Paper SM04

Let's get Groovy - Word and PDF processing in SAS

Katja Glaß, Bayer AG, Berlin, Germany

ABSTRACT

Sometimes functionality from SAS© is not sufficient to work with Word or PDF documents, whereas there are many solutions for problems we face available in Java™. There are various ways how other programming languages can be utilized in SAS. This presentation will show how PROC GROOVY can be used to apply available solutions for specific problems we face when working with PDF and word documents.

INTRODUCTION

SAS is a powerful programming language to work with data, perform statistical evaluations and to create outputs of various kinds which we use heavily for clinical study evaluations. Sometimes tasks need to be performed where other programming languages are easier to use. For this reason SAS opened up their platform to support various other languages – one of which is Java.

Java is very powerful to create applications and has a lot of libraries which can be used to perform various tasks. When PDF or Word documents should be further processed, it is likely much easier to do so in Java than in pure SAS. There are various ways to use a Java program from within SAS. The command line can be used, the Java Data Object can be used or the PROC GROOVY procedure can be used. Groovy is a programming language which is not Java, but it supports all Java syntax, so any Java program can run in a Groovy environment and for this we can use the PROC GROOVY for our purpose.

THREE ROUTES OF JAVA IN SAS

A brief overview of the different ways Java can be called from within SAS show different options each with its own advantages and disadvantages.

The **command line** call is straightforward. Any operating system commands can be executed via the "X" command assuming they are enabled in that environment. This type of call is totally independent of SAS as everything runs on the console outside the SAS environment. The command line call might need to include the following:

Command Line

```
X 'java helloWorld';
```

- It might be required to provide the full path to the java binary in case it is not already available as command.
- The classpath option[1] (-cp) allows the use of compiled classes, for example downloaded or created .jar files. Multiple files can be added by using a colon as separator.
- The class name itself (containing the called "main"-function) can be specified which is important when multiple main classes are available.

Finally any number of arguments can be provided. This might be useful for defining parameters, input files or output files.

A common way is to develop complex Java programs in any Java editor. Then this can be executed via a simple command line call. Used libraries can either be packed into one final executable or linked via the class path. This is the recommended way when there is little information flow from SAS to Java and the programming already has a specific complexity.

The **JavaObj**[2, 3] is an option to instantiate and use a Java class and its functions within a data step. Heavy information exchange between Java and SAS is possible this way. Values can be provided to the object, calculations can be done and values received back. This method is extremely valuable when data content has to be processed by Java. Here again Java is developed in any editor, compiled and then provided to SAS via a Java class path.

JavaObj

```
DATA _NULL_;
 DCL JAVAOBJ j("helloWorldClass");
 rc = j.callStringMethod("getHelloWorld");
 PUT rc=;
RUN;
```

One major disadvantage of the JavaObj is that only specific return functions can be called. Often classes return other Java classes, lists or collections. Such functions cannot be called. This is mainly an issue when using existing java solutions which can only be used "as is".

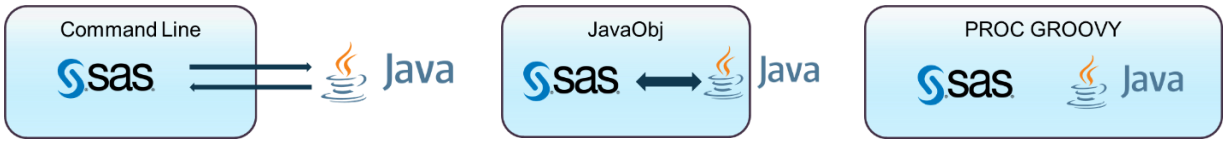
The SAS procedure **PROC GROOVY**[4, 5] allows the writing and storing of Java code in ".sas" files. SAS is not a natural Java editor, so developing in a Java environment and copying the code over might be the better solution. The main advantage is that the source code is not hidden in any other files and can easily be updated. There is then also no need for prior packing and compilation. Another advantage is that the Groovy language can do much more than just Java.

