Paper PP01

TLF_PACKAGER: Automated TFL Packaging and Bookmarking for RTF and PDF Outputs with User-Driven Excel Review and Approval Using Python

Manivannan Mathialagan, Veristat, Chennai, India

ABSTRACT

Clinical reporting teams routinely generate large sets of Tables, Listings, and Figures (TLFs) in mixed formats such as RTF, DOCX, and PDF. The final packaging step, where these files must be combined into a single bookmarked PDF for internal review or regulatory submission, is often manual, inconsistent, and difficult to reproduce. TLF_PACKAGER is a Python-based automation tool that addresses this gap by scanning a folder of TLF outputs, extracting titles directly from RTF headers, DOCX headers and body text, or the first page of PDFs, and writing all extracted information to an Excel worksheet. Users can reorder files, adjust bookmark labels, and approve the final sequence within Excel before merging. The tool then converts RTF and DOCX files to PDF using Microsoft Word automation, merges all files in the approved order, applies bookmarks, and inserts a multi-page clickable Table of Contents generated programmatically. TLF_PACKAGER can be run directly using Python, via an interactive Windows batch file, or from SAS using the X command, making it adaptable to different programming environments. This approach brings transparency, reproducibility, and efficiency to the TLF packaging process, reducing manual effort and providing a standardized method suitable for CSR, DSMB, and regulatory submissions.

INTRODUCTION

In clinical study reporting, large collections of Tables, Listings, and Figures are produced repeatedly throughout a project's lifecycle. These outputs are commonly generated in different formats depending on the study team's workflow, toolchain, or sponsor expectations. RTF files may originate from SAS procedures, DOCX files from template-driven reporting tools, and PDFs from statistical graphics packages or automated pipelines. Although the creation of TLFs has become highly automated through standard macros and controlled programming environments, the final packaging of these outputs into a single, navigable PDF is still handled manually in many organizations.

This final step is far from trivial. Programmers often sort files manually, rename them to match SAP numbering, create or adjust bookmarks using desktop PDF tools, and ensure titles appear correctly. When the volume of TLFs is high or when multiple versions must be generated (CSR, DSMB, DMC, interim analyses), the manual approach becomes time-consuming, error-prone, and difficult to maintain consistently across studies. Furthermore, external reviewers and statisticians frequently expect clear bookmarks, a logical flow, and a clean Table of Contents, none of which are guaranteed without a structured process.

TLF_PACKAGER was developed to automate this packaging step while keeping the review and approval process transparent. The tool extracts titles directly from each file, writes them to a formatted Excel sheet, and allows the user to control ordering and bookmark text. After approval, the tool merges all outputs into a single PDF with bookmarks and a fully clickable Table of Contents. Because it can be executed using Python, launched via a batch file, or invoked from SAS using the X command, it fits naturally into different reporting environments without requiring changes to existing programming workflows.

This paper explains the motivation behind the tool, the design decisions, the implementation approach, and how this method helps bring reproducibility and standardization to the final stage of TLF preparation.

AUTOMATED PACKAGING WORKFLOW OVERVIEW

The TLF_PACKAGER workflow starts with scanning the user-specified folder for RTF, DOCX, and PDF files. Once detected, the script automatically extracts titles from each file type using format-specific logic. These extracted titles serve as initial bookmark candidates, which are exported into an Excel review sheet. Users can reorder, rename, and refine the bookmark text before proceeding. After approval, the tool converts RTF and DOCX files to PDF (if needed), merges all files into a single consolidated PDF, applies bookmarks according to the reviewed Excel sheet, and finally inserts a multi-page clickable Table of Contents. This ensures that the resulting PDF is well-organized and suitable for internal review or submission workflows.

The TLF_PACKAGER script implements a linear, stepwise workflow that starts from a user-specified folder and ends with a single, bookmarked PDF containing all selected TLF outputs. The intent is to keep the overall logic simple and transparent, while still handling mixed formats (RTF, DOCX, PDF) and providing a controlled review step via Excel.

The core workflow can be described as:

- Get input parameters (folder path, output PDF name, flags).
- Scan the folder and classify files by type.
- Extract titles and construct bookmark text for each file.
- Export all information to Excel for user review and ordering.
- Optionally pause and wait for user approval.
- Read the approved order from Excel.
- Convert RTF and DOCX files to PDF using Microsoft Word.
- Merge all PDFs, apply bookmarks, and generate a clickable Table of Contents.

