PharmaSUG 2026 - Paper ET-381

Bridging the Gap: A Python-Word Integration for Detecting Ghost Page Breaks in SAS-Generated RTFs

Jun Yang and Robert Stemplinger, Organon LLC

ABSTRACT

Generating Tables, Figures, and Listings (TFLs) via SAS® ODS RTF is the industry standard for clinical trial reporting. However, SAS frequently introduces "extra" page breaks or "orphaned" headers due to margin constraints and font rendering discrepancies. Also, customizable solution integrated business logic such as abnormal indentation in the first row to identify page breaks fit various requirements in different scenarios. Manually reviewing hundreds of pages to ensure no empty pages or split tables exist is time-consuming and prone to human error.

This paper introduces a novel Python-based application designed to automate the detection of pagination anomalies. While standard text-extraction libraries often fail to capture the visual "flow" of a document, this tool leverages Microsoft Word's rendering engine as a diagnostic bridge. By utilizing the pywin32 library to interface with the Word COM object, the application programmatically opens the SAS output and analyzes the document's internal pagination structure. The specific mechanism compares expected page-break triggers against Word's actual rendered layout to identify "ghost" breaks—instances where content ends prematurely or headers appear without corresponding data.

The resulting application provides a user-friendly interface for programmers to batch-process study outputs. This automation significantly reduces the Quality Control (QC) burden, ensuring that submission-ready RTFs are free of formatting defects. By combining the data-processing power of Python with the layout intelligence of Word, programmers can achieve a higher level of precision in clinical reporting.

INTRODUCTION

The RTF format is a widely accepted standard for document sharing, compatible with various operating systems, and supported by ubiquitous reading tools. This presentation explores an alternative approach to quality control: the development of a customized, automated issue-detection system and aggregated report that requires only basic programming skills and minimal resources.

The project encompasses user interface design, data integrity validation, and automated analysis. Specifically, we will demonstrate how to parse different sections of each RTF page to detect issues such as blank pages, content overflow, and abnormal indentation. The final tool generates a comprehensive report with page-level references to streamline the remediation process. This solution is implemented using the open-source pywin32 library to bridge the gap between Python and Microsoft Word.

RTF FILE WITH PAGE ISSUES

Table U.1.1
Disposition in Study Number
(All Subjects)

Statistics	5 MG/KG Q6W				10 MG/KG Q8W			All Subjects
	AOC to AOC (N=7)	PBO to AOC (N=3)	De Novo (N=16)	Pooled (N=26)	AOC to AOC (N=10)	PBO to AOC (N=3)	Pooled (N=13)	(N=39)
Age (years), n (%)								
7 to <12	1 (14.3)	0	6 (37.5)	7 (26.9)	6 (60.0)	1 (33.3)	7 (53.8)	14 (35.9)
12 to <18	3 (42.9)	3 (100)	9 (56.3)	15 (57.7)	4 (40.0)	2 (66.7)	6 (46.2)	21 (53.8)
18+	3 (42.9)	0	1 (6.3)	4 (15.4)	0	0	0	4 (10.3)
Age (years)								
n	7	3	16	26	10	3	13	39
Mean	17.4	14.7	12.1	13.8	11.5	12.3	11.7	13.1
SD	4.96	2.52	2.63	4.02	2.55	4.62	2.93	3.79
Median	17.0	15.0	12.0	13.0	11.0	15.0	11.0	12.0
Min, Max	11, 27	12, 17	7, 18	7, 27	8, 16	7, 15	7, 16	7, 27
Race, n (%)								
White	7 (100)	3 (100)	11 (68.8)	21 (80.8)	10 (100)	3 (100)	13 (100)	34 (87.2)
Black or African American	0	0	1 (6.3)	1 (3.8)	0	0	0	1 (2.6)
Asian	0	0	2 (12.5)	2 (7.7)	0	0	0	2 (5.1)
Native Hawaiian or Other Pacific Islander	0	0	0	0	0	0	0	0
American Indian or Alaska Native	0	0	1 (6.3)	1 (3.8)	0	0	0	1 (2.6)
Other	0	0	0	0	0	0	0	0
Multiple	0	0	0	0	0	0	0	0
Not Reported	0	0	1 (6.3)	1 (3.8)	0	0	0	1 (2.6)
Ethnicity, n (%)								

Display 1: The page is missing data at the bottom.

