Paper CT05

A GROOVY way to enhance your SAS life

Karl Kennedy, GCE Solutions, Manchester, UK

ABSTRACT

PROC GROOVY is a relatively new (SAS® 9.3+) and little-known SAS procedure that can unleash the power of Apache's GROOVY language (Java™-syntax compatible object-oriented programming language) within your SAS environment. To display the power of PROC GROOVY two methods for producing a bookmarked PDF containing trial results (tables, figures, and listings) are described. The first method uses PROC GROOVY/Java while the second uses standard BASE SAS. The primary goal of this paper is to show that with minimal Java and PROC GROOVY knowledge the programming burden/complexity of the task can be greatly reduced, for example by removing the requirement to parse an RTF file. This is achieved by utilising Java's vast collection of Application Programming Interfaces (APIs). A secondary goal is to open the world of Java to SAS programmers.

INTRODUCTION

In Clinical Trial reporting it is often required to provide the analysis results (tables, figures, and listings (TFLs)) in one complete PDF document. In a previous role, a client made such a request at the end of the study and without any process in place to handle this request we had to investigate how to combine all the RTFs into one PDF document. Fortunately, an excellent paper “A SAS Macro Utility to Append SAS-Generated RTF Outputs and Create the Table of Contents” [1] describes how to append RTF tables and listings into one RTF document which could then be saved as a PDF file. Now with the benefit of hindsight I think using PROC GROOVY is a viable alternative to using just SAS to parse and edit RTF files in order to create a combined TFL PDF document.

Below is an excerpt from the above-mentioned paper which describes the overall process using Base SAS.

1. Prerequisites – SAS Generated RTF outputs should hold the title information in the document property section which can be added using TITLE option in ODS RTF Statement. Unique bookmarks should be created in the RTF outputs for hyperlinking from TOC page, using ANCHOR option in ODS RTF statements as explained later in process 1.

2. Title and page number information will be extracted from the individual RTF outputs by reading them using SAS INFILE statement. The macro will loop through all the RTF files in a given directory to accomplish this. The information collected will be used in creation of the TOC page along with the hyperlinks to respective outputs. Process 2 details this step.

3. Once the TOC page is created, the next step is to append the individual RTF files in order of the table or listing numbers, following the TOC page. Process 3 will show how the consolidated document is generated.

I based my program on the concepts described in the above-mentioned paper but unfortunately for me my RTF structure was different to the version in the paper, so I had to spend time understanding the structure of RTF files. Also, to consolidate each RTF file the last ‘}’ on each file must be removed and a “\sect” tag must be inserted between each RTF file. The understanding and manipulation of RTF can be avoided by using PROC GROOVY/Java because it can read individual PDF files directly so you can avoid the RTF manipulation step entirely by using SAS ODS to create the TFLs in PDF format and then using Java to create a combined PDF file using the individual TFL files as input files. Also using PROC GROOVY the inclusion of graphs is also possible. Unfortunately, when I used the RTF manipulation approach, I had to manually include the graphs in the combined document.

The rest of the paper describes how to create a combined TFL document using PROC GROOVY and Apache PDFBox® which is a Java API for working with PDF documents.

GETTING STARTED WITH GROOVY

Groovy is a language that runs in a Java Virtual Machine so this must be set up before you can use PROC GROOVY. This set up is beyond the scope of this paper but two sources [2] [3] about how to do this and how to test the set up can be found in References.

To be able to read/create/edit PDFs using PROC GROOVY I used Apache PDFBox® which can be downloaded from the PDFBox website [4]. Once this API has been downloaded it can be referenced within the PROC GROOVY procedure by using 'add' command. The code below shows the basic structure of the PROC GROOVY procedure and how to reference additional Java/Groovy APIs.

```
proc groovy;
   add classpath = "C:\groovy\pdfbox-app-2.0.15.jar";
   submit;
       // Place Groovy code within submit statements.

   endsubmit;
quit;
```

To use PROC GROOVY it is beneficial to understand the fundamentals of Java object-oriented programming. Many years ago I used Java while at university so I have retained some knowledge of the language and some key concepts e.g. classes, objects, scope of variables, and loops and with that basic knowledge and access to vast Java resources online I was able to produce a basic Java program that could achieve my aim of producing a PDF document that includes all tables, figures, and listings (TFLs) for a study. A good starting point for obtaining basic Java knowledge is the official Java tutorial [5].