Step 1: Reading the folder path and runtime parameters

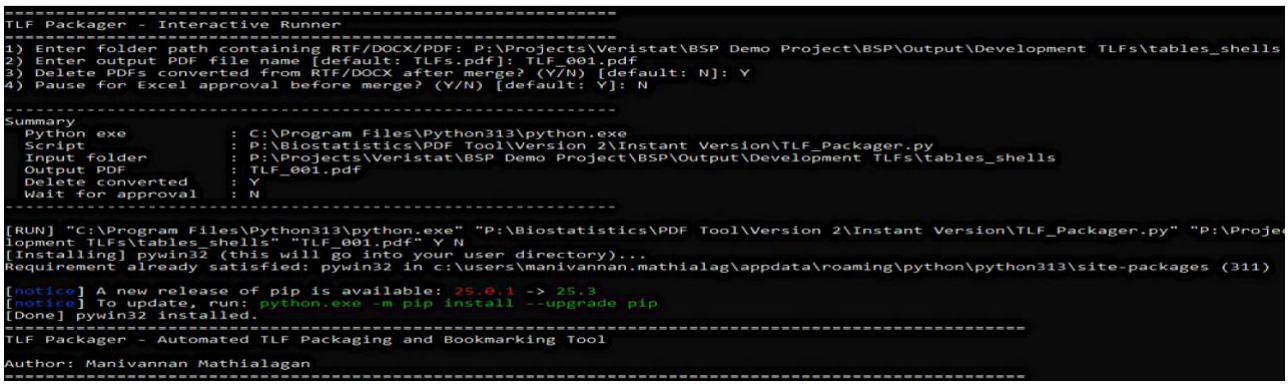
Code Snippet 1: Command-line argument handling in the Python CLI

The script is primarily designed as a command-line tool that accepts the folder path, output PDF name, and two optional flags controlling deletion of converted PDFs and whether to pause for Excel review.

```
if __name__ == "__main__":
    if len(sys.argv) < 3 or len(sys.argv) > 5:
        print("Usage: python TLF_Packager.py <folder_path> <output_pdf_name> "
              "[delete_converted_pdfs: Y/N] [wait_for_user_approval: Y/N]")
        sys.exit(1)

    path = sys.argv[1]
    output_pdf_name = os.path.basename(sys.argv[2])
    delete_intermediate_pdfs = sys.argv[3].strip().upper() == "Y" if len(sys.argv) > 3
else False
    wait_for_user_approval = sys.argv[4].strip().upper() == "Y" if len(sys.argv) > 4
else True

    if not os.path.isdir(path):
        print(f"Error: Folder not found - {path}")
        sys.exit(1)

    main(path, output_pdf_name, delete_intermediate_pdfs, wait_for_user_approval)
```

This block validates the arguments, confirms the folder exists, and then calls the main workflow function with the chosen options.

Figure 1. Example of command-line usage of TLF_Packager.py in a Windows terminal, showing folder path, output PDF name, and flags.



