Paper SM08

I Propose Aspose! Automating PDFs and Microsoft Office Files

Dominik Habel, Bayer, Berlin, Germany

Abstract

This paper looks at the commercial software Aspose which offers numerous file level APIs to work with Microsoft Office file formats as well as PDFs and many more. It discusses how you can use Aspose for Java in a SAS environment and shows several practical examples on how using Aspose can help you with your daily tasks.

Introduction

In our day-to-day business we have to work with all kinds of files on a regular basis. PDFs, RTFs, PNGs, DOCXs, XLSXs and PPTXs are just some examples of things that any of us deal with. Most of them come with their own app that you use to interact with them. For DOCX files, you open Microsoft Word to edit them. For PDFs you might use Adobe Acrobat. The difficult part begins when you want to automate certain actions. Sometimes it is easy, there are many great tools to work with images. For Excel you can sometimes just create a CSV and go from there. However, especially in a study context, there are many times where you wish you could just convert this Word file into a PDF or merge two PDFs into one. Other times, you just want to get an overview of your dozens of health authority requests or deliverables so that you know where you created which kind of Table, Figure or Listing.

These things can be tough to do in our clinical environment when all you have is SAS or R. There are ways to do it but usually they are lacking power in some form, or they are just really cumbersome to do. This is where Aspose comes into play. It offers a framework of file level APIs with which you can manipulate all kind of Microsoft document formats, PDFs and much more. Now you can automate certain simple actions in just a few lines of code, or you can go beyond and create large scale tools to automate your workflows.

Aspose

Aspose Pty Ltd is a software development company offering numerous APIs for all kinds of file formats. Their products are called Aspose.Words, Aspose.Pdf and similar depending on the file format you want to work with. There is also a Aspose.Total available that combines them all into one package.

It is not a free service. You must obtain a subscription and with that a license to be able to use it. The cost depends on the size of your company and what you want to do with it. It is not cheap but can definitely be worth it if you got the right use cases. Interestingly, if you cancel the subscription, you will not lose the license altogether. You can still use it, but you will not receive any updates. This helps for long-term planning, even if your organization decides to revoke the subscription at some point in time, your tools and macros will still work.

Aspose has started out offering their components for .NET developers and have since expanded to more platforms including Java, C++, Python (via .NET) and Android. SAS or R are not on the list which makes things a bit more challenging. If you have Python available in your Statistical Computing Environment already, you can start using Aspose that way immediately. If you want to stay closer to SAS and integrate Aspose in your existing workflows, Java can be an option. This paper will focus on the Java way of using Aspose and show different ways to use it from a SAS environment.

Unfortunately, there is no way around using another programming language in order to use Aspose. In the best-case scenario, you have someone in your team that possesses skills in one of the beforementioned languages. If not, there is an extensive documentation available for all the different parts of Aspose that give both explanations and code example to help you figure it out. Aspose also has a large forum where one can find all kinds of questions and answers to various topics. You can also use it to address your own questions. Additionally, due to the large number of forum posts and documentation, Large Language Models like ChatGPT and Claude Sonnet can help you out with programming so that some tasks can be completed in a timely manner.

Java in SAS

There are many ways of using Java inside a SAS environment. In this section, we will go over 3 different ones: PROC GROOVY, the JAVAOBJ interface and System Commands.

PROC GROOVY

This SAS procedure allows you to run Java code from inside a SAS program. You can exchange data between the SAS and Java part, and you can leverage both Java libraries as well as external third-party libraries like Aspose. It is great to use it for smaller scripts that do a single thing rather than building a larger tool. It comes with some limitations being it uses a rather old Java version, the syntax is sometimes a bit different to pure Java and for longer code it tends to be a bit overwhelming and hard to debug.

Here is a simple Hello World program call with PROC GROOVY:

```
PROC GROOVY;
SUBMIT;
    System.out.println("Hello World using Java syntax!");
ENDSUBMIT;
RUN;
```

There are PhUSE papers that go into more details, some of which are linked in the references section at the end of the paper.

JAVAOBJ Interface

JAVAOBJ is a component interface in SAS that allows DATA step programs to instantiate and interact with Java objects directly. You can build larger Java applications in a proper Java environment and access Java methods in your SAS code. This way you can directly work with SAS variables without converting them to macro variables or passing them in any way. It is also excellent for processing SAS datasets row by row as you can apply Java methods to each observation.

