PharmaSUG 2025 - Paper OS-290

Packing PDF TFL with Table of Contents and Bookmarks Using Python

Jun Yang and Yan Moore, Avidity Biosciences

ABSTRACT

Packaging tables, listings, and figures (TLFs) is a common requirement in the pharmaceutical industry. Rich Text Format (RTF) is widely used for generating individual reports, as SAS offers powerful and flexible functions for RTF output. Consequently, RTF is often preferred for combining outputs, and numerous standard programs and macros have been developed for this purpose.

However, RTF has certain limitations. Its display may vary across different operating systems and even between versions of Microsoft Word. Additionally, RTF files can be thousands of pages long, leading to large file sizes that are slow to open, browse, and transfer.

Portable Document Format (PDF) addresses these challenges. A converted and merged PDF file offers a smaller, more consistent, and manageable solution.

This paper introduces an alternative approach to converting RTF outputs into a single, well-structured PDF file. It has two options to generate a combined pdf with TOC. The first option analyzes and extracts key titles from the PDF content to generate a table of contents and bookmarks. The second option reads TLF metadata file including the order of RTF files, filenames, and corresponding titles. Leveraging Python's extensive developer community and rich set of libraries, this approach enables the efficient creation of a refined and well-formatted output.

INTRODUCTION

The PDF format is a widely accepted standard for document sharing, compatible with various operating systems, and supported by commonly installed reading tools.

This presentation explores an alternative approach: the development of a customized, automated document generation system requiring basic programming skills and minimal resources.

This complex project involves UI design, data integrity checks, and more. Specifically, we will demonstrate how to convert multiple RTF files, automatically extract titles, merge them into a single PDF, generate a bookmark outline, and create a table of contents with internal links using the open-source libraries pypdf and fitz.

PRIOR WORK

There are numerous discussions on converting RTF files to PDFs using languages such as SAS, R, Java, and Python. However, most focus solely on data format conversion.

Our approach offers several advantages:

1. **Minimal Dependencies** – It relies only on Python, Microsoft Word, and a few installable packages, eliminating the need for licensed software like SAS.
2. **Simplicity & Ease of Implementation** – Designed for straightforward setup and use.
3. **User-Friendly Execution** – A single-click solution for seamless processing.

4. **Enhanced Functionality** – Parses RTF files, extracts key titles, embeds customized text, and generates a bookmark outline and TOC.

WALKTHROUGH

PRELIMINARIES

Instead of relying solely on the Python standard library, we will install two additional packages along with Microsoft Word (which is typically pre-installed with Microsoft Office).

1. **Microsoft Word** – Efficiently recognizes the RTF format and converts it to PDF.
2. **pypdf** – Handles various PDF operations, including reading, appending, transforming, extracting text and images, merging PDFs, and creating bookmarks.
3. **fitz** – A powerful library for generating a Table of Contents (TOC) with internal links.

Since Microsoft Word is already installed, we only need to install the required Python packages. Use the following command in the terminal:

```
pip install pypdf PyMuPDF
```

Enter the following into the main.py as a program skeleton. We will show the rest of the code throughout the walkthrough:

```
import math
import win32com.client  # pywin32 for rft file conversion
from pypdf import PdfReader, PdfWriter  # For pdf merge and bookmark outline
import fitz  # PyMuPDF for creating TOC with internal link
import shutil  # for data file manipulation
import os
from pathlib import Path  # for checking data file path and folder path
import logging  # for logging operation details
import utility as utl  # for importing functions in utility class
```

```
1 usage
def generate_pdf(input_folder_path, output_file_path, pi_file_path=None):
    pass


def convert_rtf_to_pdf(word_file, output_pdf_file):
    pass


def merge_pdfs(pdf_folder_path, middle_file, output_file_path):
    pass


def extract_text_pypdf2(pdf_path, search_text):
    pass


def create_toc_page(toc_entries, middle_file, output_pdf):
    pass


# Build the PDF and output as a file.
if __name__ == '__main__':
    input_folder_path=r'dummy folder'
    output_file_path=r'dummy output file'
    pi_file_path=r'dummy matedata file'
    generate_pdf(input_folder_path,output_file_path,pi_file_path)
    print('Outputting final document.')
```