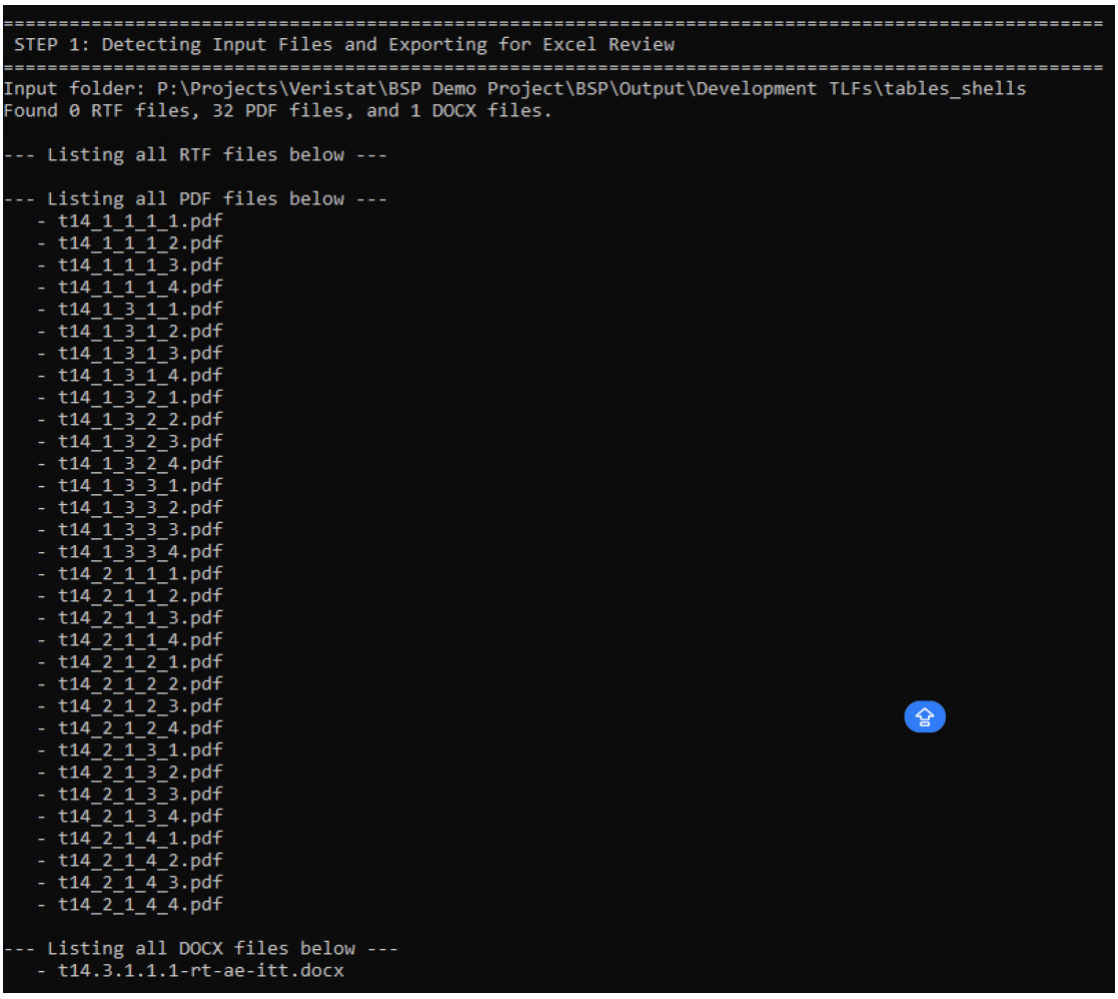
Step 2: Listing RTF, DOCX, and PDF files in the folder

Code Snippet 2: File discovery and classification

Once the folder is known, the script scans it for supported TLF file types. Temporary Office files (for example, those starting with "~$") are skipped, and a simple diagnostic listing is printed to the console.

```
rtf_files = sorted(
    f for f in os.listdir(folder)
    if f.lower().endswith('.rtf') and not f.startswith('~$')
)

pdf_files = sorted(
    f for f in os.listdir(folder)
    if f.lower().endswith('.pdf') and not f.startswith('~$')
)

docx_files = sorted(
    f for f in os.listdir(folder)
    if f.lower().endswith('.docx') and not f.startswith('~$')
)

print(f"Found {len(rtf_files)} RTF files, {len(pdf_files)} PDF files,
{len(docx_files)} DOCX files.")
```

Figure 2. Example of listing all files present in the folder.



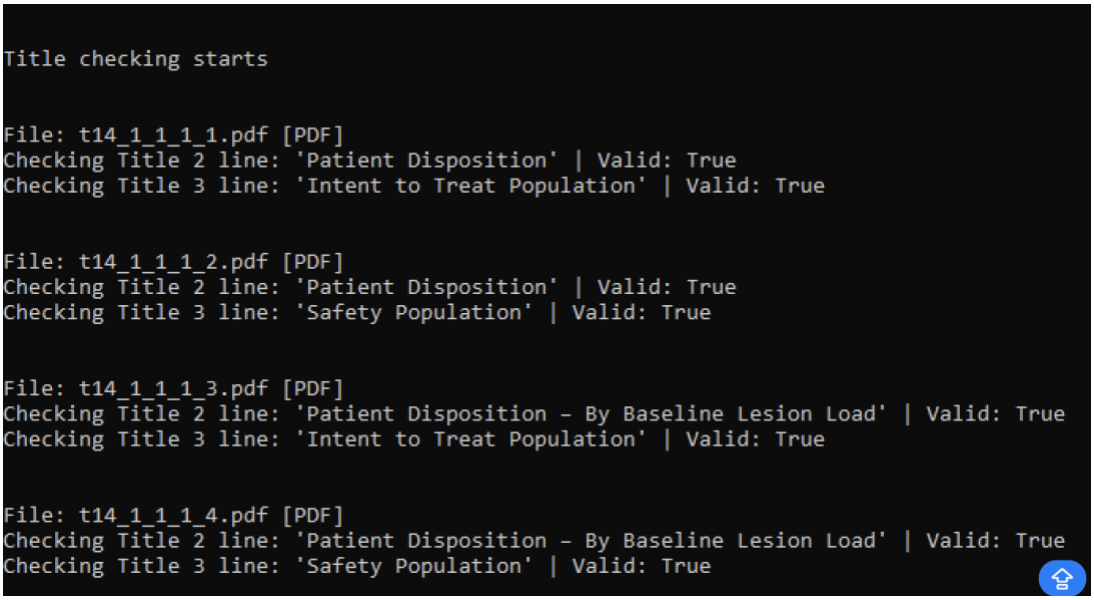
Step 3: Extracting titles and constructing bookmark text