The debugging and error handling are more difficult to manage properly. Furthermore, designing the Java methods so they work with this interface can be cumbersome at times because SAS only supports certain return types and no object-oriented things like classes or collections.

```
data _null_;
   set sashelp.class;

   /* Create a Java String object with "Hello World" message */
   declare javaobj j('java.lang.String', 'Hello World from JAVAOBJ!');

   /* Declare a variable to hold the result */
   length result $100;

   /* Call the toString method to get the string representation */
   j.callStringMethod('toString', result);

   /* Output the result to the SAS log */
   test = cat(name,' says: ',result);
   put test;

   /* Clean up the Java object */
   j.delete();
run;
```

System Commands

Depending on your computing environment this may or may not be an option for you. SAS provides several methods to execute external commands directly from a SAS program, e.g. X and PIPE command.
This is a great option for large scale tools and Java programs that contain more functionality than what can be expressed in a PROC GROOVY call. Here, Java runs as an independent process avoiding any conflicts with SAS. You can use any Java version and use any library and framework without SAS limitations.

However, it is difficult to pass complex datasets between SAS and Java and debugging from SAS is also quite difficult. In addition, you have to put in more effort to have proper logging and error handling on the Java side.

```
%LET java_bin = /etc/alternatives/java_sdk_11/bin/java;
%LET jar_location = /path/to/some/jar/java.jar;
FILENAME unxcmd PIPE "&java_bin -cp &jar_location. com.my.program.JavaEntryClass";
```

This is an example for a Unix environment. It uses a Java 11 installation and a custom Java JAR file. It uses the PIPE command to execute the Java program and store the output in a file called unxcmd which can then be read in a data step to process the log messages coming from Java.

Of course, you can adjust the shell command to make it fit your needs. For example, you can increase the memory available to the Java Virtual Machine or give the entry class some input arguments.

Java in SAS Summary

We have briefly looked at 3 different ways to use Java inside your SAS environment. As so often in programming, there is no definite way of doing it. You have to analyze what is best for your specific situation. In summary, here are some recommendations:

- Use PROC GROOVY for small script-like tasks (e.g. merge two PDFs)
- Use JAVAOBJ if you want to process data sets and do something row-by-row (e.g. do API call per observation)
- Use an external Java instance for large scale tools (e.g. create submission-ready Word documents from RTFs, images)

Practical Aspose Examples

Now that we have looked at what Aspose is and how you can use Java via SAS in different ways, this paper will go into some examples of what this can do for your daily work. The tasks may be trivial and quick to do manually but if you think on a larger scale, extracting hundreds of images or converting dozens of Word files to PDF will take a lot of time and is extremely error prone. So, automating it appears to be a great idea.
These are going to be smaller examples of very basic operations. Things that you can do even if you are not a Java developer but that still serve a purpose. The examples will limit themselves to using PROC GROOVY.

Let's first look at Aspose.Words which is the main use case for Aspose and a very mature tool. It allows for fine-grained editing of Word files and gives you control in a way that other Word libraries cannot offer at this level.

Convert DOCX to PDF

A simple task that is easy to do with Word but it can be pretty tough to get a good PDF rendition without having access to Word. Sometimes, having a PDF rendition can be very beneficial. If you have Listing documents with thousands of pages, it can be tough trying to open them with Word, it is very slow and searching them is almost impossible. A PDF on the other hand, opens instantly which is great for review. Also, if you want to share a document, it is much better to share a PDF because it is not dependent on the fonts you have installed as well as the printer drivers etc.

```
%LET basepath = /path/to/a/folder/phuse;
%LET docx = &basepath./word.docx;
%LET output_path = &basepath./converted.pdf;

PROC GROOVY;
    ADD CLASSPATH = "&basepath./aspose-words-20.9-jdk17.jar";
    SUBMIT "&docx." "&output_path.";

    import com.aspose.words.Document;
    import com.aspose.words.License;

    License tmpLic = new License();
    tmpLic.setLicense(Document.class.getClassLoader().getResourceAsStream("aspose.lic"));

    String file = args[0];
    String outputFileName = args[1];

    Document docx = new Document(file);
    docx.save(outputFileName);
```

```
    ENDSUBMIT;
QUIT;
```

First, we add Aspose to the classpath of PROC GROOVY so that it is available in the Java Virtual Machine. With SUBMIT you can pass some SAS variables to the Java part, here we specify which Word document to read and where to store the resulting PDF.