While building the demo, you can test the program by running from the command line:

python main.py

CONVERTING RTFS

We can accomplish data file conversion very simply. Change the convert_rft_to_pdf to below:

```python
def convert_rtf_to_pdf(word_file, output_pdf_file):
    word = None
    try:
        # Initialize Word Application (in the background)
        word = win32com.client.Dispatch("Word.Application")
        word.Visible = False  # Hide Word Application

        # Open the RTF document
        doc = word.Documents.Open(word_file)

        # Save as PDF
        doc.SaveAs(output_pdf_file, FileFormat=17)  # FileFormat=17 corresponds to PDF
        doc.Close()
        # print(f"Successfully converted {word_file} to {output_pdf_file}")
        logging.info(f"Successfully converted {Path(word_file).name}.")
        return True
    except Exception as e:
        logging.error(f"Error converting {Path(word_file).name}: {e}")
        return False
    finally:
        # Ensure Word is quit if it was opened
        if word:
            word.Quit()
        return True
```

After running the function, each individual RTF file will be converted into a PDF and saved at the specified output path. If you test the program at this stage, you should obtain a properly formatted PDF file.

Before moving on, let's break down the code to ensure a clear understanding of its functionality.

```
word = None
```
declare the word variable and set it None.

```
word = win32com.client.Dispatch("Word.Application")
```
Next, we need to instantiate the win32com class to call Microsoft word application.

```
word.Visible = False  # Hide Word Application
```
As preferred, we would like to run conversion in the background, so the word will be set to invisible.

```
doc = word.Documents.Open(word_file)
```
Open a data file supporting by Microsoft and cache into memory.

```
doc.SaveAs(output_pdf_file, FileFormat=17)  # FileFormat=17 corresponds to PDF
```
Convert it to pdf format using code 17. If you are interested in other formats, please go to Microsoft website to check.

```
doc.Close()
```
Once conversion is completed, destroy the class and clean the memory and return True for further usage.

```
except Exception as e:
    logging.error(f"Error converting {Path(word_file).name}: {e}")
    return False
finally:
    # Ensure Word is quit if it was opened
    if word:
        word.Quit()
    return True
```

in case of any unexpected issues, it will throw out the error messages and force it to quit.

EXTRACTING TEXT

Now, let's assume we want to extract the content from the top three rows on the first page of a converted PDF and use it as a title. Fortunately, PdfReader offers a mechanism to read pages and extract their content.

By leveraging this functionality, we can retrieve text from the first page, concatenate the relevant portions, and format them as a title for the document. Let's explore how to achieve this.

```
def extract_text_pypdf2(pdf_path, search_text):
    pdf_reader = PdfReader(pdf_path)

    for page_num, page in enumerate(pdf_reader.pages):
        text = page.extract_text()
        for key_text in search_text:
            if key_text in text:
                # print(f"Found on Page {page_num + 1}:")
                lines = text.split("\n")
                for i, line in enumerate(lines[:-2]):
                    if key_text in line:
                        # print(f"  {line}")
                        return line.strip() + ' ' + lines[i + 1].strip() + ' ' + lines[i + 2].strip()
    return ''
```

```
pdf_reader = PdfReader(pdf_path)
```

The initial step is to create an instance of the PdfReader class. This class, provided by the pypdf library, serves as an interface for reading PDF files. As we will observe, it significantly simplifies common tasks.

```
for page_num, page in enumerate(pdf_reader.pages):
```

Next, we need to loop over each page in a pdf file.

```
text = page.extract_text()
    for key_text in search_text:
        if key_text in text:
            # print(f"Found on Page {page_num + 1}:")
            lines = text.split("\n")
            for i, line in enumerate(lines[:-2]):
                if key_text in line:
                    # print(f"  {line}")
                    return line.strip() + ' ' + lines[i + 1].strip() + ' ' + lines[i +
2].strip()
return ''
```

Within the loop, the primary task is to scan the content and identify lines containing keywords such as "figure," "list," or "table." Once such a line is detected, the subsequent two lines are read and

concatenated to form a title, as they typically represent a standard header in an RTF file. If a title is identified, it is returned; otherwise, a blank value is returned.