PROC GROOVY

```
PROC GROOVY;
 SUBMIT;
  System.out.println("Hello world!");
 ENDSUBMIT;
QUIT;
```

PROC GROOVY can not only be used to execute Java, but can also create JAVA classes which can then for example be used in the JavaObj.

When looking into these three options with respect to SAS integration the best integrated solution is PROC GROOVY. The command line executes Java outside of the core SAS environment. The JavaObj has a better integration but uses still Java classes which are not directly visible and it is limited to functions returning specific types. PROC GROOVY is the most generic option but it has no direct integration into the data step.



Finally, any of these three ways to use Java can be used to meet various use cases. This paper will concentrate on PROC GROOVY as no separate class creation and compilation is required.

LET'S GET GROOVY!

THE SYNTAX

The syntax is straightforward and is embedded in PROC GROOVY and QUIT. The groovy source code itself is embedded in SUBMIT and ENDSUBMIT. Very important when using existing libraries is the additional "ADD CLASSPATH statement. Here one or multiple existing libraries can be added so that their classes and functions are available.

```
PROC GROOVY;
    ADD CLASSPATH = "<path>/<file>.jar";
    SUBMIT;
        <Java source>
    ENDSUBMIT;
QUIT;
```

As the PROC GROOVY content is a Java environment, it does not know anything about SAS. Therefore, existing datasets or macro variables are not accessible per default. The following code will return `"Hello &name!"` which is not what is intended.

```
%LET name = Katja;
PROC GROOVY;
    SUBMIT;
        System.out.println("Hello &name!");
    ENDSUBMIT;
QUIT;
```

To allow the Java code to be aware of SAS items, arguments can be provided to the Java function – this is similar to arguments on the command line. The following example will return the expected "Hello Katja!".

```
%LET name = Katja;
PROC GROOVY;
    SUBMIT "&name";
        String name = args[0];
        System.out.println("Hello " + name + "!");
    ENDSUBMIT;
QUIT;
```

To standardize and generalize, typically SAS macros are used to share code for generic usage. Because of the way macros work having "SUBMIT" in macro code is not supported. But of course there is a way to create a macro containing PROC GROOVY, the workaround is to store the code in a file and then include the file.

```
%MACRO sayHello(name);
    FILENAME cmdFile TEMP;
    DATA _NULL_;
        FILE cmdFile LRECL=500;
        PUT 'PROC GROOVY;                                    ';
        PUT '    SUBMIT "&name";                             ';
        PUT '        String name = args[0];                  ';
        PUT '        System.out.println("Hello " + name + "!");';
        PUT '    ENDSUBMIT;                                  ';
        PUT 'QUIT;                                           ';
    RUN;
    %INCLUDE cmdFile;
%MEND sayHello;
%sayHello(Katja);
```

WORKING WITH PDF

After being able to run Java in SAS the next step is to look for solutions to specific problems. Sometimes it is useful to be able to read in and process PDF, for example when a report is received in PDF format. Another use case would be when final deliverables are available in PDF format, but then after a required re-run it should be checked that no (or only specific) parts are changed – in this case a PDF compare would be required.

A commonly used open source tool which is available is the PDFBox[6] from Apache. Various things can be done with PDF: extract text, split and merge files, fill forms, validate, print, save as image, create PDFs and also signing is supported.

First we download the jar-release and store it somewhere in our SAS environment.