PROGRAM LAYOUT

To simplify the process of generating one PDF document that contains all TFLs, output the individual TFLs in PDF format using ODS in SAS. These PDF files are used as the input files for the PROC GROOVY program.

The three main aspects of the program are:

1. Appending TFLs into one PDF document.
2. Creating a bookmark for each TFL.
3. Create a Table of Contents (TOC)

APPENDING TFLS INTO ONE PDF DOCUMENT

First step is to create an empty PDF document which will contain all the appended TFLs. This is created by using the PDDocument class from PDFBox. I created this in the Java main method within the PROC GROOVY procedure.

```
doc = new PDDocument();
```

Once your PDF document has been initialized you must read in each TFL and append them to the newly created PDF document. To achieve this, I created a simple method that looped through a comma delimited text file that listed each TFL id and title and with that information I created a Java File object for each TFL which is then used to load each TFL into a new PDDocument.

```
File file = new File("C:\\List_of_Outputs.txt");
BufferedReader br = new BufferedReader(new FileReader(file));

String row;
int tflCount = 1;
row = br.readLine(); // Reads header line;
while ((row = br.readLine() != null )
{   String[] rowcells = row.split(",",2); // TFL file name;
    File pdf = new File(tflloc + rowcells[0] + ".pdf");
```

```
        // Call to method for appending TFLs to PDF document.
        // Reference section CREATE BOOKMARKS FOR EACH TFL for this code.

        tflCount++;
    }
```

I used a text file for reading the list of TFLs for simplicity but to improve this code you could read the files directly from a directory or if you used an Excel spreadsheet to track the status of the TFLs during development then this could also be used to get the TFL file names. To read Excel files using Java you can use Apache POI – the Java API for Microsoft documents [6].

The process for appending the TFLs is quite straightforward, each TFL is initially stored in a temporary PDDocument object. Then each individual page within that TFL output is appended to the root/main PDDocument by simply iterating through the TFL document and appending each page to the final combined PDF document.

```
File tflFile = new File(file.getCanonicalPath());
PDDocument tempDoc = PDDocument.load(tflFile);

// Stores the first page of TFL which is used as a bookmark destination.
PDPage page = (PDPage) tempDoc.getDocumentCatalog.().getPages().get(0);

int numOfPages = tempDoc.getNumberOfPages();
for(int i=0; i < numOfPages; i++)
{   PDPage tflPage = (PDPage) tempDoc.getDocumentCatalog().getPages().get(i);
    doc.addPage(tflPage);

    // Code for creating bookmarks & TOC links is discussed in the next sections.

}
```

CREATE BOOKMARKS FOR EACH TFL

To create bookmarks for each TFL within the document the class PDDocumentOutline which represents the outline of a PDF document is used. This PDDocumentOutline object is then attached to final PDF document. To add the individual bookmarks to the PDDocumentOutline object the class PDOutlineItem is used with its destination set as the first page of the TFL.

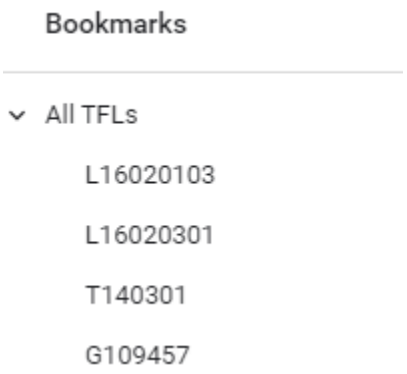
```
// Initialise objects required to create bookmarks.
PDDocumentOutline outline = new PDDocumentOutline();
doc.getDocumentCatalog().setDocumentOutline(outline);
pagesOutline = new PDOutlineItem();
pagesOutline.setTitle("All TFLs");
outline.addLast(pagesOutLine);

// Code for creating bookmarks & TOC links
// Set the bookmark title to point to first page of TFL
if (i == 0)
{   PDPageDestination dest = new PDPageFitWidthDestination();
    dest.setPage(page);
    PDOutlineItem bookmark = new PDOutlineItem();
    bookmark.setDestination(dest);
    String title = file.getName().substring(0,file.getName().indexOf("."));
    bookmark.setTitle(title);
    pagesOutline.addLast(bookmark);
```

```
        // Code to create TOC links see section CREATE A TABLE OF CONTENTS

    }
```

The above code will create PDF bookmarks that will appear like the example shown below.