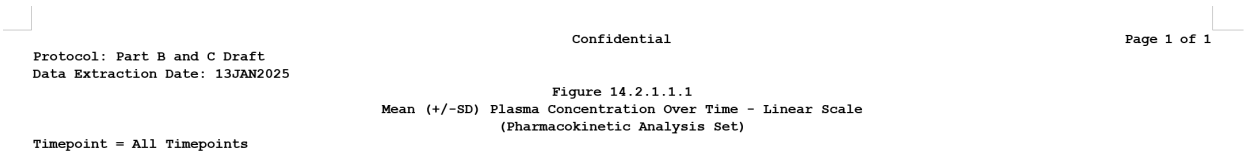

Figure 1: An example of the header in rft file.

MERGING PDFS

We can leverage the PdfWriter interface from the pypdf library to efficiently perform PDF merging operations. Below is an outline of the code used to accomplish this task.

```
def merge_pdfs(pdf_folder_path, middle_file, output_file_path):
    try:
        item_list = os.scandir(pdf_folder_path)
        if not item_list: return

        pdf_writer = PdfWriter()
        name_list = []

        for item in item_list:
            if not item.name.startswith('~'):
                name_list.append(item.name)

        name_list.sort(key=utl.natural_key)
        bookmark_dict = dict()
        # Table of Contents for Bookmark
        pdf_writer.add_outline_item('Table of Contents', 0)

        page_number = 0
        for file in name_list:
            file_path = os.path.join(pdf_folder_path, file)
            pdf_reader = PdfReader(file_path)

            title = extract_text_pypdf2(file_path, ['Table', 'Figure', 'Listing'])
            bookmark_dict[title] = page_number
            for page_num in range(len(pdf_reader.pages)):
                pdf_writer.add_page(pdf_reader.pages[page_num])  # Add each page to the output PDF
                page_number += 1

            pdf_writer.add_outline_item(title, bookmark_dict[title])  # PyPDF2 uses 0-based indexing

        with open(middle_file, "wb") as out_pdf:
            pdf_writer.write(out_pdf)
        # print(f"Merged PDFs saved as {middle_file}")
        logging.info('Merge PDFs completed.')

        create_toc_page(bookmark_dict, middle_file, output_file_path)
    except Exception as e:
        logging.error(f"Error Merging PDFs: {e}")
    finally:
        return
```

```
item_list = os.scandir(pdf_folder_path)
if not item_list: return
```

First, the directory specified by pdf_folder_path is accessed using os.scandir from Python's standard library. All file metadata within the directory is then read into item_list. If item_list is empty, it indicates that the folder contains no files, and the process is terminated.

```
pdf_writer = PdfWriter()
name_list = []
```

Next, instantiate an instance of the PdfWriter class and declare a list of name_list.

```
for item in item_list:
    if not item.name.startswith('~'):
        name_list.append(item.name)
```

Since the list of data file names in the specified folder is obtained in real time, a for loop is used to filter and include only valid RTF file names, excluding any temporary files that may be in the process of being converted.

```
name_list.sort(key=utl.natural_key)
```

It is important to sort the RTF files in ascending numeric-alphabetic order, considering multi-digit numbers within the filenames. This ensures that files such as "file2.rtf" correctly after "file10.rtf", preserving the intended sequence.

```
bookmark_dict = dict()
# Table of Contents for Bookmark
pdf_writer.add_outline_item('Table of Contents', 0)
```

Declare the book_dict dictionary to construct the Table of Contents (TOC). Since the 'Table of Contents' is always the first entry in the bookmarks, it should be added initially with a default page number of 0.

```
page_number = 0
for file in name_list:
    file_path = os.path.join(pdf_folder_path, file)
    pdf_reader = PdfReader(file_path)

    title = extract_text_pypdf2(file_path, ['Table', 'Figure', 'Listing'])
    bookmark_dict[title] = page_number
    for page_num in range(len(pdf_reader.pages)):
        pdf_writer.add_page(pdf_reader.pages[page_num])  # Add each page to the output
PDF
        page_number += 1

    pdf_writer.add_outline_item(title, bookmark_dict[title])  # PyPDF uses 0-based
indexing
```

The page_number variable is used to keep track of the current page number for both the Table of Contents and bookmarks. It is incremented by 1 each time a page from an individual PDF file is merged into the final combined PDF.

The first for loop iterates through each converted PDF file. For each file, the full path is constructed using os.path.join, and the content of the PDF is read. The text from the top three rows of the first page is extracted to form a title, which is then added to the book_dict dictionary along with the current page number.

