PharmaSUG China 2021 - Paper AD-013

A Tool to Combine TFL Outputs

Jason Wang, WuXi Clinical
Yashan Zhou, WuXi Clinical

ABSTRACT

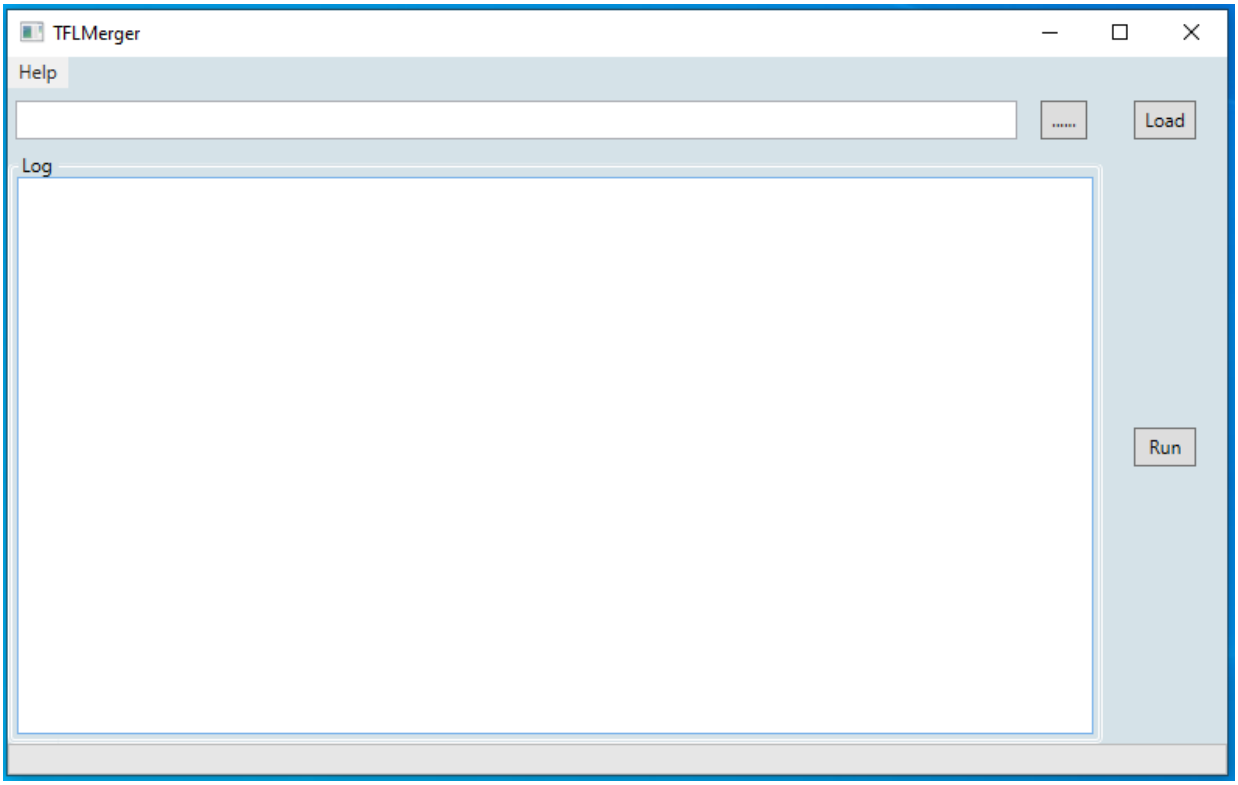
In our daily work, we generate a large number of individual table, figure, listing or some other kind of outputs (RTF or PDF format) by SAS. In most cases, we need to combine the outputs together in order to facilitate the review. There are some approaches to achieve it and each approach has its advantage and disadvantage. This paper introduces a useful tool used for output combination and offers a step-by-step view of it. This tool is a lightweight desktop application developed with C# programming language. In general, we can use this tool to check the pagination issue of the rtf file, convert individual files from RTF format to PDF format, combine individual PDF files into one with bookmark and table of contents. Moreover, this tool has a few desirable features: it can process large files, the size of the combined file is not limited to 500M; it can serialize the page numbers within the combined file; it can perform multiple combination tasks one by one with only one click; it works fine no matter the individual files are in landscape format or portrait format or mixed format.

INTRODUCTION

As a SAS programmer, writing SAS program to generate table, figure and listing in RTF format is part of our daily work. We use RTF because it is a text-based format and it is easy to manipulate, with RTF, we can control the TFL format at a very detailed level. However, RTF file is usually not our destination. Sometimes, we need to convert them to PDF format to meet the regulator's requirement. In other situation, we also need to combine individual outputs (RTF or PDF) into single PDF file.

Converting RTF to PDF is easy. Of course you can manually open the RTF file and save it as a PDF file, however, this is time consuming. You may need a tool to automate this process if you have many individual RTF files. Combining the outputs into a single one is a bit challenging, you may be asked to add bookmark or table of contents. Actually, there are several solutions for it. A solution is concatenating the individual RTF files into a single RTF or DOC/DOCX file and then save it as a pdf file. This is OK if the size of combined rtf/doc/docx file is less than 500M (in this case, you may need to create a few smaller combined RTF files, luckily this is acceptable for FDA). Another solution is to convert each RTF file into individual PDF file and then concatenate all the PDF files into single one. This method does not have the limitation of file size.

Here we would like to introduce a lightweight desktop application (we name it TFLMerger) developed with C# programming language. The key DLL used for this tool is iTextSharp, this is a powerful library for PDF manipulation. The TFLMerger integrates several functions we usually use in our daily work. With this tool, you can check the pagination issue for RTF files, convert the individual RTF files to individual PDF files, combining individual pdf files into single one. The TFLMerger (Display 1) is an executable application with concise user interface. An Excel file (we name it Filelist) which is used to configure TFLMerger and to list the individual files should be used together (Display 2).



Display 1. User Interface of TFLMerger

	A	B	C	D
1	Options	Value	Detail	Notes
2	File Type	RTF	NA	Specify the source file type to be processed, default is RTF.
3	Generate Filelist	Y	NA	Instruct the TFLmERGER to create a Filelist.xlsx based on this config.
4	Check Pagination	Y	NA	Instruct the TFLMerger to check the pagination issue. Only applicable for RTF file.
5	Convert	N	NA	Instruct the TFLMerger to convert the individual RTF/DOC/DOCX to individual PDF.
6	Combine	Y	NA	Instruct the TFLMerger to combine the outputs.
7	Book Mark	Y	NA	Generate book mark for the combined PDF file. Use title column if available, otherwise use file name.
8	TOC	Y	NA	Generate TOC for the combined PDF file. Use "Title" column if available, otherwise use "FileName" column.
9	Language	E	NA	For Chinese TFL outputs, set value to "C" to generate Chinese TOC, otherwise the TOC will be corrupted.
10	Merge Mode	A	NA	Combine all outputs to a single file if "A"; combine outputs by sheets if "S"; combine output by sheets and type if "T".
11	Sheet1	Table	[Path of Table]	specify the name of first sheet that contains the outputs to be processed. Provide output location in detail column.
12	Sheet2	Listing		specify the name of second sheet that contains the outputs to be processed.
13	Sheet3	Figure		specify the name of third sheet that contains the outputs to be processed. Add Sheetx row if needed.