Table U.1.1
Disposition in Study Number
(All Subjects)

Statistics	5 MG/KG Q6W				10 MG/KG Q8W			All Subjects
	AOC to AOC (N=7)	PBO to AOC (N=3)	De Novo (N=16)	Pooled (N=26)	AOC to AOC (N=10)	PBO to AOC (N=3)	Pooled (N=13)	(N=39)
Hispanic or Latino	2 (28.6)	0	0	2 (7.7)	4 (40.0)	0	4 (30.8)	6 (15.4)
Not Hispanic or Latino	5 (71.4)	3 (100)	15 (93.8)	23 (88.5)	6 (60.0)	2 (66.7)	8 (61.5)	31 (79.5)
Not Reported	0	0	1 (6.3)	1 (3.8)	0	1 (33.3)	1 (7.7)	2 (5.1)
Weight (kg)								
Mean	52.06	41.67	51.36	50.43	34.22	64.47	41.20	47.35
Median	51.00	43.60	48.35	49.55	35.90	45.00	36.00	43.90
Min, Max	34.8, 65.9	29.4, 52.0	28.3, 114.5	28.3, 114.5	26.1, 43.9	30.6, 117.8	26.1, 117.8	26.1, 117.8

Display 2: The page is missing a subtitle

PRIOR WORK

Numerous approaches have been proposed for reading and processing RTF files using languages such as SAS, R, Java, and Python. However, it remains challenging to identify a solution that is both simple and effective for detecting specific formatting issues.

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 intuitive use.
3. **User-Friendly Execution** – A single-click workflow enables seamless processing.
4. **Explicit Output Report** – The generated report explicitly identifies the type and location of issues on each page.

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.
2. **Pywin32** – Access to native windows APIs, enabling automation, COM object interaction.
3. **Pandas** – open-source library for data analysis and manipulation.

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

```
pip install pywin32, pandas
```

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

```python
import win32com.client as win32  #pywin32 for processing rtf files
import os  #walkthrough a folder
import pandas as pd  #data analysis and manipulation


def detect_rft_issues(input_file):
    pass


def check_rtf_files(folder_path, output_file):  1 usage
    pass

if __name__ == "__main__":
    input_folder_path = r'dummy folder'
    output_file_path = r'dummy file'
    check_rtf_files(input_folder_path, output_file_path)
    print("Done")
```

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

python main.py

DETECT RTF ISSUES

Specific issues can be identified simply by implementing logic.

```
def detect_rft_issues(input_file):  1 usage
    word = None
    try:
        word = win32.Dispatch("Word.Application")
        word.Visible = False

        doc = word.Documents.Open(input_file)
        doc.Fields.Update()

        overflow_pages = []

        for idx, table in enumerate(doc.Tables):
        # get page numbers for all rows in the table
            row_pages = [row.Range.Information(3) for row in table.Rows]
            # blank page
            if len(set(row_pages)) >1 and row_pages[0]!=row_pages[-1]:
                overflow_pages.append(row_pages[0])

            # indent page
            for r in range(1, table.Rows.Count + 1):
                row = table.Rows(r)
                page = row.Range.Information(3)

                if row.HeadingFormat:  # Skip header rows
                    continue

                row_text = row.Range.Text.rstrip('\r\x07')  # Clean up end characters
```

```
                row_text = row.Range.Text.rstrip('\r\x07')  # Clean up end characters

                if not row_text.strip():  # Skip empty rows
                    continue

                # Check indent
                if row_text.startswith((" ", "\t")):
                    if page not in overflow_pages:
                        overflow_pages.append(page)
                    break

        blank_pages = []
        page_count = doc.ComputeStatistics(2)  # get page number

        for i in range(1, page_count + 1):
            start_range = doc.GoTo(What=1, Which=1, Count=i)
            end_range = doc.GoTo(What=1, Which=1, Count=i + 1)

            if i==page_count:
                page_range = doc.Range(start_range.Start, doc.Content.End)
            else:
                page_range = doc.Range(start_range.Start, end_range.Start)

            body_text = page_range.Text.strip()
            if not body_text:
                blank_pages.append(i)
        doc.Close()
        return overflow_pages, blank_pages
    except Exception as e:
        print(f"Error converting {input_file}: {e}")
    finally:
        # Ensure Word is quit if it was opened
```

```
    if word:
        word.Quit()
```

After running the function, overflow pages and blank pages will be found out and cached into the memory.

