# BSF4ooRexx/Java:

# Apache POI

Cookbook and Nutshell Examples for Creating Microsoft Office Documents for Word, PowerPoint, and Excel

Florian Frcena

12008842

Vienna University of Business and Economics

Business Information Systems Seminar (0068)

Supervisor: Univ. Prof. Mag. Dr. Rony Flatscher

Winter Semester 2023/2024

**Declaration of Authorship**

I, Florian Frcena, declare that I have written this seminar thesis by my own and without the help of a second person or unauthorized resources.

Whenever content was used directly or indirectly from other sources, the source has been referenced.

This seminar thesis has neither previously been presented for assessment, nor has it been published in Austria or abroad.

This seminar thesis is identical with the assessed paper and the paper which has been submitted in electronic form.

Date:

Signature:

# Table of Contents

# List of Figures

# 1.Introduction

This seminar paper is created for the fifth course of the business information systems focus, at the university for business and economics in Vienna. The goal is to give an overview how to create Microsoft Office documents, using the programming languages open object Rexx and Java, in combination with the Apache POI Java API. In chapter installation describes which software components were used to create the featured Nutshell examples. Furthermore, where to download and how to install and set them up properly. Overall, nine Nutshell examples are created for the three most common Microsoft Office applications, Word, PowerPoint, and Excel. For each application 3 different examples, each example deals with at least one new concept featured in the Apache POI API. There is a detailed explanation of how the Nutshell examples are coded in the chapters named like the application. The entire code of all Nutshell examples can be found in the appendix section. At the last chapter of the paper the conclusion a summary is written and there also mentioned some difficulties that occurred during the programming process.

All featured Nutshell examples are coded in the programming language open object Rexx (ooRexx). Open object Rexx is an open-source project by Rexx Language Association and is an interpreted programming language containing many features and uses a simple syntax. It is chosen as the programming language for the featured Nutshell examples, because ooRexx is easy to understand and to use also for people with no programming experience. Another benefit of ooRexx is that there are fewer rules required in your code like in other programming languages, e.g. Java. Furthermore, ooRexx is a message-based programming language, this means that it is possible that a created variable is able to send a message, using the tilde as the message operator, to perform a method related to the variable. In the figure below there is an example code snippet of the first Word Nutshell example. The variable “titlePara” is created and defined by the variable “document” and document sends the message to perform the “createParagraph()” method. Explained in words, this snippet shows how to create a new paragraph within a Word document, using a message in the correct ooRexx syntax. This is used in all Nutshell examples and will only be mentioned here. (Open Object Rexx, 2015)

```
20      titlePara=document~createParagraph()
```

*Figure 1 Visualising messaging in ooRexx.*

## 1.1 Apache POI-OpenXML4J

Apache POI-OpenXML4J is the Java API for Microsoft documents it is required to create and manipulate Microsoft documents using Java programs. Apache POI is developed and updated by the Apache Software Foundation as an open-source project. The API contains different components, one component for the following Microsoft Office applications, Excel, PowerPoint. Word, Outlook, and Publisher. For this paper these are the used components of the API, XWPF for dealing with Word documents, XSLF for dealing with PowerPoint documents and XSSF for dealing with Excel documents. To use the API a connection to Java is required, how to get the connection is explained in detail in chapter 2.4. On the Apache POI homepage is a detailed documentation, for all the different versions and components of the API, in the section "Javadocs" available. Alternatively, the java docs of the newest version of the API can be accessed directly via this link, https://poi.apache.org/apidocs/5.0/. There can be found a detailed listing of all available classes and methods for each component of the API. The documentation and the quick guides on the homepage of the API were a very helpful source for all coded Nutshell examples featured in this paper. (Apache Software Foundation 1, 2023)

## 1.2 BSF4ooRexx850

To use any Java libraries or classes including the Apache POI Java API in your ooRexx programs the bean script framework for ooRexx, called BSF4ooRexx850, is necessary. BSF4ooRexx is a package that creates a bridge between ooRexx and Java, both have to be installed correctly on your pc. If installed correctly it is necessary to use the "::Requires BSF.CLS" directive at the end of your code to get access to Java and all the classes and libraries featured in the Java class path environment variable. In the featured Nutshell examples there is also the ".bsf" function required in the code to get access to java classes. How to get to the class path and edit it is explained in the installation chapter 2.4 Apache POI. (Sourceforge 1, 2021)

# 2. Installation

This Chapter of the paper explains which software components are necessary to code or execute the featured Nutshell examples, where to download from and how to install them. Some of them are mandatory, ooRexx 5.0.0, as the ooRexx interpreter to code and execute ooRexx programs, BSF4ooRexx850, as the bridge from ooRexx to java, to get access to java classes within ooRexx. Java 1.8 or newer and Apache POI, the java API which supplies all the necessary classes and methods used in the featured Nutshell examples. An optional software component which allows easier coding than in the standard ooRexx GUI, is the integrated development environment IntelliJ IDEA by JetBrains, if IntelliJ IDEA is used an ooRexx plugin is required to get syntax highlights for the code and to create and execute ooRexx programs within IntelliJ IDEA.