Config

Display 2. Configuration tab in Filelist

METHODOLOGY AND DISCUSSION

PREPARE THE FILELIST.

In order to use this tool, you need to prepare the Filelist at first. The Filelist is an XLSX file that contains a sheet named “Config”, it instructs TFLMerger what to do and how to do. If it is the first time you use this tool, you can download the template from the “Help” menu. The “Config” tab should be like what is showed in Display 2. It has 4 columns, the “Options” column defines the option name, the “Value” column defines the options value (for most of the options, the option value cell embeds a drop-down list), the “Detail” column defines the detailed value if additional information is required, the “Notes” column is a brief description for that option. There are 12 pre-defined options, these options can be classified into 4 categories (highlighted with 4 different colors):

1. File Type. This category only contains 1 option, this is used to specify the source format of your files to be processed. You can select RTF or PDF, actually this can be used to process DOC or DOCX file

since they can also be converted to PDF and then do the combination.

2. Action. This category consist of 4 options used to instruct the tool what to do, it includes 3 core functions and 1 auxiliary function:

   2.1 Generate Filelist. This is the auxiliary function, it instruct the TFLMerger to create file list tabs according to the “Sheet1” to “Sheetn” options. For example, you have table, listing, figure located in 3 separate folders, you can define sheet names in column B and the file locations in column C, see row 2, 3, 12-14 in Display 3. After configuring, you can load this Filelist into TFLMerger and click the “Run” button, it will generate a new Filelist located in the folder specified for “Sheet1”. The new Filelist keeps the “Config” tab from the old one and have 3 more tabs that list all the individual files under each specified folder, namely “Table”, “Listing” and “Figure” (Display 4). Please note if you turn on the “Generate Filelist” option, the following 3 options will be disabled even if you turn on them in the “Config” tab. In the file list tab, such as “Table”, the FileName column includes all the files (filename + extension) located in the specified path. There are another 3 columns, the “Title” column is the title of the output, this is used as bookmark and TOC. The “Type” column is used to define a type for a group of TFLs, for example, “Safety” and “Efficacy”, you can combine the outputs by type. The last column is named “Select”, you can select part of the files to process by filling a “Y”. By default, all values for this column are empty, in this situation, all files are selected.

	A	B	C
1	Options	Value	Detail
2	File Type	RTF	NA
3	Generate Filelist	Y	NA
4	Check Pagination	N	NA
5	Convert	N	NA
6	Combine	N	NA
7	Book Mark	Y	NA
8	TOC	Y	NA
9	Language	C	NA
10	Merge Mode	T	NA
11	Sheet1	Table	C:\Users\jason_wang\Desktop\PharmaSUG2021\test file\table
12	Sheet2	Listing	C:\Users\jason_wang\Desktop\PharmaSUG2021\test file\listing
13	Sheet3	Figure	C:\Users\jason_wang\Desktop\PharmaSUG2021\test file\figure

Config

Display 3. “Sheet1” to “Sheetn” options

	A	B	C	D
1	FileName	Title	Type	Select
2	t-14010101-disp-enrol.rtf			
3	t-14010102-anpop-itt.rtf			
4	t-14010103-pd-itt.rtf			
5	t-140102-demog-saf.rtf			
6	t-14010301-mh-itt.rtf			
7	t-14010302-mh-itt.rtf			
8	t-14010401-cm-itt.rtf			
9	t-14010402-cm-itt.rtf			
10	t-14030101-ex-saf.rtf			
11	t-1403010101-ae-saf.rtf			
12	t-1403010102-teae-saf.rtf			
13	t-1403010103-teaemf-saf.rtf			
14	t-1403010104-teaews-saf.rtf			

Config | Table | Listing | Figure

Display 4. Filelist that contains 3 file list tabs

2.2 Check Pagination. This action is to check the pagination issue for RTF files. Sometimes we may have pagination issue in our RTF output. It is hard to notice the pagination issue by eyes, especially for long listings. This action helps you to find out the pagination issues before combining them.

2.3 Convert. This action is to convert the individual RTF file to individual PDF file. Sometimes you may be asked to provide the individual PDF files. This action instructs the TFLMerger to create individual PDF files in the same folder.

2.4 Combine. This action instructs the TFLMerger to combine the source files. There are several options used for "Combine" action.

3. Combine options. See row 7-10 in "Config" tab. It instructs the TFLMerger how to do the combination for the source files.

   3.1 Book Mark. Set value to "Y" if you want to add bookmark to the combined PDF file.

   3.2 TOC. Set value to "Y" if you want to add TOC to the combined PDF file.

   3.3 If TOC is set to "Y", you need to specify the language that used for TOC. The default value is "E" which stands for English. You can change it to "C" which stands for Chinese. Why the language is necessary for TOC? Because the English font does not support the Chinese characters if you are doing a Chinese study, in this way, you need to tell the PDFMerger that you are using Chinese characters in the TOC. The font used for English is "Arial" and the font used for Chinese is "Simsun".

   3.4 Merge Mode. It defines the merge mode used for combination. There are 3 modes predefined for this option. The "A" mode instruct the PDFMerger to combine all the source files into a single PDF file regardless of the "Sheet", in the example, combine all the tables, listings and figures together into one file. The "S" mode instruct the PDFMerger to combine the source files by the sheets, in the example, it creates 3 combined files for table, listing and figure respectively. The "T" mode instruct the PDFMerger to combine the source files by type which is defined in "Type" column in each file list tab. The "T" mode is useful if you want to combine the outputs by a self-defined subgroup. For example, combine all the safety tables into one and combine all the efficacy tables into one.

4. Sheet options. This is an option group used to define a series of sheets that contains the source files. For example, you can define 3 sheets named "Table", "Listing" and "Figure" and specify the full path for each of them. The option only take effect when both "Value" and "Detail" columns are filled and the full path is a valid path. You can add "Sheet4", "Sheet5"…"Sheetn" per your needs. There is another scenario: all your TFLs are located in the same folder but you want to combine them by table, figure and listing respectively, in this case, you can define one sheet named "TFL" and fill the "Detail" column with TFL path in Filelist, then use the "Generate Filelist" function to generate a new Filelist with a "TFL" tab, after that, you fill the "Type" column with "Table", "Figure" and "Listing". At last, use the "T" mode to combine them.