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
```

As preferred, we would like to run conversion in the background, so the Word application instance will be set to invisible.

```
doc = word.Documents.Open(input_file)
```

Open the target RTF document in word.

```
doc.Fields.Update()
```

Refresh all fields like page numbers and so on in the opened document.

```
overflow_pages = []
```

Define a list to collect page numbers with overflow or indentation issues.

```
for idx, table in enumerate(doc.Tables):
            row_pages = [row.Range.Information(3) for row in table.Rows]
            if len(set(row_pages)) > 1 and row_pages[0] != row_pages[-1]:
                overflow_pages.append(row_pages[0])
            for r in range(1, table.Rows.Count + 1):
                row = table.Rows(r)  # Access row by 1-based index via COM
                page = row.Range.Information(3)
                if row.HeadingFormat:                       continue
                row_text = row.Range.Text.rstrip('\r\x07')
                if not row_text.strip():
                    continue
                if row_text.startswith((" ", "\t")):
                    if page not in overflow_pages:  # Avoid duplicate page entries
                        overflow_pages.append(page)
                break
```

Within the loop, the program iterates over all tables in the document using a zero-based index. For each table, it first determines the page number associated with every row. It then evaluates whether a page is effectively blank by checking if the table spans multiple pages and whether the first and last rows appear on different pages. If such a condition is detected, the corresponding overflow page number is recorded.

An inner loop is subsequently used to verify the presence of unwanted indentation issues at the row level. The header row—typically containing study metadata—is excluded from this check. For each remaining row, trailing carriage returns and cell-end markers are removed to ensure accurate content evaluation. If the first row on a page exhibits unintended indentation, the page number is flagged accordingly. Otherwise, the page is considered to be correctly formatted.

```
blank_pages = []
```

Define a list to collect page numbers that have no visible content.

```
page_count = doc.ComputeStatistics(2)
```
Get total page count of the document.

```
for i in range(1, page_count + 1):
    start_range = doc.GoTo(What=1, Which=1, Count=i)
    end_range = doc.GoTo(What=1, Which=1, Count=i + 1)

    if i==page_count:
        page_range = doc.Range(start_range.Start, doc.Content.End)
    else:
        page_range = doc.Range(start_range.Start, end_range.Start)

    body_text = page_range.Text.strip()
    if not body_text:
        blank_pages.append(i)
```
The process iterates through each page to determine whether it contains any text content. It first navigates to the beginning of page *i*, then determines the end boundary by moving either to the start of the next page or to the end of the document, depending on the page condition. Once the boundaries are established, the content range for the current page is defined by comparing page numbers. The main content within this range is then extracted and evaluated. If no text is detected, the page is classified as blank, and its page number is recorded.

```
doc.Close()
```
Close the document without saving changes.

```
return overflow_pages, blank_pages
```
Return both issue lists for reporting.

```
except Exception as e:
    print(f"Error converting {input_file}: {e}")
finally:
    # Ensure Word is quit if it was opened
    if word:
        word.Quit()
```
in case of any unexpected issues, it will throw out the error messages and force it to quit.

CHECK RTFs

Next, we iterate through all RTF files within a specified folder and generate an output report summarizing the results. The OS library provides mechanisms to retrieve file names along with their associated metadata. In the following section, we demonstrate how to leverage these capabilities to efficiently process and analyze multiple RTF files.

```
def check_rtf_files(folder_path, output_file):  1 usage
    res = dict()
    data_files = [f for f in os.listdir(folder_path) if not f.startswith('~') and f.endswith('.rtf')]
    for file in data_files:
        file_path = os.path.join(folder_path, file)
        filename = file.strip()
        overflow_pages, blank_pages = detect_rft_issues(file_path)
        res[filename] = {'Overflow Pages': overflow_pages, 'Blank Pages': blank_pages}

    if len(res) > 0:
        df = pd.DataFrame.from_dict(res, orient='index').reset_index()
        custom_headers = ['Filename', 'Overflow', 'Blank Page']
        df.columns = custom_headers
        df.to_excel(output_file, index=False)
```

```
res = dict()
```
Define a dictionary to accumulate results keyed by filename.