## 2.1 ooRexx 5.0.0 and BSF4ooRexx850

Main software component of this paper is ooRexx 5.0.0, as it is the used programming language for the featured Nutshell examples. Any information about ooRexx can be found on the homepage of the Rexx language association, https://www.rexxla.org/. On the main page the download link for the ooRexx 5.0.0 interpreter can be found, or alternatively via this link, https://sourceforge.net/projects/oorexx/ which leads directly to the sourceforge ooRexx page and to the download button for the latest version of the ooRexx interpreter, it can be choosen between a 32-Bit or 64-Bit version, should match the bit rate of your operating system. After downloading and before installing the user has to agree in the properties of the ooRexx installer that this is a program of a trustworthy source. The ooRexx interpreter is required to execute ooRexx programs via the terminal. To execute a program, open the terminal in the folder the program is stored in, first you need the command “rexx” then a blank space and the exact name of the Rexx program, like in the figure below.


*Figure 2 Execution and output of ooRexx program.*

The link to the latest version of BSF4ooRexx can be found at the main page of the Rexx language association, or alternatively via this link which leads to the BSF4ooRexx sourceforge page, https://sourceforge.net/projects/bsf4oorexx/files/beta/20221004/. For all the Nutshell examples featured in this paper, BSF4ooRexx_v859-20230619-beta, was used. After downloading the latest BSF4ooRexx version as a .zip file, the file should be extracted in a folder in the program's directory. When extraction is finished the installation process begins, double click on the created folder, then double click on bsf4oorexx folder and there the folder install can be found. Within the install folder are featured the install packages for all major operating systems like Windows, Linux or MacOS. Each of the install packages contains an install, a reinstall and an uninstall file, for the first instalment the install file should be executed. (Sourceforge 1, 2021)

## 2.2 Java Zulu JDK 1.8

To get access to Java classes or libraries via BSF4ooRexx a Java version has to be installed on your pc. The first step is to check if Java is already installed on your pc, this can be easily done via the terminal on your desktop. Open the terminal and type the command "java" a blank space and then the command "-version", this gives the feedback if java is already installed and which exact java version. The output of my pc can be found in figure 3 below.

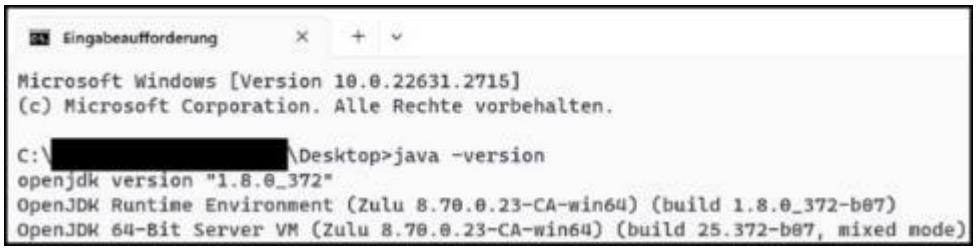



*Figure 3 Checking if Java is installed.*

The installed java version on my pc is the Java Zulu OpenJDK 64-Bit version "1.8.0_372" which can be downloaded at the Azul homepage accessible via this link, https://www.azul.com/downloads/?package=jdk#zulu. On the Azul homepage are all kinds of Java versions for all major operating systems available, from Java version 1.8, the minimum required java version for Apache POI, to the newest Java version 1.21.0. The chosen version should contain the JDK and the FX package and has to match the bit rate of the installed ooRexx version. At downloading can be chosen from two different files a .zip file or .msi file, I would

recommend downloading the .msi file, because after downloading a MSI installer is opened which leads through the installation process step by step.