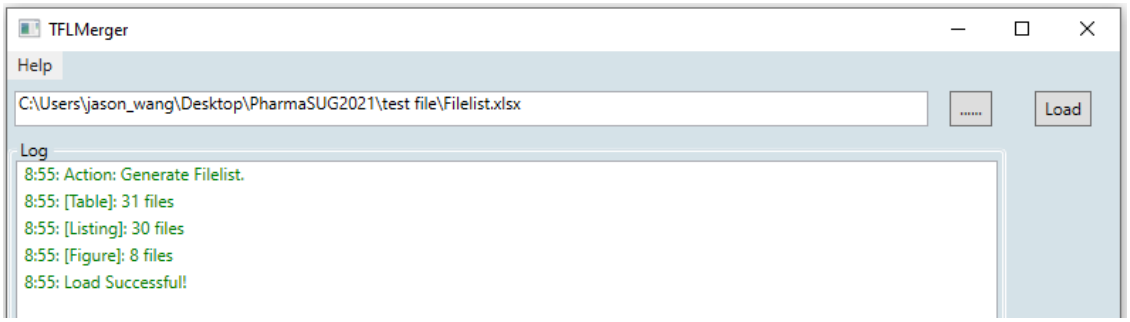
LOAD THE FILELIST

After configuring the "Config" tab, you need to save the file and load it to the PDFMerger.

1. Click the "……" button and select the Filelist. The full path and filename will appear in the textbox. Of course, you can copy the path and filename and then paste to the textbox (you can even type in the textbox, but this is not recommended since it is easy to make typo).

2. By clicking the "Load" button, the TFLMerger loads the Filelist into memory and then analyze the "Config" tab as well as the rest of the file list tab. The log window shows the logs for the "Load" procedure (Display 5). The first row displays the action(s) to be performed, the last row shows you if the "Load" is successful or not, the rows between first row and last row display the details associated with the current action(s). This helps the user to recognize the details that the TFLMerger is going to process. If failed to load the Filelist, the log window shows the reason why it failed, then you can update the "Config" tab accordingly. If the log information displayed is what you exactly wanted, you

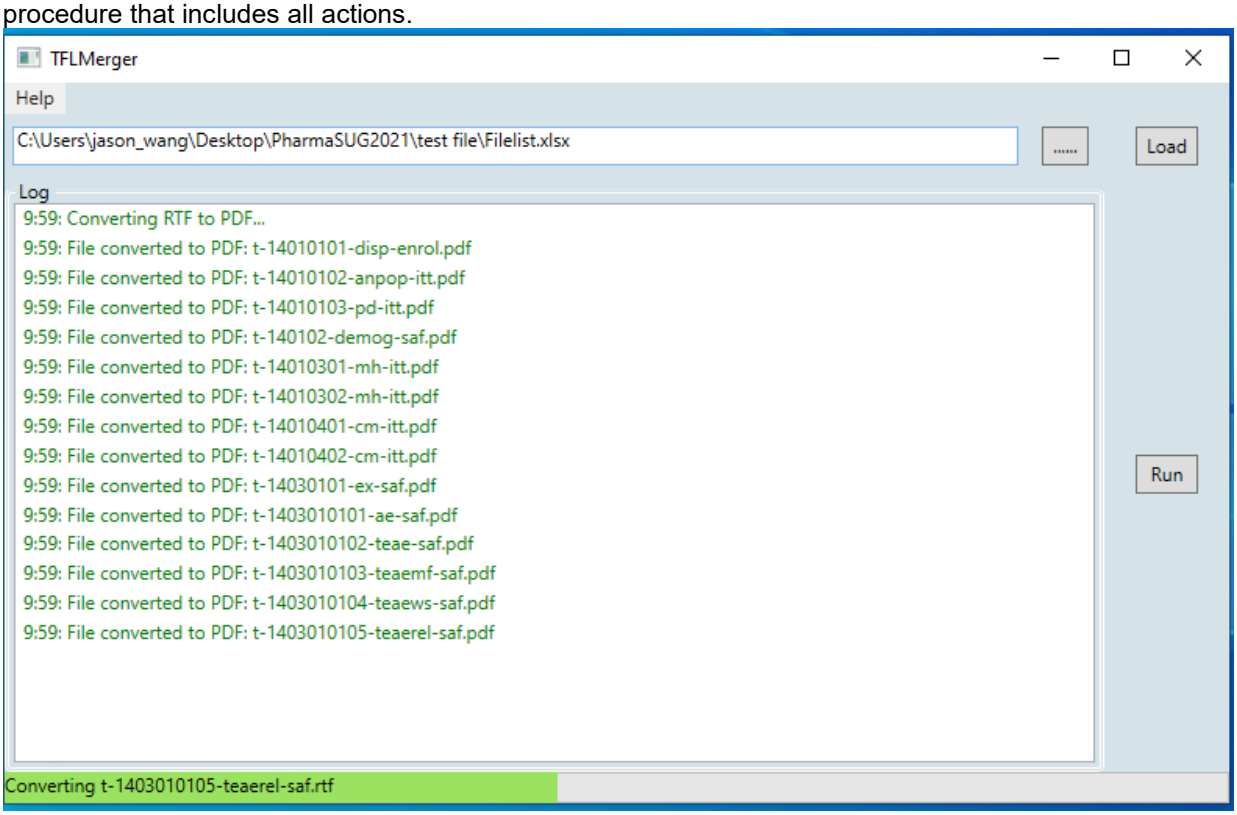
can move to the next step (execute the action), otherwise, if any information is incorrect, you need to check the reason and update the Filelist and load it again.



Display 5. Log window

EXECUTE THE ACTION

If everything is ready, click the “Run” button, it executes the action(s) one by one, the log window shows all the detailed information simultaneously. We also designed a status bar and a progress bar at the bottom of the TFLMerger (Display 6), it shows the progression for the action being processed. If the green bar reached to the end, it means the current action is completed. The TFLMerger initialize the progress bar when next action starts. That means the progress bar is for current action instead of the whole procedure that includes all actions.