Code Snippet 3: Looping through files and calling title extraction functions

The script then iterates over each file and calls a format-specific extraction function. The extracted title lines and combined bookmark text are stored in a unified structure that is later exported to Excel and re-used for merging.

```
# RTF title extraction
for rtf in rtf_files:
    rtf_path = os.path.join(folder, rtf)
    title1, title2, title3, bookmark = extract_titles_from_rtf(rtf_path)
    entries.append({
        'type': 'RTF', 'file': rtf_path,
        'title1': title1, 'title2': title2, 'title3': title3, 'bookmark': bookmark
    })

# PDF title extraction
for pdf in pdf_files:
    pdf_path = os.path.join(folder, pdf)
    title1, title2, title3, bookmark = extract_title_from_pdf(pdf_path)
    entries.append({
        'type': 'PDF', 'file': pdf_path, 'title1': title1, 'title2': title2,
        'title3': title3,'bookmark': bookmark
    })

# DOCX title extraction
for docx_file in docx_files:
    docx_path = os.path.join(folder, docx_file)
    title1, title2, title3, bookmark = extract_title_from_docx(docx_path)
    entries.append({
        'type': 'DOCX','file': docx_path,'title1': title1,'title2': title2,
       'title3': title3,'bookmark': bookmark
    })
```

Figure 3. Console output showing automatic extraction and validation of Title 2 and Title 3 lines from multiple PDF TLF files.



At the end of this step, the entries list contains one record per file, with the file type, full path, three title lines, and the initial bookmark text that will appear in the consolidated PDF.

Step 4: Exporting extracted titles to Excel for user review

Code Snippet 4: Creating the Excel workbook that stores filenames, titles, and bookmark text

```
excel_file = os.path.join(folder, "TLF_Bookmark_Worksheet.xlsx")
write_titles_to_excel(entries, excel_file)
```

This step generates a formatted Excel sheet containing one row per TLF output. Users can adjust the ordering, modify bookmark text, or fix title lines. This review sheet serves as the control point before PDF merging.

Step 5: Pausing execution for Excel approval (optional)

Code Snippet 5: Prompting the user before proceeding with the merge

```
if wait_for_user_approval:
    print("User action required: review the Excel file, save it, and close it.")
    gui_pause_and_debug(excel_file)
else:
    print("Skipping user approval as per input parameter.")
```

If approval is enabled, the script waits until the user manually reviews and closes the Excel file. If disabled, the script will continue without pause, reusing the previous Excel ordering.

Step 6: Reading the approved order back from Excel

Code Snippet 6: Importing the reviewed entries in the exact user-defined order

```
entries = read_entries_from_excel(excel_file, folder)
print("Final merge order:")
for idx, e in enumerate(entries, 1):
    print(f"{idx:2d}. {os.path.basename(e['file'])} --> {e['bookmark']}")
```

This ensures that the PDF merge sequence respects the user's manual ordering and bookmark text updates.

Step 7: Converting RTF and DOCX files to PDF using Microsoft Word

Code Snippet 7: Word automation for reliable conversion

```
if entry['type'] in ['RTF', 'DOCX']:
    pdf_path = convert_to_pdf_with_word(entry['file'], os.path.dirname(entry['file']))
else:
    pdf_path = entry['file']
```

Microsoft Word is used for conversion because it handles page layout, headers, footnotes, and table widths consistently. This results in uniform PDFs regardless of original source formatting.