### 2.3 IntelliJ IDEA ooRexx Plugin

All the Nutshell examples featured in the paper where coded in the integrated development environment IntelliJ IDEA by JetBrains, the use of IntelliJ IDEA is not required, but it is more convenient to code in IntelliJ IDEA than in the standard ooRexx GUI. IntelliJ IDEA can be downloaded on the official JetBrains homepage accessible here, https://www.jetbrains.com/idea/, you can choose between the ultimate or the community edition, the community edition is free to use and is completely sufficient. If you decide to use IntelliJ IDEA, then you need to install the ooRexx plugin for IntelliJ IDEA, because ooRexx is not featured in the standard version of IntelliJ IDEA. The newest version of the required plugin can be downloaded using this link here, https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/GA/. This link leads you to the sourceforge page of the ooRexx plugin. After downloading do not unzip the .zip file of the plugin, because the .zip file will be imported in IntelliJ IDEA. First start IntelliJ IDEA and press the register card Plugins on the left, then press the gear icon in the middle on the top of the page and there is the possibility to click, “Install Plugin from disk…”. After choosing this option you get the possibility to import the downloaded plugin zip file and to finish the process IntelliJ IDEA has to be restarted. (Sourceforge 1, 2021)

### 2.4 Apache POI

The Apache POI Java API can be downloaded on the official Apache POI website accessible via this link, https://poi.apache.org/download.html. This link leads to the download section of the website, if you want to download the necessary .jar files to use the API, you have to scroll down to the release archives section. In this section the source and binary artifacts are distributed, to use the Apache POI API in your code the latest distributed binary version is needed. The version used to code the featured Nutshell examples is Apache POI 5.2.3 and the poi-bin-5.2.3-20220909.zip was downloaded, this zip archive features all necessary .jar files which are required to work with the API, a list of all used .jar files can be found in the appendix. After downloading the zip archive extract it to a folder of your choice. To use the API in your Java code, you need to attach the downloaded Apache POI jar files to the java class path environment variable. On windows it works the following way, open the settings of your pc,

and click on system, then choose advanced system settings. This leads to a popout window with the button environment variables on the bottom left. Click the button to get to all available environment variables including the class path, after choosing the class path, it is possible to write the full path of the required .jar files in the class path environment variable. A possible way to write the .jar files in the class path can be found in figure 4 below.