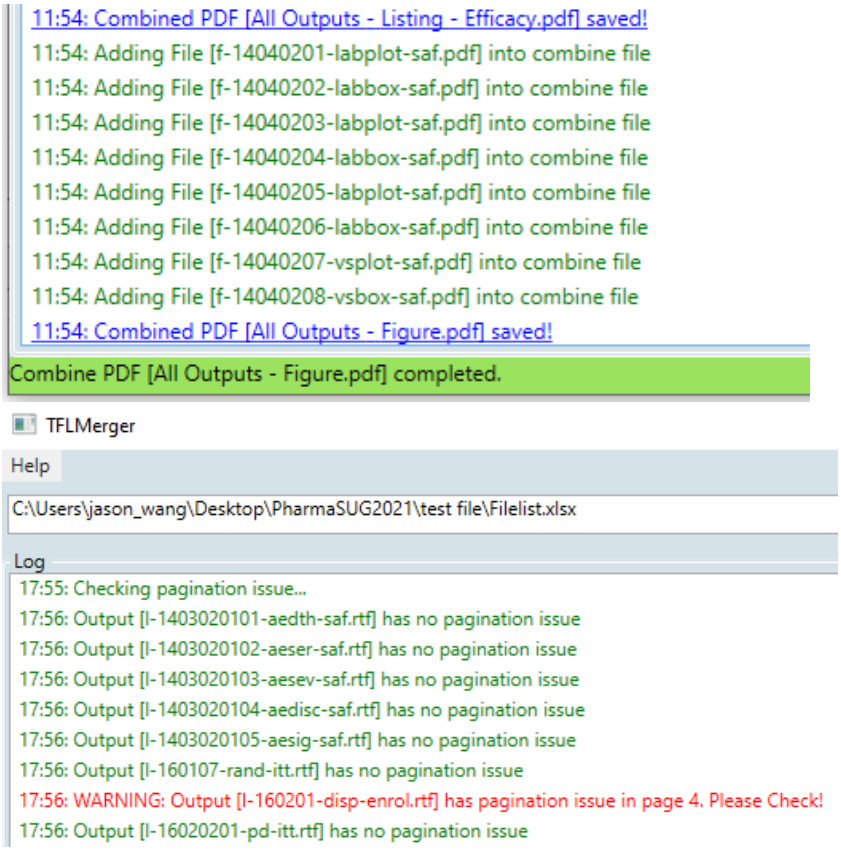


Display 6. Status bar and progress bar at the bottom

OTHER FEATURES

There are some other useful features for TFLMerger.

1. The Filelist template is integrated to the TFLMerger, you can download it from the “Help” menu without internet.
2. For “Check Pagination”, “Convert” and “Combine”, you can select anyone or any two or all three actions. If multiple actions are selected, the TFLMerger executes them one by one sequentially. However, if any issue comes out, the TFLMerger stops the process and print the issue in the log window. For example, if the “Check Pagination” action identifies any pagination issue for RTF outputs, the next action will not be executed since it is not worth to combine them if any individual output has pagination issue. Please set the value of “Generate Filelist” to “N” when you use the 3 core functions.
3. The combination order for each file is exactly the same as they are in file list. You can sort the file list in Excel per your needs if you don’t want the default order.
4. In file list tab, the “Select” column is used to select items. If all values are empty for this column, then select all items. What if we want to exclude the whole sheet? Back to the “Config” tab “Options” column, just add a “!” after “Sheetn”, for example, change “Sheet3” to “Sheet3!” in this way, the sheet3 will be exclude from the process.
5. The text in log window has 3 pre-defined color (Display 7), green for normal text, red for issues, and blue for text with hyperlink. If red text come out, you need to pay attention to it and try to figure it out. The blue text embeds the hyperlink that take you to the folder in which the destination file exist. For example, the combined file is created and saved in a folder, clicking on the blue text opens the related folder.



Display 7. Log with color

CONCLUSION

1. TFLMerger provides 3 core functions that is common used in our daily work, they are “Check pagination issue for RTF file”, “Convert individual RTF file to PDF file” and “Combine source files to a single one”.
2. TFLMerger also provides an auxiliary function to generate an Excel file with a configuration tab and other file list tabs. This file drives the TFLMerger to execute the functions with detailed behavior configured in it.
3. No programming skill is needed, what you need to do is edit the “Config” tab in Filelist and click the button in this tool. You don’t have to be a SAS programmer, any other roles who have this need is capable to use it.
4. The user interface is concise. The status bar and progress bar help you to monitor the progress of each tasks. The log window makes the process clear for users, if any problem comes out, you can catch them easily.
5. This tool is only for Windows OS, Microsoft Word and Excel should be installed on your OS.
6. The ItextSharp is an open source project that comply with AGPL, however it is not free for commercial use. The key source code can be found in appendix.

REFERENCES

ItextSharp Reference

https://afterlogic.com/mailbee-net/docs-itextsharp/

CONTACT INFORMATION

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

Name: Jason Wang
Enterprise: WuXi Clinical
E-mail: jason_wang@wuxiapptec.com

Name: Yashan Zhou
Enterprise: WuXi Clinical
E-mail: zhou_yashan@wuxiapptec.com

APPENDIX: KEY SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Windows;
using Windoc = System.Windows.Documents;
using System.Windows.Controls;
using Excel = Microsoft.Office.Interop.Excel;
using Word = Microsoft.Office.Interop.Word;
using System.IO;
using System.Data;
using System.Windows.Media;
using iTextSharp.text;
using iTextSharp.text.pdf;
using iTextSharp.text.pdf.draw;
using System.Diagnostics;
using System.Text.RegularExpressions;

namespace TFLMerger
{
  class Merger
  {
    internal string filelistLoc;
    internal Dictionary<string, string> config = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase);
    internal List<(string, string, string, DataTable)> FileDict = new List<(string, string, string, DataTable)>(); //(SheetID,
SheetName, FilePath, Filelist)
    internal List<string> logs = new List<string>();
    internal List<string> ActionList = new List<string>();
    internal bool loadSuccess = false;
    internal RichTextBox logBox;
    internal TextBlock statusBar;
    internal ProgressBar pBar;

    public Merger(string fileList, RichTextBox textBox, TextBlock textBlock, ProgressBar progressBar)
    {
      filelistLoc = fileList;
      logBox = textBox;
      this.logBox.Dispatcher.Invoke(() =>
      {
        logBox.Document.LineHeight = 3;
      });
      statusBar = textBlock;
      pBar = progressBar;
      LoadFilelist(filelistLoc);
    }

    internal void Run()
    {
      if (loadSuccess)
      {
        ClearLog();

        if (ActionList.Contains("A"))
        {
```

```
      PushpBar(0);
      GenerateFilelist();
    }
    if (ActionList.Contains("B"))
    {
      PushpBar(0);
      int num = CheckPagination();
      if (num > 0) return;//if found any pagination issue, stop the further actions
    }
    if (ActionList.Contains("C"))
    {
      PushpBar(0);
      Convert2PDF();
    }
    if (ActionList.Contains("D"))
    {
      PushpBar(0);
      Combine();
    }
  }
}

internal void LoadFilelist(string fileList)
{

  DataSet fileDS = Misc.Excel2DataSet(fileList);

  //reset flag;
  loadSuccess = false;

  //clear log;
  logs?.Clear();
  ClearLog();
  PutStatusBar("");
  PushpBar(0);

  //clear dictionary;
  FileDict?.Clear();
  config?.Clear();