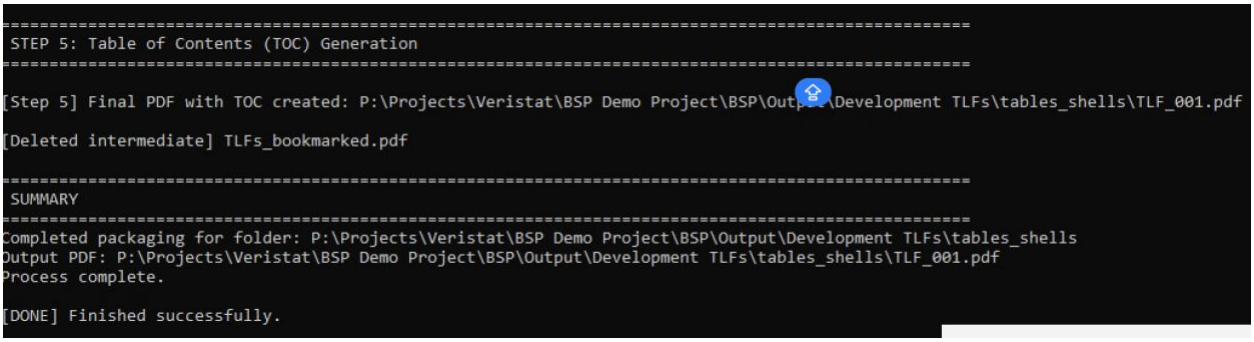
Step 8: Merging all PDFs, applying bookmarks, and generating a clickable Table of Contents

Code Snippet 8: Final merge and Table of Contents insertion

```
bookmarked_pdf = os.path.join(folder, "TLFs_bookmarked.pdf")
merge_pdfs_with_bookmarks(entries, bookmarked_pdf)
add_toc_and_links(bookmarked_pdf, final_pdf, font_size=8)
```

The script merges all PDFs in the reviewed order, adds bookmarks for each entry, then inserts a multi-page, clickable Table of Contents referencing every bookmark. The output is a single, audit-ready PDF suitable for CSR, DSMB, and submission workflows.

Figure 4. Final console summary showing successful PDF merge, TOC creation, and completion of the TLF packaging process.



TOOL EXECUTION MODES AND USAGE EXAMPLES

The TLF_PACKAGER engine is the same in all environments, but different users may prefer different ways of running it. Some programmers are comfortable with a Python command line, some prefer a guided batch script, and many SAS users want to trigger packaging from a run-all program. This section shows how the tool can be executed in three modes and explains the parameters used in each case.

Command-line parameters (common meaning across all modes)

- folder_path
  - path to the folder that contains RTF, DOCX, and PDF outputs.
- output_pdf_name
  - name of the final merged PDF (for example, Study123_TLFs.pdf)
- delete_converted_pdfs
  - Y to delete intermediate PDFs converted from RTF/DOCX after the merge;
  - N to keep them.
- wait_for_user_approval
  - Y to pause after Excel export and wait for the user to review/save/close the worksheet;
  - N to proceed automatically.

Option 1 - DIRECT PYTHON CLI USAGE

In the most direct mode, the script is executed from a command prompt using the Python interpreter. This mode is typically used by programmers or power users who are comfortable with command-line tools.

Example command:

```
python TLF_Packager.py "P:\Projects\Study123\Output" "Study123_TLFs.pdf" N Y
```

In this example:

- The input folder is P:\Projects\Study123\Output.
- The final merged PDF is named Study123_TLFs.pdf.
- Converted PDFs from RTF/DOCX are not deleted (N).
- The script pauses for Excel review and waits for approval (Y).

After the command is issued, the console shows the detected files, title extraction logs, and finally a summary that confirms the merged PDF and TOC creation. A screenshot (for example, Figure 1) can illustrate the command line and the initial step messages. This mode allows users to run the tool directly from a command prompt using the Python interpreter, making it the fastest option for programmers comfortable with CLI execution. It provides full control of all parameters in a single command.

Option 2 - INTERACTIVE BATCH FILE LAUNCHER

The batch file wrapper is intended for users who prefer a guided, question–answer style interface instead of typing all parameters on one command line. The batch file asks the user for each required value and then calls the Python script with those values.