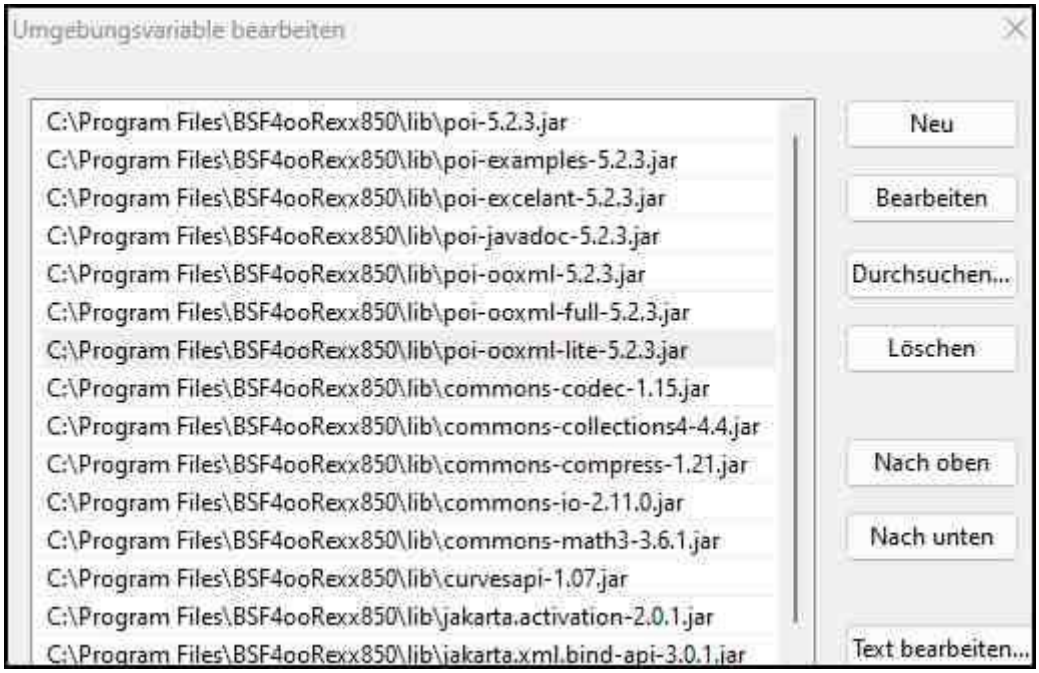



*Figure 4 Editing class path environment variable.*

# 3. Microsoft Word

Microsoft Word is a word processor designed and developed by Microsoft. Using the Java API Apache POI and the XWPF format it is possible to create and manipulate word documents and is used in all Microsoft Word Nutshell examples featured in this paper. The sources for the featured Word Nutshell examples are freely available coded Java examples on tutorials point, there can also be found a detailed explanation of Apache POI Word accessible via this link, https://www.tutorialspoint.com/apache_poi_word/apache_poi_word_quick_guide.htm. And further examples on mkyonk.com https://mkyong.com/java/java-read-and-write-microsoft-word-with-apache-poi/#document-header-and-footer. The provided Java examples are the foundation for the coded Nutshell examples using BSF4ooRexx.

## 3.1 First Nutshell Example

The first Nutshell example shows how to create a Word document and creating and editing text in paragraphs. At the end you can see how to create and store the output Word document. The second Nutshell example additionally shows how to create a header, a footer, and a table in a Word document. In the third Nutshell example accessing an existing Word document is explained and how to insert a new paragraph and an image.

First, we need to create a new unedited Microsoft Word document. To do this, we need to create a new instance of “org.apache.poi.xwpf.usermodel.XWPFDocument”,like in figure 5 below. This will give us access to all the available methods of the “XWPFDocument” class.

```
15  document=.bsf~new("org.apache.poi.xwpf.usermodel.XWPFDocument")
```

*Figure 5 Creating new Word document.*

### 3.1.1 Creating Paragraphs

In the figure below you will see how a paragraph is created and how to edit the text within a paragraph. To align a paragraph, we have to import the Class “ParagraphAlignment” as you can see in line 17. This Class allows you to align a paragraph on the left, the right or in the centre, of a page. To write a text into the created paragraph, you have to create a run for the paragraph. With in this run you are able to set an individual text with the “setText()” method and the text in brackets and quotes. It is possible to change the font type using the “setFontFamily() method, with the chosen font name in brackets, here “Bahnschrift” is used. To create a bold text the “setBold(1)” method is used, the 1 as the parameter of the method means true.

```
.bsf~bsf.importClass("org.apache.poi.xwpf.usermodel.ParagraphAlignment", "AligningParagraph")

-- Titel Paragraph
titlePara=document~createParagraph()
titleRun=titlePara~createRun()
titleRun~setBold(1)
titleRun~setFontSize(30)
titleRun~setFontFamily("Bahnschrift")
titleRun~setText("Enter Your Title Here")

--Aligning Titelparagraph
titlePara~setSpacingAfter(500)
titlePara~setAlignment(.AligningParagraph~CENTER)
```

*Figure 6 Creating paragraph and run.*

### 3.1.2 Set a Page Break

To create a paragraph on the second page of a word document you have to set a page break within the created paragraph using the “setPageBreak()” method and the attribute 1 in brackets to activate the page break. Now the created paragraph is displayed on top of the next page. The text of the paragraph can be set, aligned, and edited the same way as mentioned in the section above.

```
-- Next Page
p2=document~createParagraph()
p2~setWordWrapped(1)
p2~setPageBreak(1)
```

*Figure 7 Setting page break.*

### 3.1.3 Saving the Output File

To save the created document we need the “FileOutputStream” java class. The “FileOutputStream” java class allows one to set a filename and to store a file in any directory required. The created output document should be stored in the same directory, as the code is stored in. Therefore, the full path of the directory we are currently working in is needed. With the “parse source” built in method of ooRexx it is possible to parse the source of our coded programme and store it in a variable, here in this example “a”. We also need the local directory of the source stored in the used variable. Therefore, “call directory filespec(‘L’, a)” is needed to get the local directory of the source stored in variable a. Now it is possible to concatenate the chosen name of the document with the directory we are working in and store it in the variable filename. To correctly run our programme on different operating systems like Windows, Unix or MacOS you have to use a qualified file path.

When creating a new instance of “java.io.FileOutputStream” as the output for a document, one has to hand over the qualified filename in order to save the document correctly. At the end of the programme, it is necessary to send the created document the message “~write(output)” to get the write the output of the created program in a Word document with the previously determined name. The code is provided in figure 8 below.

```
49      -- Saving the output document
50      parse source . . a
51      call directory filespec('L', a)
52      filename=directory()"/First_Nutshell_Example.docx"
53      filename=qualify(filename)
54      output=.bsf~new("java.io.FileOutputStream", filename)
55      document~write(output)
```

*Figure 8 Saving output document.*

The explained way to save a document is used for all coded examples featured in this seminar thesis, no matter if it is a Word, PowerPoint, or Excel output file. One change is necessary, particularly the correct name of the output file and the file specification, .pptx for PowerPoint presentations and .xml for Excel workbooks.