  //initialize a config and set options to default;
  config.Add("File Type", "RTF");
  config.Add("Generate Filelist", "N");
  config.Add("Check Pagination", "N");
  config.Add("Convert", "Y");
  config.Add("Combine", "Y");
  config.Add("Combine PageNum", "N");
  config.Add("Book Mark", "Y");
  config.Add("TOC", "N");
  config.Add("Language", "E");
  config.Add("Merge Mode", "S");

  //load the config dataset and overwrite the default;
  if (!fileDS.Tables.Contains("config"))
```

```
    {
      PutLog("Config Issue: config tab is not available in the xlsx file", Brushes.Red);
      return;
    }
    else
    {
      string options;
      string optionsup;
      string value;
      string detail;

      DataTable configdt = fileDS.Tables["config"];

      foreach (DataRow item in configdt.Rows)
      {
        options = item["Options"].ToString();
        optionsup = options.ToUpper();
        value = item["Value"].ToString().ToUpper();
        detail = item["Detail"].ToString();

        if (!options.StartsWith("sheet", StringComparison.OrdinalIgnoreCase))
        {
          switch (optionsup)
          {
            case "FILE TYPE":
              if (value == "RTF" || value == "PDF" || value == "DOC" || value == "DOCX") config["File Type"] = value;
              else PutLog($"Config Issue: The value of Option [{options}] should be 'RTF' or 'PDF' or 'DOC' or 'DOCX'",
Brushes.Red);
              break;
            case "GENERATE FILELIST":
            case "CHECK PAGINATION":
            case "CONVERT":
            case "COMBINE":
            case "BOOK MARK":
            case "TOC":
              if (value == "Y" || value == "N") config[options] = value;
              else PutLog($"Config Issue: The value of Option [{options}] should be N or Y", Brushes.Red);
              break;
            case "COMBINE PAGENUM":
              if (value == "Y" || value == "N")
              {
                if (value == "Y")
                {
                  if (string.IsNullOrEmpty(detail)) PutLog($"Config Issue: The detail shoud not be missing if Option
[{options}] is set to Y", Brushes.Red);
                  else config[options] = detail;
                }
              }
              else PutLog($"Config Issue: The value of Option [{options}] should be N or Y", Brushes.Red);
              break;
            case "LANGUAGE":
              if (value == "C" || value == "E") config[options] = value;
              else PutLog($"Config Issue: The value of Option [{options}] should be C or E", Brushes.Red);
              break;
```

```
      case "MERGE MODE":
        if (value == "A" || value == "S" || value == "T") config[options] = value;
        else PutLog($"Config Issue: The value of Option [{options}] should be: A, S or T", Brushes.Red);
        break;
      default:
        PutLog($"Config Issue: Options [{options}] is not a valid option", Brushes.Red);
        break;
    }
  }
}

if (logs.Count > 0) return;

//validate for action: generate filelist;
if (config["Generate Filelist"] == "Y")
{
  ActionList.Add("A");
  foreach (DataRow item in configdt.Rows)
  {
    string sheetid = item["Options"].ToString();
    string sheetname = item["Value"].ToString();
    string path = item["Detail"].ToString();

    if (Regex.IsMatch(sheetid, @"^sheet\d+$", RegexOptions.IgnoreCase))
    {
      if (!string.IsNullOrEmpty(sheetname) && !string.IsNullOrEmpty(path))
      {
        if (!Directory.Exists(path))
        {
          PutLog($"Config Issue: Path for [{sheetname}] is invalid", Brushes.Red);
          continue;
        }
        string[] files = Directory.GetFiles(path, $"*.{config["File Type"]}", SearchOption.TopDirectoryOnly);
        if (files.Length == 0)
        {
          PutLog($"There are no {config["File Type"]} file in the folder for sheet [{sheetname}]", Brushes.Red);
          continue;
        }

        DataTable dt = new DataTable(sheetname);
        dt.Columns.Add("FileName");
        dt.Columns.Add("Title");
        dt.Columns.Add("Type");
        dt.Columns.Add("Select");
        foreach (var file in files)
        {
          DataRow newrow = dt.NewRow();
          newrow["FileName"] = Path.GetFileName(file);
          dt.Rows.Add(newrow);
        }
        FileDict.Add((sheetid, sheetname, path, dt));
      }
    }
  }
```

```
        if (FileDict.Count == 0) PutLog("Config Issue: At least one Sheet should be specified in the config tab",
Brushes.Red);
      }
      else//validate for other actions
      {
        //validate for action: check pagination
        if (config["Check Pagination"] == "Y")
        {
          ActionList.Add("B");
          if (config["File Type"] != "RTF") PutLog("Config Issue: Unable to check pagination since the File Type is not
RTF", Brushes.Red);
        }
        //validate for action: convert
        if (config["Convert"] == "Y")
        {
          ActionList.Add("C");
          if (config["File Type"] == "PDF") PutLog("Config Issue: Unable to convert to PDF since the File Type is already
PDF", Brushes.Red);
        }
        //validate for action: combine
        if (config["combine"] == "Y")
        {
          if (!ActionList.Contains("C") && config["File Type"] != "PDF") ActionList.Add("C");
          ActionList.Add("D");
        }
        //validate the specified sheets
        foreach (DataRow item in configdt.Rows)
        {
          string sheetid = item["Options"].ToString().Trim();
          string sheetname = item["Value"].ToString();
          string path = item["Detail"].ToString();

          if (Regex.IsMatch(sheetid,@"^sheet\d+$",RegexOptions.IgnoreCase))
          {
            if (!string.IsNullOrEmpty(sheetname) && !string.IsNullOrEmpty(path))
            {
              if (!fileDS.Tables.Contains(sheetname))
              {
                PutLog($"Config Issue: Can not find [{sheetname}] sheet", Brushes.Red);
                continue;
              }
              else
              {
                if (fileDS.Tables[sheetname].Rows.Count == 0)
                {
                  PutLog($"Config Issue: [{sheetname}] sheet contains 0 rows", Brushes.Red);
                  continue;
                }
              }
              if (!Directory.Exists(path))
              {
                PutLog($"Config Issue: Path for [{sheetname}] is invalid", Brushes.Red);
                continue;
              }
```