When searching on how to read in a PDF file with Java immediately various results come up showing the required code. The following code example is used[7].

```
import java.io.File;
import java.io.IOException;

import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.text.PDFTextStripper;

public class ReadingText {
   public static void main(String args[]) throws IOException {
      //Loading an existing document
      File file = new File("C:/PdfBox_Examples/new.pdf");
      PDDocument document = PDDocument.load(file);

      //Instantiate PDFTextStripper class
      PDFTextStripper pdfStripper = new PDFTextStripper();
```

```
        //Retrieving text from PDF document
        String text = pdfStripper.getText(document);
        System.out.println(text);

        //Closing the document
        document.close();
    }
}
```

Java uses imports to make various classes or functions available. This is similar to including a SAS macro which can be used after it is made available. In the next step the class "ReadingText" is defined which should contain the corresponding source code. The "main" function is what is typically executed when calling the class. In PROC GROOVY the code which is executed within the SUBMIT block is the equivalent to the main class from Java. It can easily be guessed what the code is doing as the class, variable and function names are very descriptive. The command "`System.out.println(text)`" is used to print the text into the output. In the case of SAS this will go into the log.

The next step is to transfer the code to SAS using PROC GROOVY. Some macro variables are defined outside to have the opportunity to change this easily. All Java basic libraries are already available, however PDFBox needs to be added to the class path. By using one parameter (infile), we read this into a Java string variable. Then the imports need to be defined. We do not need the Java standard imports for io, only the ones from PDFBox. Then the remaining commands can simply be copied (and the file name changed to the macro variable).

```
%LET infile = <path>/sdtmadampilotprojectreport.pdf;
%LET pdfboxPath = <path>;
PROC GROOVY;
    ADD CLASSPATH = "&pdfboxPath./pdfbox-app-2.0.26.jar";
    SUBMIT "&infile";
        String infile = args[0];
        import org.apache.pdfbox.pdmodel.PDDocument;
        import org.apache.pdfbox.text.PDFTextStripper;
        // load document
        File file = new File(infile);
        PDDocument document = PDDocument.load(file);
        PDFTextStripper pdfStripper = new PDFTextStripper();
        // get text from PDF
        String text = pdfStripper.getText(document);
        System.out.println(text);
        // close document
        document.close();
    ENDSUBMIT;
QUIT;
```

Executing this prints all text into the log output. All of this can be done within SAS without the need to set up and use a separate Java environment. This is very straightforward. Finally the text should be further processed in SAS. To have it in the log is not sufficient. The information flow from PROC GROOVY to SAS is not particularly good. For this use case it might be best to store the text into a text file which we can then read into SAS afterwards. Searching the internet provides quick and easy solutions for this task[8]. The source code can be used to replace the "`System.out.println(text);`". As the file location we can use the work library path which we can provide as a parameter to GROOVY.

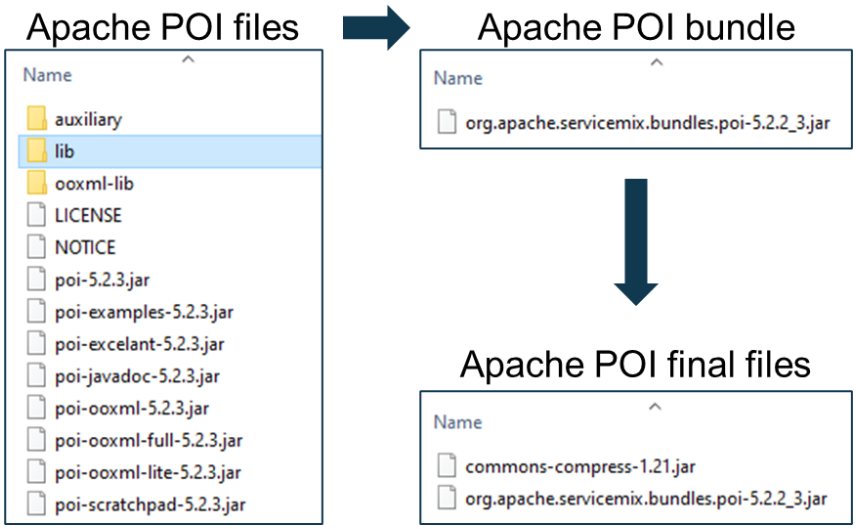
```
%LET outtxt = %SYSFUNC(PATHNAME(work))/pdfcontent.txt;
...
PROC GROOVY;
    ADD CLASSPATH = "&pdfboxPath./pdfbox-app-2.0.26.jar";
    SUBMIT "&infile" "&outtxt";
        String infile = args[0];
        String outtxt = args[1];
        ...
```

```
// Write text to file
FileWriter myWriter = new FileWriter(outtxt);
myWriter.write(text);
myWriter.close();

...
```