## 3.2 Second Nutshell Example

This Nutshell example additionally explains how to create a header and a footer at a Word document and to insert a table of any chosen size in a Word document.

### 3.2.1 Creating Header or Footer and Setting Text

To create a header or footer for the created document, one have to load the “org.apache.poi.wp.usermodel.HeaderFooterType” class to get access to all header and footer related methods. It is possible to create three different header or footer types, namely default, even or odd. The default type means that the header or footer is placed on each page, the even type places headers or footers only on even pages and the odd type only on odd pages. For this nutshell example the default type for the header and footer was used. To create a header the “createHeader()” method is used and in brackets it is specified which header type should be used. After the header is created one has to create a paragraph within the header and in order to set a text one has to create a run like in a normal paragraph.

In this example a different variant was used to set the text of a paragraph. First the text of the header was stored in the variable “headerText”. During the run with the “setText” method the variable “headerText” was handed over and the text stored in the variable was set in the header. To create a footer it works the same way as to create a header, one has only to use the “createFooter()” method, instead of the “createHeader()” method, and the names of the used variables for easier visualisation. In figure 9 below is a section of how it is done in the featured example.

```
.bsf~bsf.loadClass("org.apache.poi.wp.usermodel.HeaderFooterType", "HeaderFooterType")

-- Creating Header
header=document~createHeader(.HeaderFooterType~DEFAULT)
headerParagraph=header~createParagraph()
-- Add text to the header
headerText="This is the header of this document."
run=headerParagraph~createRun()
run~setText(headerText)

-- Creating Footer
footer=document~createFooter(.HeaderFooterType~DEFAULT)
footerParagraph=footer~createParagraph()
-- Add text to the footer
footerText="This is the footer of this document."
run1=footerParagraph~createRun()
run1~setText(footerText)
```

*Figure 9 Creating header or footer.*

### 3.2.2 Creating a Table

To create a Table within a word document one has to use the “createTable()” method of the “XWPFDocument” class. This method generates a Table with no specific size. In order to get the first cell of the table one has to define the first row with the “getRow()” method. In this example 0 means the first row, because the index of BSF4ooRexx850 starts at 0. The first cell of the row is created automatically, further two more cells in the first row are created. If additional table rows are created, there are automatically generated three cells in each row. To set a text in a specific table cell one has to address the desired cell. If you want to set a text in the second cell of the second row, you have to define row 2 using the create row method and the method “getCell(1)” for cell two and the method "setText()” with your text in brackets and quotes.

```
-- Creating Table
table=document~createTable()
-- Addressing first row cells
row1=table~getRow(0)
row1~getCell(0)~setText("First Row, First Column")
row1~addNewTableCell()~setText("First Row, Second Column")
row1~addNewTableCell()~setText("First Row, Third Column")
row2=table~createRow()
row2~getCell(0)~setText("Second Row, First Column")
row2~getCell(1)~setText("Second Row, Second Column")
row2~getCell(2)~setText("Second Row, Third Column")
row3=table~createRow()
row3~getCell(0)~setText("Third Row, First Column")
row3~getCell(1)~setText("Third Row, Second Column")
row3~getCell(2)~setText("Third Row, Third Column")
```

*Figure 10 Creating a table.*

## 3.3 Third Nutshell Example

This Nutshell example is an extended version of the first Nutshell example. Additionally, to the first example this contains an extra paragraph and an image at the second page of the created Word document.

### 3.3.1 Accessing an Existing Word Document

Unlike with the two previous examples the Word document is not created directly at the beginning of the code. First, we need to get the full qualified path of the Word document which we want to access and store the path in a variable. Secondly a variable which contains a new "java.io.FileInputStream", with the previous created filename, is created. Now it is possible to create a new Word document with the file input variable as the parameter to get access to the desired document. Figure 11 gives an overview how it is done.

```
parse source . . c
call directory filespec('L', c)
filename1=directory()"/First_Nutshell_Example.docx"
filename1=qualify(filename1)

olddocument=.bsf~new("java.io.FileInputStream", filename1)

-- Create a new XWPFDocument for the output
document=.bsf~new("org.apache.poi.xwpf.usermodel.XWPFDocument", olddocument)
```

