- tfl_list[[k]] |>
          dplyr::select(dplyr::matches("^h\\d+|^g\\d+|^hw\\d+|^gw\\d+")) |>
          dplyr::distinct()
      }

      tf <- tf_list[[k]]
      temp_file <- sprintf("temp_tbl%s.rtf", k)
      temp_files[k] <- temp_file

      r_tfl_rtfwrite(
        tfl = tfl,
        tf = tf,
        GroupCols = GroupCols[k],
        heading = heading,
        titlenote = titlenote,
        rtfout = temp_file
      )
    }

    output_file <- file.path(
      output_dir,
      sprintf("%s-site-%s-bimo.rtf", study_text, site)
    )

    combine_rtf_sections(
      input = temp_files,
      output = output_file
    )

    unlink(temp_files)
  }
}
```

Program 9. Production Generation of generate_bimo_listings()

Several aspects of the `generate_bimo_listings()` function are technically significant. First, the study identifier serves as a compact source of metadata for directory construction, site information lookup, document headings, and output filenames. This design eliminates hard-coded protocol-specific values from the workflow and makes the same function portable across studies that follow the same naming convention. Second, the implementation demonstrates clear functional decomposition: file parsing is delegated to `r_tfl_rtfread()`, document construction is delegated to `r_tfl_rtfwrite()`, and section assembly is handled by `combine_rtf_sections()`. Third, the nested site-by-listing iteration makes the workflow naturally scalable. The same driver can be easily reused across protocols.

EXAMPLE ILLUSTRATION

Application of the tool can be illustrated using an FDA approved study. The `rtf_names` object defines the complete set of input RTF listings to be included in the BIMO package, while `GroupCols` specifies the number of grouping columns to be applied when each listing is reformatted. The `generate_bimo_listings()` function is executed using these two inputs together with the study identifier to generate site-level BIMO outputs in a standardized and reproducible manner. In the following program demonstration, "VXxxx-yyy" is used as a dummy study number for confidentiality. The list of outputs is shown in Figure 6, and an individual output example is shown in Figure 7.

```
# xxx and yyy are used as dummy compound and study number

rtf_names <- c(
"l-BIMO-consent-yyy.rtf",
"l-BIMO-treat-yyy.rtf",
"l-BIMO-ds-disc-yyy.rtf",
"l-BIMO-pop-yyy.rtf",
"l-BIMO-ie-yyy.rtf",
"l-BIMO-ae-yyy.rtf",
"l-BIMO-deviation-yyy.rtf",
"l-BIMO-qs-spid48-yyy.rtf",
"l-BIMO-qs-nprs-yyy.rtf",
"l-BIMO-qs-nprs-reduc-yyy.rtf",
"l-BIMO-cm-yyy.rtf",
"l-BIMO-cm-resmed-yyy.rtf",
"l-BIMO-lb-chem1-yyy.rtf",
"l-BIMO-lb-chem2-yyy.rtf",
"l-BIMO-lb-hema-yyy.rtf",
"l-BIMO-lb-coag-yyy.rtf",
"l-BIMO-ecg-yyy.rtf",
"l-BIMO-vs-yyy.rtf"
)

GroupCols <- c(1, 1, 1, 1, 1, 3, 1, 1, 2, 1, 1, 1, 3, 3, 3, 3, 3, 3)