```
            DataTable dt = fileDS.Tables[sheetname];
            if (!dt.Columns.Contains("FileName"))
            {
              PutLog($"Error: 'FileName' column is missing for [{sheetname}] sheet", Brushes.Red);
              continue;
            }
            if (!dt.Columns.Contains("Title")) dt.Columns.Add("Title");
            if (!dt.Columns.Contains("Type")) dt.Columns.Add("Type");
            if (!dt.Columns.Contains("Select")) dt.Columns.Add("Select");

            bool anyselect = false;
            foreach (DataRow row in dt.Rows)
            {
              if (row["Select"].ToString().ToUpper() == "Y") anyselect = true;
            }
            if (anyselect)
            {
              foreach (DataRow row in dt.Rows)
              {
                if (row["Select"].ToString().ToUpper() != "Y") row.Delete(); ;
              }
            }
            dt.AcceptChanges();
            FileDict.Add((sheetid, sheetname, path, dt));
          }
        }
      }
      if (FileDict.Count == 0) PutLog("Error: No file to be processed", Brushes.Red);
    }
  }
  if (logs.Count == 0)
  {
    loadSuccess = true;
    if (ActionList.Contains("A")) PutLog("Action: Generate Filelist.");
    else
    {
      if (ActionList.Contains("B")) PutLog("Action: Check Pagination.");
      if (ActionList.Contains("C")) PutLog($"Action: Convert individual {config["File Type"]} to PDF.");
      if (ActionList.Contains("D")) PutLog("Action: Combine Files.");
    }
    foreach (var item in FileDict)
    {
      PutLog($"[{item.Item2}]: {item.Item4.Rows.Count} files");
    }
    PutLog("Load Successful!\n");
  }
  else PutLog("Load failed!\n");
}

internal void GenerateFilelist()
{
  List<string> logs = new List<string>();
  Excel.Application xlsapp=null;
```

```
  Excel.Workbook xlsbook=null;
  try
  {
    xlsapp = new Excel.Application();
    xlsapp.Visible = false;
    xlsapp.DisplayAlerts = false;
    xlsapp.Workbooks.Open(filelistLoc, false, true);
    xlsbook = xlsapp.Application.ActiveWorkbook;
    foreach (Excel.Worksheet sht in xlsbook.Worksheets)
    {
      //only keep config tab;
      if (sht.Name.ToUpper() != "CONFIG") sht.Delete();
    }
    foreach (var fileGroup in FileDict)
    {
      string sheetname = fileGroup.Item2;
      DataTable dt = fileGroup.Item4;
      Excel.Worksheet newsheet = xlsbook.Worksheets.Add(After: xlsbook.Worksheets[xlsbook.Worksheets.Count]);
      newsheet.Name = sheetname;
      Excel.Range range = newsheet.Range["A1", "D1"];
      range.Interior.Color = 0xC0C0C0;
      newsheet.Range["A1","B1"].ColumnWidth = 50;

      newsheet.Cells[1, 1] = "FileName";
      newsheet.Cells[1, 2] = "Title";
      newsheet.Cells[1, 3] = "Type";
      newsheet.Cells[1, 4] = "Select";

      int rowno = 1;
      foreach (DataRow row in dt.Rows)
      {
        rowno++;
        newsheet.Cells[rowno, 1] = row["FileName"];
        PutLog($"[{sheetname}]: {row["FileName"]}");
      }
    }
    string savepath = FileDict[0].Item3;
    xlsbook.SaveAs(savepath+ $"\\Filelist-{DateTime.Now.ToString("yyyyMMdd")}.xlsx");
    PutLog($"Completed: Filelist saved in [{savepath}].\n",hyperlink:savepath);
  }
  catch (Exception ex)
  {
    PutLog(ex.Message, Brushes.Red);
    PutLog($"Failed: Generate Filelist failed!\n", Brushes.Red);
  }
  finally
  {
    Misc.xlsDispose(xlsapp, xlsbook);
  }
}

internal int CheckPagination()
{
  PutLog("Checking pagination issue...");
```

```
string msg = null;
Word.Application o_wordapp = new Word.Application();
o_wordapp.Visible = false;
o_wordapp.ScreenUpdating = false;
Word.Document vdoc = null;

string vstr, path, filename;
DataTable sheet = null;

int issueCount = 0;
double count = 0;
double total = 0;
foreach (var item in FileDict)
{
  total += item.Item4.Rows.Count;
}

try
{
  foreach (var item in FileDict)
  {
    path = item.Item3;
    sheet = item.Item4;

    foreach (DataRow rw in sheet.Rows)
    {
      count++;
      filename = rw["FileName"].ToString();
      if (!string.IsNullOrEmpty(filename))
      {
        vstr = path + "\\" + filename;
        if (File.Exists(vstr) && Path.GetExtension(vstr).ToLower() == ".rtf")
        {
          PutStatusBar($"Checking pagination for [{filename}]");

          vdoc = o_wordapp.Documents.Open(vstr, false, true);    //open the rtf file in read-only mode

          //we are assuming each page is a section
          int pagenum = vdoc.Sections.Count;
          var range = vdoc.Range().GoTo(Word.WdGoToItem.wdGoToPage, Word.WdGoToDirection.wdGoToLast);
          int actpagenum = range.get_Information(Word.WdInformation.wdActiveEndPageNumber);

          Word.Range secrange;

          if (pagenum != actpagenum)
          {
            for (int i = 1; i <= pagenum; i++)
            {
              // Find the first section that contains more than 1 page.
              if (vdoc.Sections[i].Range.ComputeStatistics(Word.WdStatistic.wdStatisticPages) >= 2)
              {
                secrange = vdoc.Sections[i].Range;

                if (i < pagenum)
```

```
                {
                  secrange.MoveEnd(Word.WdUnits.wdWord, -1);
                }

                if (secrange.ComputeStatistics(Word.WdStatistic.wdStatisticPages) >= 2)
                {
                  issueCount++;
                  msg = $"WARNING: Output [{filename}] has pagination issue in page {i}. Please Check!";
                  PutLog(msg, Brushes.Red);
                  break;
                }
              }
            }
          }
          else
          {
            PutLog($"Output [{filename}] has no pagination issue");
          }
          vdoc.Close(false);
          vdoc = null;
        }
      }
      PushpBar(count / total * 100);
    }//end loop of each file
  }//end loop of each sheet
  PutLog("Checking pagination completed.\n");
  PutStatusBar("Checking pagination completed.");
 }
 catch (Exception ex)
 {
  MessageBox.Show(ex.Message);
  issueCount=999;
 }
 finally
 {
  if (vdoc != null)
  {
   vdoc.Close();
   vdoc = null;
   o_wordapp.Quit();
  }
  if (o_wordapp != null) o_wordapp.Quit();
 }

 return issueCount;
} //end of CheckPagination

internal void Convert2PDF()
{
 PutLog($"Converting {config["File Type"]} to PDF...");

 string vsrcf, vdstf;
 string fext = null;
 string rtffnm = null;
```

```
string pdffnm = null;
string path;
DataTable sheet;

Word.Application o_wordapp = new Word.Application();
o_wordapp.Visible = false;
o_wordapp.ScreenUpdating = false;
Word.Document vdoc = null;

double count = 0;
double total = 0;
foreach (var item in FileDict)
{
  total += item.Item4.Rows.Count;
}

try
{
  foreach (var item in FileDict)
  {
    path = item.Item3;
    sheet = item.Item4;

