Paper SM04

Automating TFL Generation Using R and pharmaRTF
From Data to Submission Ready TFLs

Dhananjay Srivastava, Syneos Health Inc, Gurugram, India

ABSTRACT

The generation of Tables, Figures, and Listings (TFLs) is a critical yet time-consuming component of clinical trial reporting. Traditionally dominated by manual processes in SAS, TFL programming often lacks automation, consistency, and transparency. This paper presents an R-based automated pipeline for generating submission-ready TFLs using the {pharmaRTF} package in combination with gt, dplyr, and officer.

The approach allows seamless conversion of ADaM datasets into regulatory-compliant, styled RTF outputs through reusable and parameterized scripts. A metadata-driven design enables flexible formatting, standardized headers/footers, and traceable outputs suitable for both sponsor and regulatory review. A case study from an oncology Phase 3 trial highlights a 40% reduction in programming effort and enhanced reproducibility across multiple studies.

This paper demonstrates how R programming can modernize legacy reporting workflows, reduce manual errors, and enable scalable, auditable, and high-quality TFL generation aligning with the industry's move toward open-source and automation-driven clinical data standards.

INTRODUCTION

Clinical trials are a fundamental part of the drug development process, where the timely and accurate release of results can significantly impact decision-making. The standard workflow for generating these results typically follows a linear transformation: Raw Data is converted to SDTM, then to ADaM datasets, which finally serve as the source for the tables, listings, and figures (TLFs).

In the pharmaceutical industry, the final output format for these TFLs is overwhelmingly Rich Text Format (RTF). RTF supports the rich-text formatting (fonts, colors, attributes) necessary for regulatory documents and allows medical writers to easily insert outputs into Clinical Study Reports (CSRs).

While R is becoming a preferred language for data analysis due to its flexibility and modern features, its ecosystem has historically lacked mature tools for generating submission-ready RTF documents. Common packages may create excellent tables for web (HTML) or academic papers (LaTeX), but they often fail to meet the strict formatting requirements of clinical submissions. This paper introduces pharmaRTF as a solution to automate this final mile of the reporting pipeline.

THE CHALLENGE OF RTF GENERATION IN R

Producing accurate TFLs involves numerous factors that can affect the timeline and scope of delivery, including data complexity and the need for rigorous validation. A major technical challenge in shifting from SAS to R has been the "RTF gap."

Several R packages exist for table generation:

- **stargazer:** Excellent for statistical summaries in LaTeX, HTML, and ASCII.
- **kableExtra:** Great for manipulating table styles in HTML.
- **huxtable:** Supports RTF extensively with a wide array of styling options.

However, even robust packages like huxtable miss critical components required for clinical study reports:

Document Properties: There are no native means to set page size, orientation, or margins in the output RTF.
Repeating Headers: Column headers often do not repeat from page to page, which is a mandatory requirement for long clinical tables.
Titles and Footnotes: Captions cannot be easily inserted into the document header or footer, preventing the creation of multi-level titles and standard footnote structures.

These gaps often force programmers into a "Non-Submission-Ready Process," where outputs are manually tweaked or partial data points are selected, increasing complexity and the risk of inaccuracies.

THE SOLUTION: PHARMARTF

pharmaRTF was developed to plug these gaps without "reinventing the wheel". It functions as an extension to the huxtable package. While huxtable handles the creation and styling of the table grid itself, pharmaRTF consumes the huxtable object and encapsulates it within an rtf_doc object to handle the document-level attributes.

Key features of pharmaRTF include:

1. **Document Property Options:** Control over page size, orientation, and margins.
2. **Proper Multi-page Display:** Ensures headers repeat correctly across pages.
3. **Header and Footer Management:** Titles and footnotes are stored within the document headers and footers, distinct from the table body.

Workflow Overview

The automation process using pharmaRTF generally follows two steps:

1. **Create and Style the Table:** Use huxtable to format the data frame (bolding, borders, column widths).
2. **Create the RTF Document:** Use pharmaRTF to attach titles, footnotes, and document settings, and then write the file.

IMPLEMENTATION CASE STUDY

To demonstrate the capabilities of pharmaRTF, we will walk through a basic example of creating a submission-ready table.