You need to import the Java classes used by your code. It is best to code in a Java environment and then copy over the necessary imports to the groovy code.

Before you use Aspose, you need to apply the license file, a ".lic" text file that you get after purchasing Aspose. Without it, Aspose goes into evaluation mode with very limited functionality. In this case, the license file is placed inside the JAR so that it is accessible via the class loader. You can do that as well, just open the Aspose JAR like a ZIP file and place the license file inside.

Now the passed SAS variables are read into Java variables.

Afterwards the input document is read and saved to a PDF file, and this is it. You will now see a PDF file pop up.

You may run into issues with fonts as Word documents heavily rely on the ones installed on your system. If so, you can tell Aspose where to look for fonts because it may not find the right directory on your system.

```
String fontPath = /path/to/some/fonts/;
List<FontSourceBase> tmpFontSrc = new ArrayList<>
(Arrays.asList(FontSettings.getDefaultInstance().getFontsSources()));
tmpFontSrc.add(new FolderFontSource(fontPath, true));
FontSourceBase[] tmpNewFontSrc = tmpFontSrc.toArray(new FontSourceBase[0]);
FontSettings.getDefaultInstance().setFontsSources(tmpNewFontSrc);
```

This code may seem complicated, but it just points Aspose to a folder that should contain a bunch of fonts. You can use this folder to maintain the fonts you are using for your documents and ensure that way that the PDF rendition works properly. You would not have to create this code yourself, the Aspose documentation and forum contain many occurrences of this problem and solution.

Extract Images from DOCX

At times you get a Word document with figures in it and you want to extract them to use them further down the line. Using Aspose, you can loop over all Shapes and then save the images separately without much hassle.

```
Document docx = new Document(file);
NodeCollection<Shape> shapes = docx.getChildNodes(NodeType.SHAPE, true);
int imageCount = 0;
for (Shape shape : shapes) {
    if (shape.hasImage()) {
        imageCount++;
        String imageFileName = outputPath + File.separator + "image_" + imageCount + ".png";
        shape.getImageData().save(imageFileName);
    }
}
```

This example is missing the PROC GROOVY wrapper, but it works just as the other examples.
The main point of this example is that you can use Aspose to search documents for certain node types like Tables, Paragraphs, Shapes and many more and then loop over them and apply certain actions.

In this case we loop over all Shapes, then filter for Shapes that contain actual images and save them separately as PNG files.

You can use this logic to analyze your input documents, apply actions to certain objects and change different parts of the document based on your specific needs.

Update Header

Perhaps you have a Compound ID or a study number or something similar in your header or footer of your output documents. It would be a pity if you did a typo, or some identifier changes and you would have to rerun all your analyses. You could use Aspose to see what is inside the Headers and Footers and fix it on the fly.

```
Document docx = new Document(file);
for (Section section : docx.getSections()) {
    for (HeaderFooter headerFooter : section.getHeadersFooters()) {
        Range headerRange = headerFooter.getRange();
        FindReplaceOptions options = new FindReplaceOptions();
        options.setMatchCase(true);
        options.setFindWholeWordsOnly(true);
        headerRange.replace("12345", "ABCDE", options);
    }
}
docx.save(outputPath);
```

Here, we loop over all sections and their headers and footers. There can be multiple headers/footers in a section (these options are available in Word: “Different First Page”, “Different Odd and Even Pages”). Additionally, after a Section break, the headers and footers can change as well.

Aspose offers some tools to do Search and Replace actions on Ranges of text. This is necessary since Word splits up Paragraphs into smaller Runs of text and these Runs specify font, color, etc. That means if you are searching through the Runs yourself looking for the string ‘12345’, you may find two runs ‘12’ and ‘345’ (if they have different color for example). Aspose’s FindReplaceOptions make it a lot easier to work with that.

As can be seen from this example, it is helpful to know how Word documents and Word’s DOM (Document Object Model) works in order to write the logic that you need for your use cases. Fortunately, the Aspose classes are also modeled after the DOM of Word so if you learn Aspose you will also learn a lot about the mechanics of Word itself.

Merge PDFs

Moving on to some examples that are using Aspose.Pdf to work on PDF files directly.