*Figure 11 Accessing existing Word document.*

### 3.3.2 Adding New Text and an Image

The additional paragraph is created the same way as paragraphs in the first Nutshell example, but the code has to be placed after the file input section. To insert an image into a Word document the two classes in line 17 and 18 of the figure below are needed, the document class is required to address the document when inserting an image. Same as a text a paragraph and a run must be created. The image has to be stored in the same directory as your coded example and the path must be qualified to get access to the image. Now a new file input stream with the image path is created. To add the picture to a run the "addPicture()" method is required. This method contains of 5 different necessary parameters. The input stream, stored in a variable, for the image, the definition of the image file format, in this example ".png" and the name of the png image. The last to parameters require the "org.apache.poi.util.Units" util class to set the width and the height of the inserted image, using the "toEMU" method and the size in points, here 250 points for height and width. Figure 12 contains all necessary lines of code to create a paragraph and inserting an image.

```
25  -- Add a new paragraph with text
26  newParagraph=document~createParagraph()
27  newRun=newParagraph~createRun()
28  newRun~setText("It was added after accessing the previously created Word document.")
29
30  .bsf~bsf.loadClass("org.apache.poi.xwpf.usermodel.Document", "Document")
31  .bsf~bsf.importClass("org.apache.poi.util.Units", "Units")
32
33  -- Create a new paragraph for the image
34  imageParagraph=document~createParagraph()
35  imageRun=imageParagraph~createRun()
36
37  -- Specify the path to the image file
38  imagePath=directory()"/java.png"
39  imagePath=qualify(imagePath)
40  image=.bsf~new("java.io.FileInputStream", imagePath)
41
42  -- Add the image to the document
43  imageRun~addPicture(image, .Document~PICTURE_TYPE_PNG, "java.png", .Units~toEMU(250), .Units~toEMU(250))
```

*Figure 12 Additionally adding text and image.*

# 4. Microsoft PowerPoint

PowerPoint is a presentation application developed and designed by Microsoft. Via the Java Api Apache POI and the XSLF format it is possible to create and manipulate PowerPoint documents and is used in all Microsoft PowerPoint nutshell examples featured in this paper. The sources for the coded PowerPoint Nutshell examples are coded Java examples freely available on tutorials point and for further detailed explanation of the possibilities of Apache POI XSLF API component, accessible direclty via this link, https://www.tutorialspoint.com/apache_poi_ppt/apache_poi_ppt_quick_guide.htm. The second source used is baeldung.com, where also is explained how to create a PowerPoint presentation using Apache POI available here, https://www.baeldung.com/apache-poi-slideshow.

The first Nutshell example shows how to create a new presentation and slides with a specific layout and custom text boxes. Additionally shows the second Nutshell example how to insert a hyperlink to a slide and a table to a custom created text box. The last Nutshell example explains how to merge the presentations of the first and second example together in a single presentation and how to change the order of slides.

## 4.1 First Nutshell Example

First, we have to create a new unedited Microsoft PowerPoint document. In Apache POI a PowerPoint presentation is called slideshow. Therefore, we have to create a new instance of the "org.apache.poi.xslf.usermodel.XMLSlideShow" class. This instance allows us to get access to all available methods and classes of the "XMLSlideShow" class.

```
15    presentation=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow")
```

*Figure 13 Creating new PowerPoint slideshow.*

### 4.1.1 Creating Layout Slides

In PowerPoint exists a so-called slide master. With in the slide master the layouts of all standard slides, like title slide, title only slide or title and content slide, are stored. To get access to all slides of the slide master the method "getSlideMasters()" is used and the method "get()" with the attribute 0 in brackets to retrieve all available master slides. To create a slide with a specific layout the class "org.apache.poi.xslf.usermodel.SlideLayout" is required. Now it is possible to store a slide layout in a variable. The first slide created in this example is a title slide. As you

can see in line 7 in figure 14 below. Furthermore, in this example a slide with only a title and a slide with title and content are created.

```
19  -- Creating slide layouts
20  template=presentation~getSlideMasters()~get(0) -- retrieves all master slides
21  layout=template~getLayout("title slide") -- Set Layout
22  layout1=template~getLayout("title only")
23  layout2=template~getLayout("title and content")
24
25  -- Creating title slide
26  titleslide=presentation~createSlide(layout)
```