The second for loop appends each page from the individual PDF into the merged PDF document and increments page_number accordingly. Finally, the add_outline_item method from pdf_writer is used to create the bookmark outline for navigation.

```
with open(middle_file, "wb") as out_pdf:
    pdf_writer.write(out_pdf)
```

After all the steps are completed, write the merged file out at the specified path using pdf_writer.write.

```
create_toc_page(bookmark_dict, middle_file, output_file_path)
```

Finally, call create_toc_page function to add TOC pages into the merged pdf file.

CREATE TOC

```
def create_toc_page(toc_entries, middle_file, output_pdf):
    # Open the PDF
    try:
        font_path = "C:/Windows/Fonts/arial.ttf"
        doc = fitz.open(middle_file)

        landscape_size = fitz.paper_size("letter")  # Get default letter size
        landscape_size = (landscape_size[1], landscape_size[0])  # Swap width & height
        lines = 20
        custom_font = "F0"
        header_size = 16
        content_size = 8
        for j in range(math.ceil(len(toc_entries) / lines)):
            # Create a new blank TOC page
            toc_page = doc.new_page(j, width=landscape_size[0], height=landscape_size[1])  # Insert at the beginning

            toc_page.insert_font(fontname=custom_font, fontfile=font_path)

            left_margin = 40
            right_margin = landscape_size[0] - 40  # Keep space on the right
            y_position = 50  # Start position (Top)

            # Add "Table of Contents" heading
            start_point = (left_margin, y_position + 5)
            end_point = (right_margin, y_position + 5,)
            toc_page.insert_text((left_margin, y_position), "Table of Contents", fontsize=header_size,
                                 fontname=custom_font)
            toc_page.draw_line(start_point, end_point, width=1)

            y_position += 40  # Starting Y position for TOC entries

            # Function to add dotted leader
            def add_dotted_leader(title, start_x, end_x, y):
                font = fitz.Font(fontfile=font_path)
                text_width = font.text_length(title, content_size)
                dot_start_x = start_x + text_width + 10  # Start position for dots
                dots = "." * int((end_x - dot_start_x) / 3)  # Generate dots dynamically
                toc_page.insert_text((dot_start_x, y), dots)
```

```
        def draw_right_aligned_text(right, y, text):
            # Load font
            font = fitz.Font(fontfile=font_path)
            # Measure text width using Font object
            text_width = font.text_length(text, content_size)
            x_position = right - text_width
            # Insert text at calculated position
            toc_page.insert_text((x_position, y), text, fontsize=content_size, fontname=custom_font)

        keys = list(toc_entries.keys())
        for i in range(j * lines, min(j * lines + lines, len(keys))):
            key = keys[i]
            page = toc_entries.get(key) + 1
            toc_page.insert_text((left_margin, y_position), key, fontsize=content_size, fontname=custom_font)
            add_dotted_leader(key, left_margin, right_margin - 25, y_position)
            draw_right_aligned_text(right_margin, y_position, str(page))
            link_rect = fitz.Rect(left_margin, y_position - 10, right_margin,
                                  y_position + 7)  # Define clickable area
            toc_page.insert_link({'kind': 1, 'from': link_rect, 'type': 'goto', 'page': page + j, 'zoom': 0.0})
            y_position += 25

    # Save the updated PDF with the TOC page
    doc.save(output_pdf)
    logging.info(f"New PDF with TOC page created: {output_pdf}")
except Exception as e:
    logging.error(f"Error Creating TOC: {e}")
```

This part of the process is responsible for generating the Table of Contents (TOC) pages and appending them to the beginning of the merged PDF file. A key challenge is that, unlike TOC features in word processors, PDF libraries don't provide a built-in method to link a specific text region (e.g., a line containing a title and page number) directly to a corresponding page in the document.

To address this, a custom algorithm is implemented to format each TOC entry such that the title is aligned to the left margin, the page number is aligned to the right margin, and a dotted line is placed in between for visual clarity. This mimics the familiar appearance of traditional TOC layouts.

The provided code demonstrates the specifics of this formatting and placement logic. Naturally, you're encouraged to adapt the layout, font, spacing, or other style elements to better suit your organization's formatting standards or branding.