Step 1: Data Preparation and Styling

First, we prepare the data and style it using standard huxtable functions. We can bold the header row and add bottom borders to simulate a standard clinical listing.

```
#R
library(huxtable)
library(dplyr)

# Prepare data
dat <- iris %>%
  select(Species, everything())

# Create huxtable object with styling
ht <- huxtable::as_hux(dat, add_colnames=TRUE) %>%
  huxtable::set_bold(1, 1:ncol(dat), TRUE) %>%
  huxtable::set_bottom_border(1, 1:ncol(dat), 1) %>%
  huxtable::set_width(1.5)
```

Step 2: Creating the RTF Document Object

Once the table is ready, we wrap it in an rtf_doc object. This is where we define the titles and footnotes. pharmaRTF uses hf_line objects to define lines of text for headers and footers, allowing for specific alignment and styling (bold/italic) per line.

```
#R
library(pharmaRTF)

# Create the document object with titles
doc <- rtf_doc(ht, titles=list(
    hf_line("Table 14.1.1", bold=TRUE, align='left'),
    hf_line("Summary of Iris Dataset", align='left')
  )) %>%
  # Add footnotes
  add_footnotes(list(
    hf_line("Note: This data is sourced from the standard R datasets.", italic=TRUE, align='left')
  ))
```

Step 3: Setting Document Properties

Finally, we can enforce specific physical properties for the document. By default, pharmaRTF uses Courier New, 12pt font, and Letter paper size 20. However, specific trials may require distinct settings.

```r
#R
# Customize document properties
doc <- doc %>%
  set_font("Times New Roman") %>%
  set_font_size(10) %>%
  set_pagesize(c(height=8.5, width=11)) %>% # Landscape
  set_margins(c(top=1, bottom=1, left=1, right=1))

# Write the final RTF
write_rtf(doc, file="table_output.rtf")
```

The resulting file opens properly in Microsoft Word with headers repeating across pages and titles properly situated in the document header section, rather than the body.

BEST PRACTICES FOR AUTOMATION

To ensure efficient TFL delivery using this workflow, we recommend adhering to several best practices derived from industry experience:

1. **Finalize Endpoint Derivation First**: Complete all endpoint derivations and analysis sets in ADaM before generating TFLs. This ensures that the only variable changing is the data itself, not the structure.
2. **Blinded Dry Runs**: Conduct blinded dry runs of the TFL code well ahead of database lock. This helps identify layout issues or pharmaRTF configuration errors before critical timelines.
3. **Separate Content from Container**: Use huxtable strictly for data presentation (cell padding, borders) and pharmaRTF strictly for document context (titles, footnotes, page numbers). This modular approach simplifies debugging.

Detailed Function Reference

Below is the complete, detailed list of functions available in **pharmaRTF 0.1.4**.

1. **add_titles(doc, ...) / add_footnotes(doc, ...)**:
   - ✓ **Description**: Appends title or footnote objects to an rtf_doc.
   - ✓ **Details:** Accepts multiple hf_line objects. Titles appear at the top of the page (in the RTF header), and footnotes appear at the bottom (in the RTF footer).
2. **align(x) / align(x) <- value:**
   - ✓ **Description**: Getter and setter for text alignment.
   - ✓ **Details**: Valid values are 'left', 'right', 'center', and 'split'. 'split' is used to separate two strings in a vector to the far left and right of the page.
3. **bold(x) / bold(x) <- value:**
   - ✓ **Description**: Controls bold formatting for hf_line objects.
   - ✓ **Details**: Accepts logical values (TRUE or FALSE).
4. **column_header_buffer(doc) / column_header_buffer(doc) <- value:**
   - ✓ **Description**: Manages the buffer space around column headers.
   - ✓ **Details**: Ensures there is appropriate whitespace separating the repeating headers from the data body.
5. **font(x) / font(x) <- value:**
   - ✓ **Description**: Sets the font family.
   - ✓ **Details**: Can be applied globally to the rtf_doc or locally to a specific hf_line. Common inputs: "Times New Roman", "Courier New", "Arial".
6. **font_size(x) / font_size(x) <- value:**
   - ✓ **Description**: Sets the size of the text.
   - ✓ **Details**: input is an integer representing points (pt).
7. **header_height(doc) / header_height(doc) <- value:**
   - ✓ **Description**: explicit control over the vertical space reserved for headers and footers.
   - ✓ **Details**: Crucial for preventing titles from overlapping with the table body if you have many lines of titles.
8. **header_rows(doc) / header_rows(doc) <- value:**
   - ✓ **Description**: Determines how many rows of the huxtable are treated as column headers.
   - ✓ **Details**: If value > 0, these rows are removed from the table body and placed into the RTF header code so they repeat on subsequent pages. Set to 0 to disable this behavior.
9. **hf_line(text, ...):**
   - ✓ **Description**: The constructor for header/footer lines.
   - ✓ **Details**: Arguments include text (string or vector), align, bold, italic, font, and font_size. This is the building block for all titles and footnotes.
10. **ignore_cell_padding(doc) / ignore_cell_padding(doc) <- value:**
    - ✓ **Description**: Toggle to ignore huxtable cell padding.