This way of processing enables very generic and easy usage of Java tools within SAS. It is also possible to read just specific pages, read each page one by one and store information in a more tabular format, for example using a comma separated file to help move the investigated information to SAS.

WORKING WITH DOCX

Sometimes documents we want to read and process into SAS are not in PDF but in DOCX format. This enables also a much better way to read in tabular content. A nice open source package for Java and Word formats is the Apache POI[9] library. The downloaded Apache POI contains many jar files as well as dependent jar files which are available as single files. All these must be added to the class path which is a bit cumbersome. Luckily there are also bundles available[11], so only the bundle needs to be included. When running our use case one of the commons-compress classes is missing, so that library needs to be added additionally to the bundle.



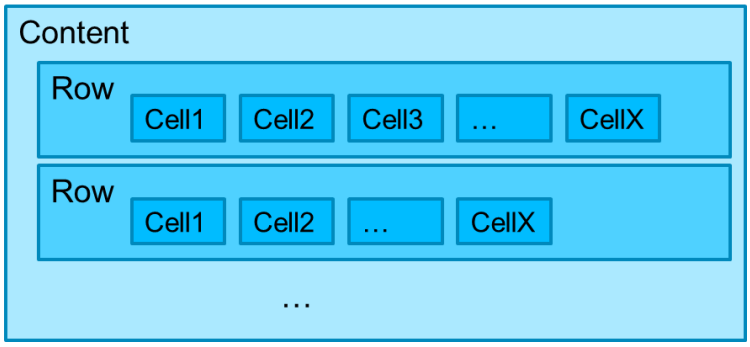
Searching the internet provides various examples on how to read in tables. As our use case is a bit more complex than just reading in the table, the complete solution is not available and some steps have to be programmed additionally. The final source might use as its basis an example which is found on the internet[10]. Then the fine-tuning and update process starts. With some experience and searching through the general Apache POI documentation optimizations can be done. It is possible to “getTables()” directly from the complete document. Exception handling, which is in general important, could also be ignored for a proof of concept. The simplified code for PROC GROOVY looks like this:

```
%LET infile = <path>/14-2.01.docx;
%LET jarPath = <path>/docum;
PROC GROOVY;
    ADD CLASSPATH = "&jarPath./commons-compress-1.21.jar";
    ADD CLASSPATH = "&jarPath./org.apache.servicemix.bundles.poi-5.2.2_3.jar";
    SUBMIT "&infile";
    import org.apache.poi.xwpf.usermodel.XWPFDocument;
    import org.apache.poi.xwpf.usermodel.XWPFTable;
    import java.nio.file.Files;
    import java.nio.file.Paths;

    String fileName = args[0];
    XWPFDocument doc = new XWPFDocument(Files.newInputStream(Paths.get(fileName)))
    List<XWPFTable> xwpfTableList = doc.getTables();
```

```
    for (XWPFTable xwpfTable : xwpfTableList) {
        System.out.println("Total Rows : " + xwpfTable.getNumberOfRows());
        for (int i = 0; i < xwpfTable.getRows().size(); i++) {
            for (int j = 0; j < xwpfTable.getRow(i).getTableCells().size(); j++) {
                System.out.println(xwpfTable.getRow(i).getCell(j).getText());
            }
        }
    }
    ENDSUBMIT;
QUIT;
```

This shows how we can access the cell values. The next step would be to save those values and export them as a comma separated file. This file type is ideal for importing back to SAS for further processing. A generic usage of lists here is a nice solution which can then also easily be exported as CSV.