Let's walk through the code step by step for a detailed explanation.

```
font_path = "C:/Windows/Fonts/arial.ttf"
doc = fitz.open(middle_file)

landscape_size = fitz.paper_size("letter")  # Get default letter size
landscape_size = (landscape_size[1], landscape_size[0])  # Swap width & height
lines = 20
custom_font = "F0"
header_size = 16
content_size = 8
```

To customize the font, specify the font file path and register it as "F0". Using fitz.open, load the merged PDF to enable internal linking. Create TOC pages in **letter size** with **landscape** orientation by swapping width and height. Limit each TOC page to **20 lines** for readability. Set the **header font size** to 16 and **content font size** to 8 for a clean, consistent layout.

```
for j in range(math.ceil(len(toc_entries) / lines)):
```

The number of TOC pages depends on the total number of TOC entries. To determine the required number of pages, use the ceil function to divide the total number of entries by the maximum number of lines per page.

```
toc_page = doc.new_page(j, width=landscape_size[0], height=landscape_size[1])  # Insert
at the beginning

toc_page.insert_font(fontname=custom_font, fontfile=font_path)

left_margin = 40
right_margin = landscape_size[0] - 40  # Keep space on the right
y_position = 50  # Start position (Top)
```

Create a blank page using the predefined size and insert it at the beginning of the document. Apply customized font to this page. Define the left and right margins and set the starting vertical position near the top of the page to begin placing content.

```
start_point = (left_margin, y_position + 5)
end_point = (right_margin, y_position + 5,)
toc_page.insert_text((left_margin, y_position), "Table of Contents",
fontsize=header_size,
                     fontname=custom_font)
toc_page.draw_line(start_point, end_point, width=1)

Create the TOC header section by defining start and end coordinates. Insert "Table of
Contents" at a set position using the custom font and header size, then draw a horizontal
line below it as a separator.

y_position += 40  # Starting Y position for TOC entries
```

Move the y position to 40 for the first entry of TOC.

```

def add_dotted_leader(title, start_x, end_x, y):
    font = fitz.Font(fontfile=font_path)
    text_width = font.text_length(title, content_size)
    dot_start_x = start_x + text_width + 10  # Start position for dots
    dots = "." * int((end_x - dot_start_x) / 3)  # Generate dots dynamically
    toc_page.insert_text((dot_start_x, y), dots)
```

The add_dotted_leader function draws dots between a title and its page number. It estimates the space based on the font and size, sets the start position, dynamically generates a dot string to fit the gap, and inserts it into the page.

```
def draw_right_aligned_text(right, y, text):
    # Load font
    font = fitz.Font(fontfile=font_path)
    # Measure text width using Font object
    text_width = font.text_length(text, content_size)
    x_position = right - text_width
    # Insert text at calculated position
    toc_page.insert_text((x_position, y), text, fontsize=content_size,
fontname=custom_font)
```

The draw_right_aligned_text function draws a right-aligned page number. It first loads the custom font, measures the text width, calculates the start position based on the right margin, and then inserts the page number into the TOC page.

```
keys = list(toc_entries.keys())
for i in range(j * lines, min(j * lines + lines, len(keys))):
    key = keys[i]
    page = toc_entries.get(key) + 1
    toc_page.insert_text((left_margin, y_position), key, fontsize=content_size,
fontname=custom_font)
    add_dotted_leader(key, left_margin, right_margin - 25, y_position)
    draw_right_aligned_text(right_margin, y_position, str(page))
    link_rect = fitz.Rect(left_margin, y_position - 10, right_margin,
                          y_position + 7)  # Define clickable area
    toc_page.insert_link({'kind': 1, 'from': link_rect, 'type': 'goto', 'page': page + j,
'zoom': 0.0})
    y_position += 25
```

To split TOC entries across multiple pages, first retrieve the list of keys from the TOC dictionary. Use a for loop to iterate through the entries in groups of 20. For each entry:

1. Get the title and corresponding page number.
2. Insert the title at a specific coordinate.
3. Add dotted leaders and the page number.
4. Define a rectangle area covering the title and page number.
5. Create a link from this rectangle to the target page in the merged PDF.
6. Move down by 25 pixels to position the next entry.

```
doc.save(output_pdf)
```

After all steps are completed, save the merged pdf to pre-define the path.