    //check if file exist
    foreach (DataRow r in sheet.Rows)
    {
      count++;
      PutStatusBar($"Converting {rtffnm}");

      rtffnm = r["FileName"].ToString();
      vsrcf = path + "\\" + rtffnm;
      if (!File.Exists(vsrcf))
      {
        PutLog($"Warning: File [{rtffnm}] does not exist.", Brushes.Orange);
        continue;
      }

      //check file extension
      FileInfo finfo = new FileInfo(vsrcf);
      fext = finfo.Extension;
      if (!string.IsNullOrEmpty(fext)) fext = fext.ToLower().Trim();
      if (fext == ".pdf")
      {
        PutLog($"Warning: File [{rtffnm}] is already a PDF file.", Brushes.Orange);
        continue;    //not a rtf file, continue on
      }

      pdffnm = Path.ChangeExtension(rtffnm, ".pdf");
      vdstf = path + "\\" + pdffnm;

      //convert
      vdoc = o_wordapp.Documents.Open(vsrcf, false, true);
      vdoc.Activate();
```

```
        vdoc.SaveAs(vdstf, Word.WdSaveFormat.wdFormatPDF);

        vdoc.Close(Word.WdSaveOptions.wdDoNotSaveChanges, null, null);
        vdoc = null;
        PutLog($"File converted to PDF: {pdffnm}");
        PushpBar(count / total * 100);
      }
    }
    PutLog("Convert to PDF completed.\n");
    PutStatusBar("Convert to PDF completed.");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
  finally
  {
    if (vdoc != null)
    {
      ((Word._Document)vdoc).Close(Word.WdSaveOptions.wdDoNotSaveChanges, null, null);
      vdoc = null;
    }
    if (o_wordapp != null)
    {
      o_wordapp.Quit();
      System.Runtime.InteropServices.Marshal.ReleaseComObject(o_wordapp);
      o_wordapp = null;
      GC.Collect();
    }
  }
}

internal void Combine()
{
  PutLog("Combining...");

  string fnm = null;
  string title = null;
  string vstr;
  string sheetname;
  string path;
  DataTable sheet;
  DataTable allSheets;
  PdfReader vrdr = null;
  PdfImportedPage vpage = null;
  FileStream voustream = null;
  Document vdoc = null;
  PdfCopy vpdf = null;

  //prepare numbers for progress bar
  double count = 0;
  double total = 0;

  string firstPath = FileDict[0].Item3;
```

```
allSheets = FileDict[0].Item4.Clone();
allSheets.Columns.Add("InPath");
allSheets.Columns.Add("OutPath");

foreach (var item in FileDict)
{
  sheetname = item.Item2;
  path = item.Item3;
  sheet = item.Item4.Copy();
  sheet.Columns.Add("InPath");
  sheet.Columns.Add("OutPath");

  total += item.Item4.Rows.Count;

  //derive Type filed per Merge Mode defined in config tab
  foreach (DataRow r in sheet.Rows)
  {
    switch (config["Merge Mode"])
    {
      case "A":
        r["Type"] = "All Outputs";
        r["InPath"] = path;
        r["OutPath"] = firstPath;
        break;
      case "S":
        r["Type"] = "All Outputs - " + sheetname;
        r["InPath"] = path;
        r["OutPath"] = path;
        break;
      case "T":
        if (string.IsNullOrEmpty(r["Type"].ToString().Trim())) r["Type"] = "All Outputs - " + sheetname;
        else r["Type"] = "All Outputs - " + sheetname + " - " + r["Type"].ToString().Trim();
        r["InPath"] = path;
        r["OutPath"] = path;
        break;
      default:
        break;
    }
    if (string.IsNullOrEmpty(r["Title"].ToString().Trim())) r["Title"] = r["FileName"];
    else r["Title"] = r["Title"].ToString().Trim();
  }
  allSheets.Merge(sheet);
}

if (allSheets.Rows.Count == 0)
{
  PutLog("Error: No file has been listed for combining. Please Check", Brushes.Red);
  return;
}

try
{
  DataTable uniqType = new DataView(allSheets).ToTable(true, "Type", "OutPath");
```

```
        //loop through each type
        foreach (DataRow typeRow in uniqType.Rows)
        {
          string type = typeRow["Type"].ToString();
          string outpath = typeRow["OutPath"].ToString();
          string outfile = outpath + "\\" + type + ".pdf";
          DataRow[] SubsetType = allSheets.Select($"Type='{type}'");

          //create TOC;
          if (config["TOC"] == "Y")
          {
            if (!string.IsNullOrEmpty(config["Language"]))
            {
              PutStatusBar($"Processing TOC for {type}");
              string language = config["Language"].Trim();
              CreateTOC(SubsetType, language, null);
            }
          }

          BaseFont bf = BaseFont.CreateFont(BaseFont.HELVETICA, BaseFont.CP1250,
BaseFont.NOT_EMBEDDED);
          voustream = new FileStream(outfile, FileMode.Create);

          //use copy method to get the whole page of individual pages;
          vdoc = new Document();
          vpdf = new PdfCopy(vdoc, voustream);

          vdoc.Open();

          //Prepare Bookmark font

FontFactory.RegisterDirectory(System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Wi
ndows), "fonts"), true);
          var xxfont = iTextSharp.text.FontFactory.RegisteredFonts;
          iTextSharp.text.Font bkfont = iTextSharp.text.FontFactory.GetFont(iTextSharp.text.FontFactory.HELVETICA,
11, iTextSharp.text.Font.BOLD, iTextSharp.text.BaseColor.RED);

          if (config["TOC"] == "Y")
          {
            fnm = outpath + "\\toc - " + type + ".pdf";

            if (File.Exists(fnm))
            {
              using (vrdr = new PdfReader(fnm))
              {
                for (int i = 0; i < vrdr.NumberOfPages; i++)
                {
                  vpage = vpdf.GetImportedPage(vrdr, i + 1);
                  vpdf.AddPage(vpage);
                }

                vpdf.FreeReader(vrdr);
              }
            }
```

```
  else
  {
    PutLog($"Error: TOC file [{fnm}] does not exist.", Brushes.Red);
  }
}

int fileno = 0;
foreach (DataRow bkm in SubsetType)
{
  fnm = System.IO.Path.ChangeExtension(bkm["FileName"].ToString(), ".pdf");
  string shortfnm = fnm;
  title = bkm["Title"].ToString();
  fnm = bkm["InPath"] + "\\" + fnm;

  count++;
  PutLog($"Adding File [{shortfnm}] into combined file");
  PutStatusBar($"Adding File [{shortfnm}] into combined file");
  PushpBar(count / total * 100);

  if (File.Exists(fnm))
  {
    fileno++;
    using (vrdr = new PdfReader(fnm))
    {
      iTextSharp.text.Paragraph para = new iTextSharp.text.Paragraph(title, bkfont);
      //Then create a chapter from the above paragraph
      iTextSharp.text.Chapter chpter = new iTextSharp.text.Chapter(para, fileno);
      chpter.BookmarkTitle = title;
      chpter.NumberDepth = 0;
      vdoc.Add(chpter);

      for (int i = 0; i < vrdr.NumberOfPages; i++)
      {
        vpage = vpdf.GetImportedPage(vrdr, i + 1);
        vpdf.AddPage(vpage);
      }

      vpdf.FreeReader(vrdr);
    }
  }
  else
  {
    PutLog($"Error: File [" + fnm + "] does not exist.", Brushes.Red);
  }
} // looping through all input files