Key part of the batch script:

```
set "PYTHON_EXE=C:\Program Files\Python313\python.exe"
set "SCRIPT_PATH=%~dp0TLF_Packager.py"

rem 1) Folder path
set "FOLDER_PATH="
set /p FOLDER_PATH=1) Enter folder path containing RTF/DOCX/PDF:

rem 2) Output PDF name (default TLFs.pdf)
set "OUTPUT_PDF="
set /p OUTPUT_PDF=2) Enter output PDF file name [default: TLFs.pdf]:
if not defined OUTPUT_PDF set "OUTPUT_PDF=TLFs.pdf"

rem 3) Delete converted PDFs? (Y/N)
set "DEL_YN="
set /p DEL_YN=3) Delete PDFs converted from RTF/DOCX after merge? (Y/N) [default: N]:
if /I "%DEL_YN%"=="Y" (set "DEL_YN=Y") else (set "DEL_YN=N")

rem 4) Wait for Excel approval? (Y/N)
set "WAIT_YN="
set /p WAIT_YN=4) Pause for Excel approval before merge? (Y/N) [default: Y]:
if /I "%WAIT_YN%"=="N" (set "WAIT_YN=N") else (set "WAIT_YN=Y")

"%PYTHON_EXE%" "%SCRIPT_PATH%" "%FOLDER_PATH%" "%OUTPUT_PDF%" %DEL_YN% %WAIT_YN%
```

The prompts map directly to the four parameters:

1. FOLDER_PATH → folder_path
2. OUTPUT_PDF → output_pdf_name
3. DEL_YN → delete_converted_pdfs
4. WAIT_YN → wait_for_user_approval

A screenshot (for example, Figure 1) can show the batch window prompting for each value, followed by the standard Python console log with title extraction and final summary.

The batch file launcher provides a guided, user-friendly interface for running TLF_PACKAGER without requiring any command-line knowledge. Instead of typing parameters manually, users are prompted one by one for the folder path, output PDF name, deletion preference, and approval pause option. This makes the tool accessible to non-technical users, statisticians, or team members who prefer a simple point-and-enter workflow. The batch file then constructs the appropriate command and calls the Python script automatically, ensuring consistent execution while reducing the chance of input errors.

Option 3 - SAS MACRO WRAPPER USING X COMMAND

In many clinical programming workflows, the preferred entry point is a SAS program. To support this, TLF_PACKAGER can be wrapped in a simple SAS macro that constructs the batch command and calls it via the X statement (or SYSTASK if desired).

In addition, the SAS wrapper can dynamically construct all required arguments—such as input folder, output PDF name, deletion flags, and TOC options—based on macro parameters or project-level metadata. This allows the CLI to be triggered in a fully automated fashion during batch runs, nightly builds, or QC pipelines. The macro can also capture the return code from the X or SYSTASK command, write it to the SAS log, and optionally halt the workflow if packaging fails. This approach keeps the packaging logic external to SAS but still enables end-to-end orchestration from a single

driver program, making it ideal for studies where all deliverables must be generated and validated through a controlled SAS execution path.

Example SAS macro:

```
%macro run_tlf_packager(folder=,
                        outpdf=TLFs.pdf,
                        delete_conv=N,
                        wait_approval=Y);

    %local folder_q outpdf_q delete_flag wait_flag batfile cmd;

    /* Quote parameters to handle spaces in paths */
    %let folder_q    = "%sysfunc(dequote(&folder))";
    %let outpdf_q    = "%sysfunc(dequote(&outpdf))";
    %let delete_flag = %upcase(&delete_conv);
    %let wait_flag   = %upcase(&wait_approval);

    /* Path to the batch launcher (or direct Python call if preferred) */
    %let batfile = C:\bsp_apps\TLF_Packager\Run_TLF_Packager.bat;

    /* Build the full command line */
    %let cmd = "%superq(batfile)" &folder_q &outpdf_q &delete_flag &wait_flag;

    options noxwait noxsync;
    x &cmd;

%mend run_tlf_packager;
Typical usage from a SAS run-all program:
%run_tlf_packager(
    folder      = P:\Projects\Study123\Output,
    outpdf      = Study123_TLFs.pdf,
    delete_conv = Y,
    wait_approval = Y
);
```

Here again, the macro parameters map directly to the underlying tool options:

1. folder – folder_path
2. outpdf – output_pdf_name
3. delete_conv – delete_converted_pdfs flag (Y/N)
4. wait_approval – wait_for_user_approval flag (Y/N)

The SAS log will show the constructed X command and any messages echoed from the batch/Python process. A screenshot (for example, Figure 5) can show the macro call in a SAS program and the corresponding log lines, demonstrating how the TLF packaging step is integrated into the wider analysis workflow.