```
data_files = [f for f in os.listdir(folder_path) if not f.startswith('~') and
f.endswith('.rtf')]
```
List all RTF files in the folder, excluding temporary files that start with '~'.

```
for file in data_files:
    file_path = os.path.join(folder_path, file)
    filename = file.strip()
    overflow_pages, blank_pages = detect_rft_issues(file_path)
    res[filename] = {'Overflow Pages': overflow_pages, 'Blank Pages': blank_pages}
```
Iterate individual rtf file and check if issues are existed and cache result into the dictionary.

```
if len(res) > 0:
    df = pd.DataFrame.from_dict(res, orient='index').reset_index()
    custom_headers = ['Filename', 'Overflow', 'Blank Page']
    df.columns = custom_headers
    df.to_excel(output_file, index=False)
```
if the dictionary contains any records, the output result file with excel format will be generated at pre-defined filename and file path.

OUTPUT

Filename	Overflow	Blank Page
t_demo1.rtf	[2, 3, 6, 8]	[]
t_demo2.rtf	[1, 4]	[3]
t_teae_socpt_sev1.rtf	[2]	[]
t_teae_socpt_sev2.rtf	[1, 3, 5]	[]

Figure 1: An example of an output result file.

USER INTERFACE

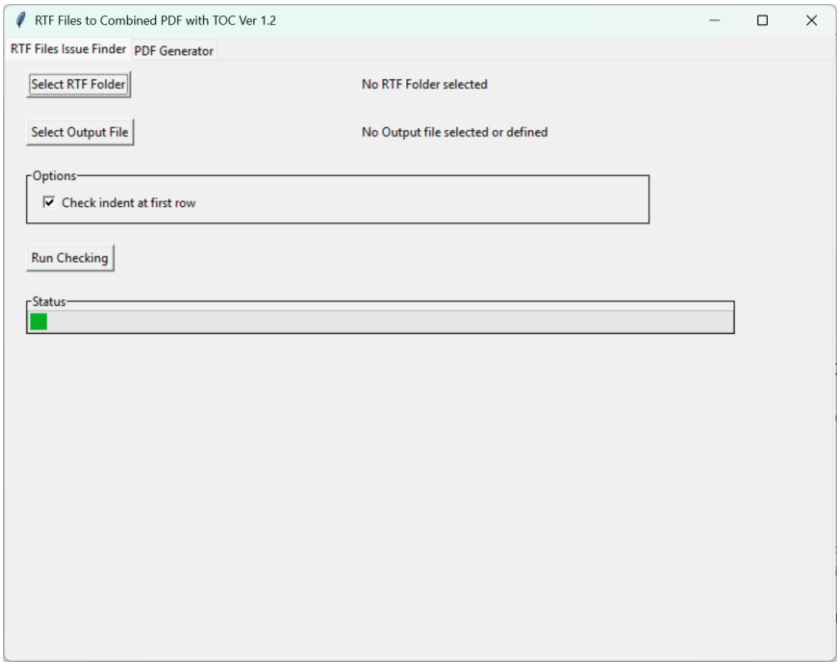

Figure 2: This is the official version—it not only includes the rtf issues detection described above but also features a PDF generator which converts rtfs and combined into one pdf file with TOC (Table of Content with navigable page numbers.

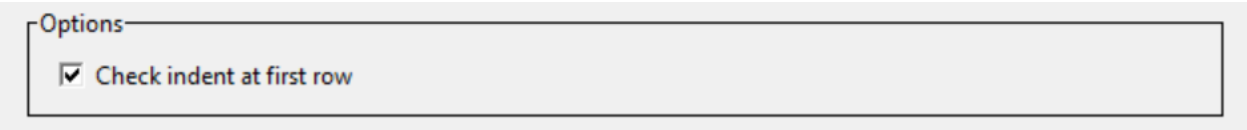

Figure 3: Check indentation at first row is optional.

CONCLUSION

We have demonstrated that the application is not only capable of detecting overflow, blank page and unwanted indentation issues to RTFs but also serves as a powerful tool for consolidating multiple RTFs into a single PDF document. 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 alternative which includes issues checking and document consolidation.

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

REFERENCES

[1] pywin32 https://pypi.org/project/pywin32/

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

CONTACT INFORMATION

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

Jun Yang
Organon LLC
jun.yang@organon.com

Robert Stemplinger
Organon LLC
robert.stemplinger@organon.com