*Figure 14 Creating different slide layouts.*

### 4.1.2 Accessing Placeholders

Whenever a slide with a specific layout is created, there are also created different textboxes automatically referring to the created layout, so called placeholder. To get access to a placeholder the method “getPlaceholder()” is used. To get access to the first placeholder on a slide the first index of the placeholder has to be addressed. As the programming language ooRexx is used in all examples the first index starts always with zero. After Accessing the placeholder, it is possible to set an individual text. If the slide layout provides more than one placeholder it is possible to access them with their correct index. Placeholder two is accessed by index 1 and placeholder three by index 2. Figure 15 explains how to set texts in placeholders of a title and content slide.

```
58  secondslide=presentation~createSlide(layout2)
59  secondTitleTextbox=secondslide~getPlaceholder(0)
60  secondTitleTextbox~setText("Here is a Slide With Bullet Points")
61
62  ContentTextbox=secondslide~getPlaceholder(1)
63  ContentTextbox~setText("First Text")
```

*Figure 15 Accessing placeholders on slides.*

### 4.1.3 Creating Custom Textboxes

To create a custom textbox the class “java.awt.Rectangle” and the “createTextBox()” method are required. Each textbox needs a so-called anchor, with a new created rectangle, to determine the location and size of the textbox at a slide. Therefore the method “setAnchor()” is used and all necessary input parameters can be found in line 47 of figure 16 below. After the creation of

the textbox, it is possible to create paragraphs and text runs with individual text the same way as in automatically created placeholders.

```
43      .bsf~bsf.importClass("java.awt.Rectangle", "Rectangle")
44
45      -- Creating custom textbox
46      textbox=firstslide~createTextBox()
47      textbox~setAnchor(.Rectangle~new(50, 150, 400, 200))
48      paragraph=textbox~addNewTextParagraph()
49      run=paragraph~addNewTextRun()
50      run~setText("Text in a custom created textbox.")
51      paragraph1=textbox~addNewTextParagraph
52      run1=paragraph1~addNewTextRun()
53      run1~setText("Second paragraph of the textbox.")
```

*Figure 16 Creation of custom textboxes.*

## 4.2 Second Nutshell Example

In addition to the first Nutshell example, this example shows how to insert hyperlinks and a table within a created textbox.

### 4.2.1 Creating Hyperlinks

In PowerPoint it is possible to embed a hyperlink to any website to a previous set text. For this part of the example a “title and content” slide with two placeholders is used. The first placeholder contains the title of the slide. The second placeholder needs to be cleared from the automatically created standard text of the content placeholder. After clearing the text contained in the placeholder and creating a new paragraph and text run, the text to be converted into a hyperlink is set. Now the variable link is used to create a hyperlink within the previous created text run, using the "createHyperlink()” method. Now the variable link needs to be linked to the URL of the requested website, using the “setAddress()” method with the full path to the website in brackets featured in figure 17 below.

```
24      -- Hyperlink at Slide 1
25      firstslide=presentation~createSlide(layout1)
26      Textbox=firstslide~getPlaceholder(0)
27      Textbox~setText("Hyper Link to the Apache POI Javadocs")
28
29      Textbox1=firstslide~getPlaceholder(1)
30      Textbox1~clearText()
31      textRun=Textbox1~addNewTextParagraph()~addNewTextRun()
32      textRun~setText("Apache POI Javadocs")
33      link=textRun~createHyperlink()
34      link~setAddress("https://poi.apache.org/apidocs/5.0/")
```

*Figure 17 Creation of hyperlinks.*

### 4.2.2 Creating a Table Textbox

To create a table textbox the “java.awt.Rectangle” class is needed, like in the creation of a normal textbox. The table is created at the second slide using the “createTable()” method and needs an rectangle as an anchor to determine the place and the size of the table textbox. Two variables are defined to set the amounts of the rows and columns of the table, in this example the table contains three rows with two columns each. To create the table in an efficient way an outer and an inner do-loop is used in the code featured in figure 18 below. The outer do-loop is required to iterate over the variable “rowCount” to add a new row to the table and reduce the value at each pass by one and stops when the row counter “rowCount” hits zero. Within the outer do-loop the inner do-loop is required to create cells for the created rows. The inner do-loop iterates over the variable “colCount” to add all required cells, set text to them with automatic numbering and set the size of each column to 150 points at the previous created row. The inner do-loop stops when the column counter “colCount” hits zero. In this example a table with three rows and two cells in each row are created.

```
-- Insert a table on slide1
table=secondslide~createTable()
table~setAnchor(.Rectangle~new(50, 150, 500, 200))

-- Add rows and cells to the table
rowCount=3
colCount=2
do row=0 to rowCount -1
    tableRow=table~addRow()
    do col=0 to colCount -1
        tableCell=tableRow~addCell()
        tableCell~setText("Row" row+1 ", Column" col+1)
        table~setColumnWidth(col, 150)
    end
end
```

*Figure 18 Creation of a table within a textbox.*

## 4.3 Third Nutshell Example

This Nutshell example combines the two slide shows created in the two examples explained above and shows how to get access to previous created slides and change their order.