CREATE A TABLE OF CONTENTS (TOC)

An excellent blog [7] on how to use hyperlinks was a great help in developing this aspect of the program along with sample code from AddAnnotations.java [8]. Even though AddAnnotations.java wouldn't run on my SAS & Java configuration.

In order to create a TOC an empty page must be created and added to the final PDF document. The code below has only been tested on a single page TOC.

```
toc = new PDPage();
doc.addPage(toc);
tocMain = new PDPageXYZDestination();
tocMain.setPage(toc);
tocMain.setLeft(0);
tocMain.setTop(0);
```

Then I created a method that uses PDFs annotation links that enables the user to navigate to specific TFLs by clicking on the TFL title in the TOC. This is done by using the PDRectangle class to place invisible rectangles over each TFL title in the TOC which then gives the impression that each TFL title is an internal hyperlink. These rectangles are then attached to the PDAnnotationLink object.

```
// Sets up a specific PDF 'go-to' action within TOC.
action = new PDActionGoTo();
action.setDestination(tocMain);
pdest = new PDPageFitWidthDestination();
pdest.setPage(thePage);

// Create link for an individual TFL title in TOC.
link = new PDAnnotationLink();
link.setAction(action);

// Use x,y coordinates to place the PDRectangle over the TFL title.
rect = new PDRectangle();
rect.setLowerLeftX(llx);
rect.setLowerLeftY(lly);
rect.setUpperRightX(urx);
```

```
rect.setUpperRightY(ury);
link.setRectangle(rect);

// Add link to TOC.
toc.getAnnotations().add(link);

// Write the TFL title to the TOC. The x,y coordinates should be inside the
//rectangle coordinates.
PDPageContentStream = stream = new PDPageContentStream(doc, toc, true, true);
stream.beginText();
stream.setFont(font, TOC_ROW_FONT_SIZE);
stream.setTextTranslation(X_TOC_POS, Y_TOC_POS + theIncrem);
stream.drawString(theRowText);
stream.endText();
stream.close();
```

CONCLUSION

As a SAS programmer PROC GROOVY isn't a procedure that I would need for my day to day tasks but it is a procedure that I will keep in mind for any non-standard tasks. The Java code in this paper is quite standard and basic and shouldn't be an issue for any SAS programmer with limited Java experience. With a small bit of research, I discovered two useful Java APIs for PDF and Excel documents which could be used by a SAS programmer. The Excel API could be a genuine alternative for handling Excel files if you don't have access to SAS/ACCESS to PC Files.

REFERENCES

[1] Anbazhagan, Sudhakar & Patel, Shridhar - A SAS Macro Utility to Append SAS-Generated RTF Outputs and Create the Table of Contents [PharmaSUG 2012 – Paper AD12].

[2] Hamilton, Jack – Writing a Useful Groovy Program When All You Know about Groovy is How to Spell It [SAS Global Forum 2013 – Paper 493-2013].

[3] SAS Community Blog – Hello Groovy (http://www.sascommunity.org/planet/blog/category/groovy/).

[4] Apache PDFBox® – A Java PDF Library (https://pdfbox.apache.org/)

[5] The Java™ Tutorials (https://docs.oracle.com/javase/tutorial/)

[6] Apache POI – the Java API for Microsoft documents (https://poi.apache.org/)

[7] Stuckert, Ralf - Hyperlinks with PDFBox-Layout (http://hardmockcafe.blogspot.com/2016/08/hyperlinks-with-pdfbox-layout_46.html)

[8] Apache Software Foundation – AddAnnotations.java (https://svn.apache.org/repos/asf/pdfbox/trunk/examples/src/main/java/org/apache/pdfbox/examples/pdmodel/AddAnnotations.java)

ACKNOWLEDGMENTS

I would like to thank Diane Peers and Robert Horton from GCE Solutions for their help and guidance throughout the development of this paper and Frank Senk (GCE Solutions) for providing the test TFLs. The GCE Solutions IT team for configuring a dedicated machine with SAS and Java.

Also, Geoff Charlwood my previous manager who provided support and guidance during the development of the SAS/RTF programming.

CONTACT INFORMATION

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

Karl Kennedy
GCE Solutions Ltd
Suite 18A, City Tower, Piccadilly Plaza
Manchester M1 4BT
karl.kennedy@gcesolutions.com