GENERATE PDF

```
def generate_pdf(input_folder_path, output_file_path, pi_file_path=None):
    try:
        output_folder_path = os.path.dirname(output_file_path)

        pdf_path = os.path.join(output_folder_path, 'temp')
        if not os.path.exists(pdf_path):
            os.makedirs(pdf_path)

        middle_file_path = os.path.join(pdf_path, 'merged.pdf')

        item_list = os.scandir(input_folder_path)
        if not item_list: return

        for item in item_list:
            if not item.name.startswith('~'):
                file_path = os.path.join(input_folder_path, item.name)
                if item.name.lower().endswith('.rtf'):
                    pdf_path_file = os.path.join(pdf_path, item.name.split('.')[0] + '.pdf')
                    if not convert_rtf_to_pdf(file_path, pdf_path_file):
                        logging.error('Generating the Combined PDF is failed.')
                        return

            merge_pdfs(pdf_path, middle_file_path, output_file_path)
        logging.info('Generate the Combined PDF completely.')

        pdf_path = Path(pdf_path)
        shutil.rmtree(pdf_path)
    except Exception as e:
        logging.error(f"An error occurred in generating: {e}")
```

The generate_pdf function creates a temporary folder to store converted PDFs, then calls the merge_pdfs function to merge them, generate bookmark outlines, and build the TOC. Once processing is complete, it deletes the temporary folder to clean up.

```
output_folder_path = os.path.dirname(output_file_path)

pdf_path = os.path.join(output_folder_path, 'temp')
if not os.path.exists(pdf_path):
    os.makedirs(pdf_path)

middle_file_path = os.path.join(pdf_path, 'merged.pdf')
```

First, extract the output folder path and append "temp" to define the temporary folder path. If the folder doesn't exist, create it. Then, define the path for the merged PDF file within the output folder.

```
item_list = os.scandir(input_folder_path)
if not item_list: return

for item in item_list:
    if not item.name.startswith('~'):
        file_path = os.path.join(input_folder_path, item.name)
        if item.name.lower().endswith('.rtf'):
            pdf_path_file = os.path.join(pdf_path, item.name.split('.')[0] + '.pdf')
            if not convert_rtf_to_pdf(file_path, pdf_path_file):
                logging.error('Generating the Combined PDF is failed.')
                return
```

Retrieve the list of RTF files. If the list is empty, terminate the process. Otherwise, iterate through each RTF file, skipping any file that starts with '~', as these are likely temporary files created by Word. If any conversion fails, stop execution and display an error message.

```
merge_pdfs(pdf_path, middle_file_path, output_file_path)
logging.info('Generate the Combined PDF completely.')

pdf_path = Path(pdf_path)
shutil.rmtree(pdf_path)
```

Merge all converted pdfs and delete the temporary folder. Show the completion message.

OUTPUT

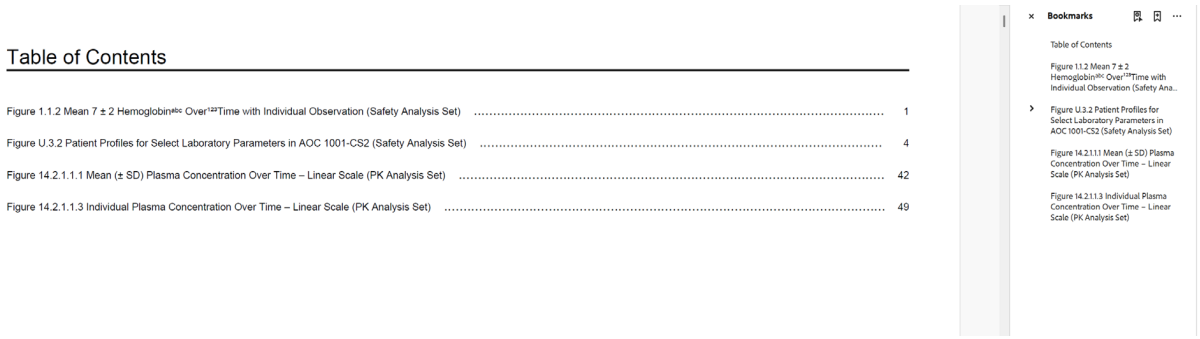

Figure 2: An example of a merged pdf file.