//vdoc.AddAuthor(o_author);
vdoc.AddCreationDate();

vstr = Environment.UserName;
if (!string.IsNullOrEmpty(vstr)) vdoc.AddCreator(vstr);

//close object
if (vdoc != null)
```

```
        vdoc.Close();
      if (vpdf != null)
        vpdf.Close();
      if (voustream != null)
        voustream.Close();
      PutLog($"Combined PDF [{type}.pdf] saved!",hyperlink:outpath);
      PutStatusBar($"Combine PDF [{type}.pdf] completed.");
    }
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
  finally
  {
    if (vdoc != null)
      vdoc.Close();
    if (vpdf != null)
      vpdf.Close();
    if (voustream != null)
      voustream.Close();
  }

}

private void CreateTOC(DataRow[] subSetRows, string language, string font = null)
{
  string title = null;
  string fnm = null;
  string fpath = null;
  string ftype = null;
  string tocTitle = null;
  int totpages = 0;
  Rectangle fstpage = null;
  Document doc = null;
  PdfWriter wrt = null;

  DataTable vtbl = new DataTable();
  vtbl.Columns.Add("file-name");
  vtbl.Columns.Add("title");
  vtbl.Columns.Add("dest-name");
  vtbl.Columns.Add("number of pages");
  vtbl.Columns.Add("starting-page");

  try
  {
    foreach (DataRow r in subSetRows)
    {
      fnm = r["FileName"].ToString().Trim();
      fnm = Path.ChangeExtension(fnm, ".pdf");
      fpath = r["InPath"].ToString().Trim();
      ftype = r["Type"].ToString().Trim();

      using (PdfReader rd = new PdfReader(fpath + "\\" + fnm))
```

```
        {
          int n = rd.NumberOfPages;
          if (n == 0) continue;
          if (fstpage == null) fstpage = rd.GetPageSizeWithRotation(1);

          DataRow rw = vtbl.NewRow();
          rw["file-name"] = fnm;
          rw["title"] = r["Title"].ToString();
          rw["dest-name"] = r["Title"].ToString();       //max allowed to have 100,000 files, hope never reached
          rw["number of pages"] = n;
          rw["starting-page"] = totpages + 1;
          totpages = totpages + n;

          vtbl.Rows.Add(rw);
        }
      } //end of loop

      if (language == "E")
      {
        if (string.IsNullOrEmpty(font)) font = "Arial";
        tocTitle = "Table of Contents";
      }
      else
      {
        if (string.IsNullOrEmpty(font)) font = "Simsun";
        tocTitle = "目录";
      }

      if (!FontFactory.IsRegistered(font))
      {

FontFactory.RegisterDirectory(System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Wi
ndows), "fonts"), true);
      }
      var fntHead = FontFactory.GetFont(font, BaseFont.IDENTITY_H, BaseFont.EMBEDDED, 12, Font.BOLD);
      var fntBody = FontFactory.GetFont(font, BaseFont.IDENTITY_H, BaseFont.EMBEDDED, 10);

      //Font fntHead = FontFactory.GetFont("Arial", 10, Font.NORMAL, clrBlack);
      FileStream fs = new FileStream(fpath + "\\toc - " + ftype + ".pdf", FileMode.Create, FileAccess.Write,
FileShare.None);
      doc = new Document();
      doc.SetPageSize(fstpage);
      wrt = PdfWriter.GetInstance(doc, fs);
      doc.Open();

      Chunk dottedLine = new Chunk(new DottedLineSeparator());

      //output TOC
      Paragraph p;
      p = new Paragraph(tocTitle, fntHead);
      p.Alignment = 1;
      doc.Add(p);
      p = new Paragraph(new Chunk(new iTextSharp.text.pdf.draw.LineSeparator(0.0F, 100.0F, BaseColor.BLACK,
Element.ALIGN_CENTER, 1)));
```

```
    doc.Add(p);

    foreach (DataRow r in vtbl.Rows)
    {
      title = r["title"].ToString();
      int stpg = int.Parse(r["starting-page"].ToString());

      Chunk chunk = new Chunk(title, fntBody);
      p = new Paragraph(chunk);
      p.Add(dottedLine);

      chunk = new Chunk(stpg.ToString());
      p.Add(chunk);

      doc.Add(p);
    }
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
  finally
  {
    if (doc != null)
      doc.Close();
    if (wrt != null)
    {
      wrt.Close();
      wrt.Dispose();
    }
  }
}

internal void PutLog(string log, SolidColorBrush color = null, string hyperlink = null)
{
  logs.Add(log);

  if (color == null)
  {
    if (string.IsNullOrEmpty(hyperlink)) color = Brushes.Green;
    else color = Brushes.Blue;
  }
  string time = DateTime.Now.ToShortTimeString();
  log = time + ": " + log;

  this.logBox.Dispatcher.Invoke(() =>
  {
    Windoc.Paragraph para = new Windoc.Paragraph();
    if (string.IsNullOrEmpty(hyperlink))
    {
      Windoc.Run runtext = new Windoc.Run(log);
      runtext.Foreground = color;
      para.Inlines.Add(runtext);
    }
```

```
        else
        {
          Windoc.Hyperlink link = new Windoc.Hyperlink();
          link.Foreground = color;
          link.Inlines.Add(log);
          link.MouseLeftButtonDown += ((s, arg) =>
          {
            Process proc = new Process();
            proc.StartInfo.FileName = hyperlink;
            proc.Start();
          });
          link.MouseEnter += ((s, arg) =>
          {
            logBox.Cursor = System.Windows.Input.Cursors.Hand;
          });
          link.MouseLeave += ((s, arg) =>
          {
            logBox.Cursor = System.Windows.Input.Cursors.Arrow;
          });
          para.Inlines.Add(link);
        }

        logBox.Document.Blocks.Add(para);
        logBox.ScrollToEnd();
      });
    }

    internal void ClearLog()
    {
      logs.Clear();
      this.logBox.Dispatcher.Invoke(() =>
      {
        logBox.Document.Blocks.Clear();
      });
    }

    internal void PutStatusBar(string text)
    {
      this.statusBar.Dispatcher.Invoke(() =>
      {
        statusBar.Text = text;
      });
    }

    internal void PushpBar(double percent)
    {
      this.pBar.Dispatcher.Invoke(() =>
      {
        pBar.Value = percent;
      });
    }
  }
}
```