generate_bimo_listings(
    rtf_names = rtf_names,
    GroupCols = GroupCols,
    study_text = "VXxxx-yyy"
)
```

Program 10. Illustration of example study BIMO listing generation

VX[redacted]-site-005-bimo
VX[redacted]-site-016-bimo
VX[redacted]-site-018-bimo
VX[redacted]-site-058-bimo
VX[redacted]-site-059-bimo
VX[redacted]-site-060-bimo
VX[redacted]-site-062-bimo
VX[redacted]-site-063-bimo
VX[redacted]-site-065-bimo
VX[redacted]-site-085-bimo
VX[redacted]-site-145-bimo
VX[redacted]-site-153-bimo
VX[redacted]-site-155-bimo
VX[redacted]-site-169-bimo
VX[redacted]-site-170-bimo
VX[redacted]-site-176-bimo
VX[redacted]-site-177-bimo
VX[redacted]-site-257-bimo
VX[redacted]-site-279-bimo
VX[redacted]-site-280-bimo
VX[redacted]-site-392-bimo
VX[redacted]-site-860-bimo
VX[redacted]-site-867-bimo

Figure 6 List of Outputs from the Illustration Program

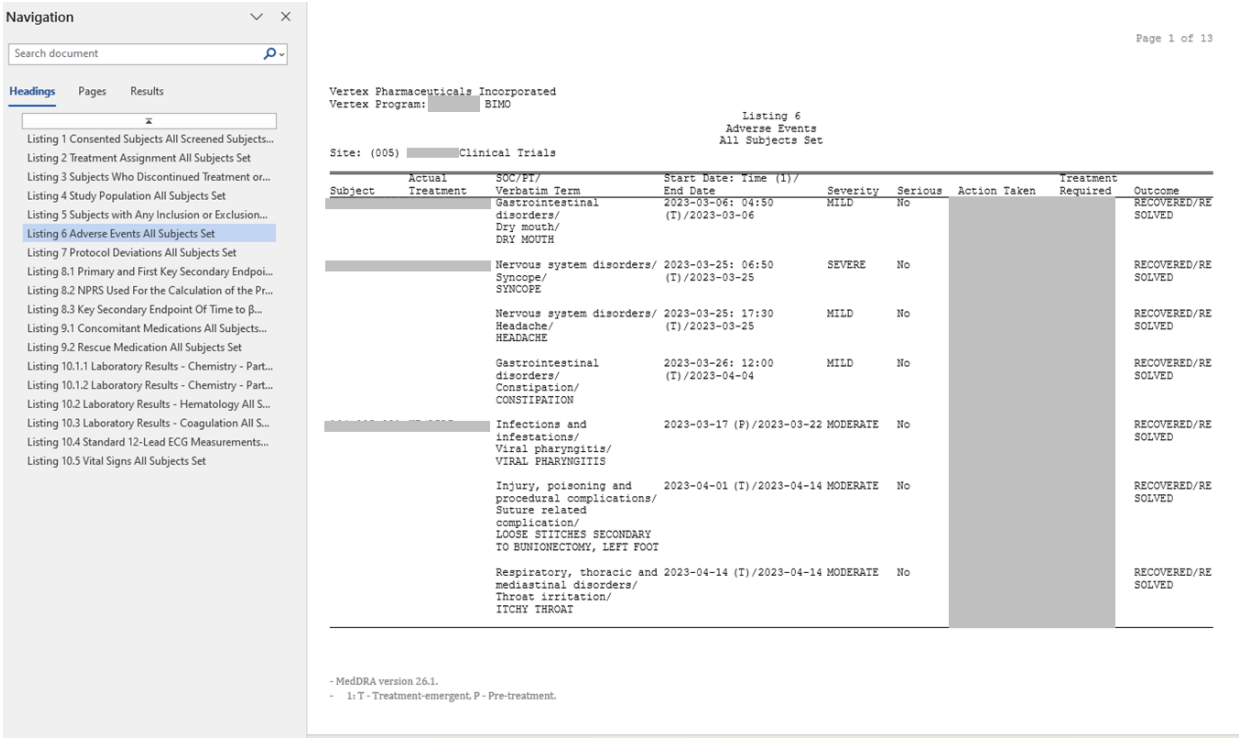


Figure 7 Example Individual Output from the Illustration Program

Operationally, programmers can prepare a complete site-level BIMO package for a study by defining the set of input listings once and then calling a single wrapper function. This is especially useful in large submission settings, where the total number of outputs can become substantial. By integrating the listing-read, site-subset, listing-write, and section-combine logic into one function call, the highly generalized workflow improves reproducibility and reduces the amount of study-specific code maintenance without sacrificing flexibility.

DISCUSSION

PRACTICAL BENEFITS

Our R tool provides several clear practical benefits. The primary benefit is its efficiency. By applying the outputs-to-outputs approach, the process eliminates duplicated manual listing specifications generation, listing production and validation efforts. A second benefit is standardization: with the same production pattern reusable across studies and compounds, we are able to uniform the structure and formatting across all BIMO listings for submission studies. Thirdly, our tool enhances accuracy and simplifies quality control. The standardized automation tool minimizes the risk of human error. Furthermore, the traditionally required level 3 QC of the listings could be replaced by a level 1 validation against the CSR listings, which significantly reduces QC burden.

IMPLEMENTATION EXPERIENCE AND BROADER USE

This tool has been applied in two Vertex development programs, both of which ultimately received FDA approval. Notably, one of these approvals occurred in 2025, after the most recent BIMO Technical Conformance Guide had taken effect. Based on this implementation experience, we believe the workflow has been proved to satisfy current regulatory expectations and technical standards for BIMO listing generation.