USER INTERFACE

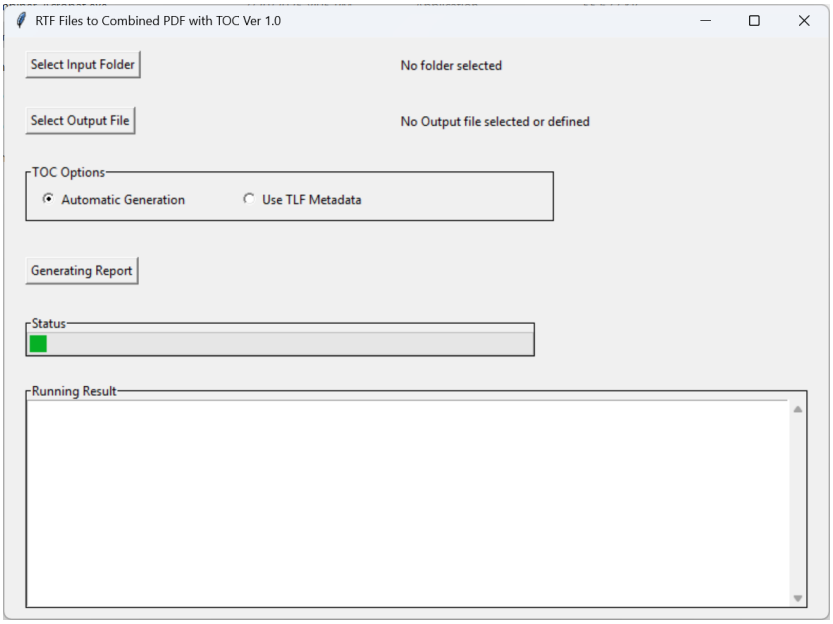

Figure 3: This is the official version—it not only includes the automatic generation described above but also features a customized “Use TLF Metadata” mode.

Figure 4: When clicking User TLF Metadata radio button, selecting PI data file option will appear.

Type	Shell	HeaderDate	OutputNumber	Title	Footnote	OutputName	OutputFileName
F			U.3.1	Mean (*ESC*){super [abc]}Hemoglobin(*ESC*){super [223]}Over Time		Figure U.3.1	f03-01-lb-mean-scatt-saf-cs1-cs2.rtf
F			U.3.2	Patient Profiles 7 (*ESC*){unicode '03BB'x} 2 (*ESC*){unicode '03BE'x}		Figure U.3.2	f03-02-lb-pp-selected-cs1-cs2.pdf
F			14.2.1.1.3	Individual Plasma Concentration Over Time – Linear Scale (PK Analysis Set)		Figure 14.2.1.1.3	f-14-2-1-1-3.rtf
F			14.2.1.1.1	Mean (± SD) Plasma Concentration Over Time – Linear Scale (PK Analysis Set)		Figure 14.2.1.1.1	f-14-2-1-1-1.rtf

Figure 5: The example of the metadata file below to customize number, title and merging order

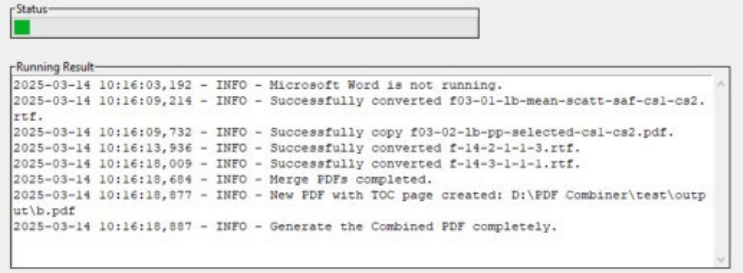

Figure 6: The status and Running Result section can show you detailed progress.

CONCLUSION

We have demonstrated that the software is not only capable of converting RTF files to PDF but also serves as a powerful tool for manipulating PDF documents. In fact, assembling reports can be both straightforward and user-friendly through two available modes: **automation** and **customization**, all made possible using free and open-source libraries. The tool is as reliable as proprietary alternatives, thanks to the PDF format being an ISO standard.

We hope the information provided empowers you to develop tailored solutions that meet your specific needs.

REFERENCES

[1] pypdf https://github.com/py-pdf/pypdf

[2] fitz https://pymupdf.readthedocs.io/en/latest/index.html

[3] openpyxl https://pypi.org/project/openpyxl/

[4] pandas https://pandas.pydata.org/

CONTACT INFORMATION

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

Jun Yang
Avidity Biosciences
jun.yang@aviditybio.com

Yan Moore
Avidity Biosciences
yan.moore@aviditybio.com