First we store all information in a list of rows where each row is a list of strings, each string representing one cell content. In Java this can be done very easily and generically by using List<List<String>>. With this generic type it is also no problem if there are different numbers of cells.



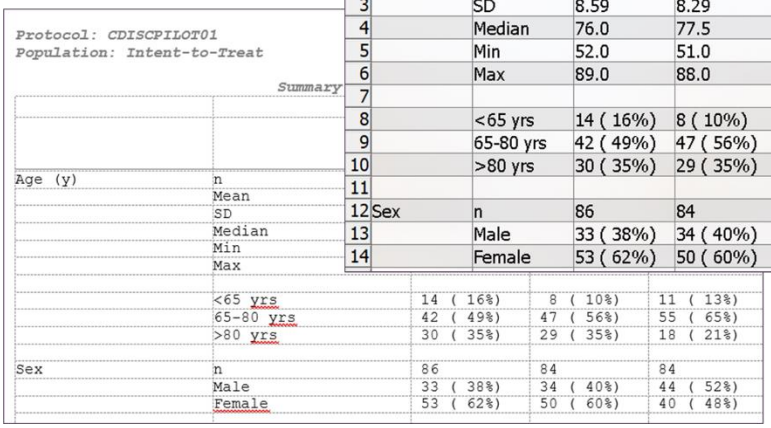
Finally this content needs to be written into a text file in a tab separated format (csv is also an option but causes more often troubles). Here we can simply include a new line for each new row and print all cells separated by a tabulator (\t) into each line. The final source transferred to PROC GROOVY is the following:

```
%LET infile = <path>/14-2.01.docx;
%LET outfile = %SYSFUNC(PATHNAME(work))/content_xx.csv;
%LET jarPath = <path>;
PROC GROOVY;
    ADD CLASSPATH = "&jarPath./commons-compress-1.21.jar";
    ADD CLASSPATH = "&jarPath./org.apache.servicemix.bundles.poi-5.2.2_3.jar";
    SUBMIT "&infile" "&outfile";

    import org.apache.poi.xwpf.usermodel.XWPFDocument;
    import org.apache.poi.xwpf.usermodel.XWPFTable;
    import java.nio.file.Files;
    import java.nio.file.Paths;

    String inFile = args[0];
    String outFile = args[1];
    XWPFDocument doc = new XWPFDocument(Files.newInputStream(Paths.get(inFile)))
    List<XWPFTable> xwpfTableList = doc.getTables();
    List<List<String>> listOfLists = new ArrayList<>();
    for (XWPFTable xwpfTable : xwpfTableList) {
        for (int i = 0; i < xwpfTable.getRows().size(); i++) {
            List<String> currentRow = new ArrayList<String>();
            for (int j = 0; j < xwpfTable.getRow(i).getTableCells().size(); j++) {
                currentRow.add(xwpfTable.getRow(i).getCell(j).getText());
            }
            listOfLists.add(currentRow);
        }
    }
```

```
    // Writing text into file
      FileWriter myWriter = new FileWriter(outFile);
      for (List<String> currentRow : listOfLists) {
          myWriter.write(String.join("\t", currentRow));
          myWriter.write('\n');
      }
      myWriter.close();
    ENDSUBMIT;
QUIT;
```

Finally the CSV file can be read into SAS for example via PROC IMPORT.

```
PROC IMPORT DATAFILE="&outfile" OUT=data DBMS=dlm REPLACE;
    getnames  = NO;
    delimiter = '09'x;
RUN;
```