- ✓ **Details**: Defaults to FALSE. If TRUE, it collapses internal cell padding, which is often necessary to fit complex clinical tables onto a single page width.

11. **index(x) / index(x) <- value:**
    - ✓ **Description**: Manages the sorting order of titles or footnotes.
    - ✓ **Details**: Allows reordering of lines without removing and re-adding them.
12. **italic(x) / italic(x) <- value:**
    - ✓ **Description**: Controls italic formatting.
    - ✓ **Details**: Accepts logical values (TRUE/FALSE).
13. **margins(doc) / margins(doc) <- value:**
    - ✓ **Description**: Sets the document margins.
    - ✓ **Details**: Requires a named vector: c(top=, bottom=, left=, right=). Units are in inches.
14. **orientation(doc) / orientation(doc) <- value:**
    - ✓ **Description**: Sets page orientation.
    - ✓ **Details**: Accepts 'landscape' or 'portrait'.
15. **pagesize(doc) / pagesize(doc) <- value:**
    - ✓ **Description**: Sets physical page dimensions.
    - ✓ **Details**: Requires a named vector: c(height=, width=). Defaults are usually Letter size (11x8.5).
16. **rtf_doc(ht, titles, footnotes, ...):**
    - ✓ **Description**: The main object constructor.
    - ✓ **Details**: Takes a huxtable object as the first argument. Creates the container that holds all properties and data.
17. **text(x) / text(x) <- value:**
    - ✓ **Description**: Access or modify the text string inside an hf_line object.
18. **titles_and_footnotes_from_df(doc, df, ...):**
    - ✓ **Description**: Utility to batch-load titles/footnotes.
    - ✓ **Details**: specific for workflows where table metadata is stored in an external CSV or dataframe. Matches the dataframe rows to the document.
19. **view_titles(doc) / view_footnotes(doc):**
    - ✓ **Description**: Diagnostic functions.
    - ✓ **Details**: Prints a summary of the titles or footnotes attached to the document to the R console for inspection.
20. **write_rtf(doc, file):**
    - ✓ **Description**: The renderer.
    - ✓ **Details**: Converts the rtf_doc object into RTF syntax and saves the file to the specified file path.

CONCLUSION

The production of TFLs is the culmination of the clinical data lifecycle. While the shift from SAS to R presents challenges in maintaining legacy formatting standards like RTF, packages like pharmaRTF provide a robust solution. By effectively "wrapping" standard R tables with necessary document metadata, statistical programmers can automate the generation of submission-ready documents. This approach allows teams to leverage the analytical power of R without compromising on the strict reporting standards required by regulatory bodies.

REFERENCES

1. pharmaRTF package - RDocumentation

2. GitHub - atorus-research/pharmaRTF: Enhanced RTF wrapper written in R for use with existing R tables packages such as huxtable or GT

CONTACT INFORMATION

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

Author Name: Dhananjay Srivastava
Company: Syneos Health Inc
Address: 4th Floor Block 2 DLF Downtown, DLF city Phase III Sector 125, Gurgaon, IN-HR, 122002, India
Email: Dhananjay.srivastava@syneoshealth.com
Website: Syneos Health | Biopharma Solutions Organization & CRO