Beyond the immediate BIMO use case, the core functions `r_tfl_rtfread()` and `r_tfl_rtfwrite()` have broader utility in clinical reporting and programming workflows. Because these functions enable structured extraction of content from RTF outputs and reconstruction of formatted deliverables, they can support a range of downstream applications. Examples include quality control of production outputs, conversion of RTF listings into Excel workbooks or SAS datasets, and generation of presentation-ready materials such as PowerPoint-based reports for internal or external presentation of study results.

FUTURE WORK

A natural extension of the current framework is automation of additional BIMO package components. One direction already identified is the use of R Markdown to generate sections of the BIMO Data Reviewer's Guide, including material related to the list of subject-level listings, site summaries, and external datasets and sources.

CONCLUSION

Subject-level data line listings are a required and inspection-critical component of the BIMO package. However, conventional production approaches often recreate content that already exists in validated CSR outputs, leading to unnecessary programming effort and introducing potential human-errors.

The R-based workflow described here addresses this inefficiency by introducing a structured intermediate representation between existing RTF listings and submission-ready site-level deliverables. This tool reduces manual effort, simplifies quality control, and improves traceability from CSR to BIMO deliverables.

Initial implementation experience indicates that the approach is practical in real submission settings and flexible enough to support broader output reuse beyond the immediate BIMO use case. For teams seeking to modernize listing production, the outputs-to outputs model provides a strong foundation for further automation.

REFERENCES

[1] U.S. Food and Drug Administration. 2024. "Standardized Format for Electronic Submission of NDA and BLA Content for the Planning of Bioresearch Monitoring (BIMO) Inspections for CDER Submissions." Accessed March 20, 2026. https://www.fda.gov/regulatory-information/search-fda-guidance-

documents/standardized-format-electronic-submission-nda-and-bla-content-planning-bioresearch-monitoring-bimo.

[2] U.S. Food and Drug Administration. 2024. “Bioresearch Monitoring Technical Conformance Guide.” Accessed March 20, 2026. www.fda.gov/regulatory-information/search-fda-guidance-documents/bioresearch-monitoring-technical-conformance-guide.

ACKNOWLEDGMENTS

We would like to express our gratitude to the Vertex Statistical Programming Team, especially Todd Case, Ina Jazic, and Lingyun Chen, for their support on this project. We also express our thanks to the Vertex TRIKAFTA and JOURNAVX Biometrics Team for their foundational work, which served as the reference for this project.

CONTACT INFORMATION

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

Weishan Song
Vertex Pharmaceuticals Inc.
Weishan_Song@vrtx.com

Margaret Huang
Vertex Pharmaceuticals Inc.
Margaret_huang@vrtx.com