Merging PDFs and in the same sense editing them seems simple but depending on what software your organization allows you to have, it can be tricky. A simple Adobe Reader cannot do much in this regard. Doing it automatically on a large scale is an even bigger problem. With Aspose it is very easy to merge multiple PDFs, delete certain pages or do some editing.

```
%let basepath = /path/to/a/folder/phuse;
%LET pdf1   = &basepath./part1.pdf;
%LET pdf2 = &basepath./part2.pdf;
%LET output_path = &basepath./merge.pdf;

PROC GROOVY;
    ADD CLASSPATH = "&basepath./aspose-pdf-20.9-jdk17.jar";
    SUBMIT "&pdf1." "&pdf2." "&output_path.";

    import com.aspose.pdf.Document;
    import com.aspose.pdf.License;

    License tmpLic = new License();
    tmpLic.setLicense(Document.class.getClassLoader().getResourceAsStream("aspose.lic"));

    String file1 = args[0];
    String file2 = args[1];
    String outputFileName = args[2];

    Document pdf = new Document(file1);
    Document pdf2 = new Document(file2);
    pdf.getPages().add(pdf2.getPages());
    pdf.save(outputFileName);

    ENDSUBMIT;
```

```
QUIT;
```

This example has the same structure as the previous one, except now we are using Aspose.Pdf instead of Aspose.Words.

It reads in two PDFs, appends all pages from the second one to the first one and then saves a new PDF under the specified output path.

There is a concept called SaveOptions in Aspose with which you can adjust the saving process, e.g. choose which headings to convert to bookmarks and if they are expanded by default.

Of course, you can also add more logic to only include certain pages, add separator pages or add logging.

Extract Bookmarks

Depending on how your environment is set up and how many different analyses and reports you do on your study data, it can get pretty crowded. After the 50th set of TFL you might wonder if you haven't done this type of table before. There are multiple ways of getting there. One could be to collect all Bookmarks from your PDF documents and collecting them in a single document so you can search over multiple areas.

```
PROC GROOVY;
    ADD CLASSPATH = "&basepath./aspose-pdf-19.6-jdk17.jar";
    SUBMIT "&pdf.";

    import com.aspose.pdf.*;

    License tmpLic = new License();
    tmpLic.setLicense(Document.class.getClassLoader().getResourceAsStream("aspose.lic"));

    String file = args[0];

    Document pdf = new Document(file);
    for (OutlineItemCollection outline : pdf.getOutlines()) {
        extractBookmarkNames(outline);
    }

    void extractBookmarkNames(OutlineItemCollection outlines) {
        for (OutlineItemCollection subOutline : outlines) {
            System.out.println(subOutline.getTitle());
            if (subOutline.size() > 0) {
                extractBookmarkNames(subOutline);
            }
        }
    }
    ENDSUBMIT;
QUIT;
```

This code crawls through all bookmarks of the input PDF document and prints the bookmark title into the output. The difficulty here is that bookmarks can be nested which is reflected by the Aspose class model. So, an Outline (which is basically a bookmark) can have one or more Outlines as children. That is why you need to set up a recursive function that loops over all children until none are found anymore.

Conclusion

As you can see from the examples above, there are many ideas that can come to mind when you have Aspose at your disposal. It can be a bit intimidating at first to dive into such a large framework but it is very much worth it. Especially the Aspose.Words part is fantastic and allows you to control everything in your Word documents up to the last details. But also Aspose.Pdf and Aspose.Imaging offers very handy tools where it is hard to find something comparable on the market.

The examples here focus mainly on the reading and extraction of information from these documents. However Aspose also offers a wide range of functionality to create and edit content inside the documents.

It is dearly recommended to get a trial version of Aspose and try to fix your problems!

Recommended Reading

Katja Glaß: Let's get Groovy - Word and PDF processing in SAS:
https://www.lexjansen.com/phuse/2022/sm/PAP_SM04.pdf

Karl Kennedy: A GROOVY way to enhance your SAS life:
https://www.lexjansen.com/phuse/2019/ct/CT05.pdf

Magnus Mengelbier & Jan Skowronski: SAS® Talking via the Java Object Interface:
https://www.lexjansen.com/phuse/2008/ts/TS04.pdf

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Dominik Habel
Company: Bayer

Email: dominik.habel@bayer.de