BENEFITS AND IMPACT

The TLF_PACKAGER tool delivers substantial operational gains by automating a step that traditionally demands extensive manual effort. Preparing a final bookmarked PDF from mixed RTF, DOCX, and PDF outputs often requires multiple rounds of sorting, renaming, and editing using desktop tools. By extracting titles directly from each file, generating a structured Excel review sheet, and enabling user-led refinement of bookmark text, the tool eliminates inconsistencies and removes several opportunities for human error. Tasks that previously took hours—even for experienced programmers—can now be completed within minutes with uniform accuracy across studies and deliverables.

Another significant advantage is improved reproducibility and audit readiness. Every run follows a deterministic workflow, ensuring the same ordering logic, title extraction rules, and bookmark formatting each time. Whether

packaging is done for a CSR, DSMB, DMC, or multiple interim updates, the resulting document always uses standardized bookmark hierarchy and a clean, clickable Table of Contents. Reviewers—statisticians, medical writers, and regulatory teams—benefit from a consistent navigation structure that reduces cognitive load and speeds up the document review cycle.

The tool also supports seamless integration into existing SAS-driven study pipelines. A lightweight SAS macro wrapper can trigger the Python engine automatically at the end of a run-all program, enabling a true single-step workflow. As soon as programmers refresh TLF outputs, the packaged PDF is generated without requiring any additional manual action. For studies with frequent reruns, this eliminates delays, reduces hand-offs, and ensures the packaged output is always synchronized with the latest results. Teams can choose between interactive Excel approval or using a previously reviewed version for continuous refresh cycles, making the tool equally useful for both exploratory reruns and final submission preparation.

Finally, TLF_PACKAGER adapts to diverse user environments by supporting three execution modes: direct Python CLI, an interactive batch launcher for non-technical users, and SAS X-command integration for automated pipelines. This flexibility ensures adoption across different roles and experience levels, making the tool a reliable component of standardized, scalable clinical reporting workflows.

CONCLUSION

TLF_PACKAGER transforms a traditionally manual and error-prone step in clinical reporting into a fast, consistent, and fully reproducible workflow. By automating title extraction, enabling structured Excel-based review, and generating a merged PDF with standardized bookmarks and a clickable Table of Contents, the tool streamlines TLF packaging for CSR, DSMB, and interim deliverables. Its flexibility—running via Python, batch script, or SAS—allows seamless integration into existing study pipelines, supporting both automated batch execution and interactive review when required. This approach reduces effort, minimizes variability, and ensures high-quality, submission-ready outputs with every run.

REFERENCES

PHUSE AI Guidance Working Group. (2025). PHUSE guidance on referencing the use of AI in PHUSE Connect papers and Working Group white papers. PHUSE.

OpenAI. (2025). ChatGPT (GPT-4o, 2 July version) [Large language model]. https://chat.openai.com/

ACKNOWLEDGMENTS

The complete conceptualization, Python implementation, algorithm design, and technical development of the TLF_PACKAGER— including title extraction, Excel-driven review logic, PDF merging, bookmark generation, and Table of Contents automation—were fully authored by Manivannan M. Generative AI tools (ChatGPT, OpenAI, 2025) were used solely to enhance written clarity, ensure consistency in documentation style, and improve readability of this paper. No aspects of the programming logic, architectural decisions, or technical problem-solving were performed by AI.

RECOMMENDED READING

Manivannan M. (2025). TLF_Packager_CLI – Python-based TLF Packaging Utility [Source code]. GitHub. https://github.com/manivannan-mathiazhagan/TLF_Packager_CLI

CONTACT INFORMATION

Your comments and questions are welcome. Please contact the author at:

Author: Manivannan Mathialagan
Company: Veristat
Email: manivannan.mathialagan@veristat.com
LinkedIn: https://www.linkedin.com/in/manivannan-mathialagan
GitHub Repository (Tool Source Code): https://github.com/manivannan-mathiazhagan/TLF_Packager_CLI