	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7
1	Age (y)	n	86	84	84	254	0.5934
2		Mean	75.2	75.7	74.4	75.1	.
3		SD	8.59	8.29	7.89	8.25	.
4		Median	76.0	77.5	76.0	77.0	.
5		Min	52.0	51.0	56.0	51.0	.
6		Max	89.0	88.0	88.0	89.0	.
7							.
8		<65 yrs	14 (16%)	8 (10%)	11 (13%)	33 (13%)	0.1439
9		65-80 yrs	42 (49%)	47 (56%)	55 (65%)	144 (57%)	.
10		>80 yrs	30 (35%)	29 (35%)	18 (21%)	77 (30%)	.
11							.
12	Sex	n	86	84	84	254	0.1409
13		Male	33 (38%)	34 (40%)	44 (52%)	111 (44%)	.
14		Female	53 (62%)	50 (60%)	40 (48%)	143 (56%)	.

There are many more options available on what can be done with Word files. The complete text could be saved as a text file to ensure for example that there are no changes without looking into the table itself. The Apache POI library has a lot of functionality. The tool can be used to do any kind of reading, updating and creation of Word documents.

CONCLUSION

There are different ways to utilize other programming languages in SAS. Java is a very powerful language with many available libraries to solve all kinds of problems. It is worth checking out these options. Depending on your purpose either the command line, the data step “JavaObj” object or PROC GROOVY might be the best option.

A core advantage of PROC GROOVY is that the complete source code is visible and stored within a SAS program, no additional environment setup is needed. Of course complex program development is much easier when done in a Java development environment like Eclipse or VSCode.

This paper showed how Java code can be modified to run in SAS within the PROC GROOVY procedure. Utilizing packages like PDFBox to read in PDFs as an example or the Apache POI library to read in tables into a CSV file is not very complex and can ease up our daily work when we need to compare these formats for example.

LICENSE FOR SOURCE

The source code available in this paper uses the unlicensed license unless the original code is licensed under a different license (see links to origins).

EXAMPLE INPUTS

- For PDF the following file has been used: https://github.com/cdisc-org/sdtm-adam-pilot-project/blob/master/sdtmadampilotprojectreport.pdf
- For Word, the following file has been used (stored as docx): https://github.com/atorus-research/CDISC_pilot_replication/tree/master/outputs/14-2.01.rtf

REFERENCES

[1] Oracle Java SE Documentation – java (https://docs.oracle.com/javase/7/docs/technotes/tools/windows/java.html)

[2] Java in SAS® JavaObj, a DATA Step Component Object by Richard A. DeVenezia, Independent Consultant, SUGI 30, 2005 (https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/241-30.pdf)

[3] Using the Java Object (SAS Documentation) (https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lrcon/n0swy2q7eouj2fn11g1o28q57v4u.htm)

[4] GROOVY Procedure, SAS Documentation (https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p0pq7kp0misx44n14vvub91rzihh.htm)

[5] A GROOVY way to enhance your SAS life by Karl Kennedy, GCE Solutions, Manchester, UK, PHUSE 2019 (https://www.lexjansen.com/phuse/2019/ct/CT05.pdf)

[6] Apache PDFBox Homepage (https://pdfbox.apache.org/)

[7] PDFBox - Reading Text, accessed on 04.10.2022 (https://www.tutorialspoint.com/pdfbox/pdfbox_reading_text.htm)

[8] W3Scools - Java Create and Write To Files, accessed on 04.10.2022 (https://www.w3schools.com/java/java_files_create.asp)

[9] POI API Documentation, accessed on 04.10.2022 (https://poi.apache.org/apidocs/5.0/)

[10] Java – Read and Write Microsoft Word with Apache POI, accessed on 04.10.2022 (https://mkyong.com/java/java-read-and-write-microsoft-word-with-apache-poi/)

[11] Maven Repository - Apache ServiceMix :: Bundles :: POI , accessed on 04.10.2022 (https://mvnrepository.com/artifact/org.apache.servicemix.bundles/org.apache.servicemix.bundles.poi)

CONTACT INFORMATION

Katja Glaß
Bayer AG
Müllerstraße 178
13353 Berlin, Germany
Katja.Glass@bayer.com