### 4.3.1 Opening the Existing PowerPoint Presentations

Accessing an existing PowerPoint presentation works the same way as explained in chapter 3.3.1. Otherwise, then in that example it is necessary to create two independent “java file input streams” one for each presentation and store the two input streams in individual variables, here “firstoldpresentation” and “secondoldpresentation”.

```
parse source . . g
call directory filespec('L', g)

firstfilename=directory()"/First_Nutshell_Example.pptx"
firstfilename=qualify(firstfilename)
firstoldpresentation=.bsf~new("java.io.FileInputStream", firstfilename)

secondfilename=directory()"/Second_Nutshell_Example.pptx"
secondfilename=qualify(secondfilename)
secondoldpresentation=.bsf~new("java.io.FileInputStream", secondfilename)
```

*Figure 19 Opening PowerPoint slideshows of first and second Nutshell example.*

### 4.3.2 Merge two Presentations

The “java.io.FileInputStream” class allows one to import an existing file, here a “.pptx” file, one to one, however it is only possible to import one file per file input stream, so there is a little work around needed to achieve the goal of merging two PowerPoint presentations together. First one has to create a new java array list, using the “java.util.ArrayList” class, to store all slides from both presentations in this array list. Secondly, two new slide shows were created using the variables in which the two separate java file input streams were stored previously. The next step is to get access to all slides of both presentations using the “getSlides()” method. Now it is possible to put them all in the previous created java array list, defined in the variable “slidesList”, using the “addAll()” method, with the name of the used slide show and the “getSlides()” method in brackets. This has to be done twice, one time for each presentation and is featured in line 30 and line 31 in figure 20 below. Only now is the PowerPoint presentation created that serves as the output for this example.

```
26  -- Create an ArrayList to hold the slides
27  slidesList=.bsf~new("java.util.ArrayList")
28
29  -- Create XMLSlideShow instances for each input file
30  xmlSlideShow1=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow", firstoldpresentation)
31  xmlSlideShow2=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow", secondoldpresentation)
32
33  -- Add slides from the first presentation to the list
34  slidesList~addAll(xmlSlideShow1~getSlides())
35
36  -- Add slides from the second presentation to the list
37  slidesList~addAll(xmlSlideShow2~getSlides())
38
39  -- Create a new XMLSlideShow instance
40  presentation=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow")
```

*Figure 20 Storing slides of both slideshows in a java arraylist.*

If you would import the content of all slides using the “importContent” method and open the created PowerPoint presentation now, you would mention that the alignment of the textboxes of the created slides is way different than in the presentations created in the first or second Nutshell example. I assume that’s because the layouts of the slides were set individually, to solve that problem the same layouts for the slides were created, as used in the other nutshell examples. Now it is possible to set the correct layout for each individual slide using a do-loop and a select-loop. The do-loop is required to iterate over the “slidesList” variable to create the same number of slides stored in that list. After getting a slide the select-loop figures out which

slide gets which layout. The way the select-loop is implemented in this example only works here, because it is known in which layout the original slide was created. The first slide is the title slide, so therefore when it is chosen the layout of the slide will be set to “layout”. If the second and fourth slide is chosen, there will be created a slide with the layout “layout1” and for all other slides a slide with the layout “layout2” is created. The last step of the do-loop is to import the content of the original slide on the correct layout using the “importContent()” method with the parameter “slide” in the brackets, which refers to the previously created slide in the select-loop. The do-loop is stopped automatically after all slides that are stored in the java array list were created. The hole loop is featured from line 48 to line 58 in figure 21 below.

```
42  -- Retrieve slide master and layouts
43  template=presentation~getSlideMasters()~get(0)
44  layout=template~getLayout("title slide")
45  layout1=template~getLayout("title only")
46  layout2=template~getLayout("title and content")
47
48  -- Add the slides from the list to the new presentation with specific layouts
49  do slide over slidesList
50      select
51          when slide==slidesList~get(0) then newSlide=presentation~createSlide(layout)
52          when slide==slidesList~get(1) then newSlide=presentation~createSlide(layout1)
53          when slide==slidesList~get(3) then newSlide=presentation~createSlide(layout1)
54          otherwise newSlide=presentation~createSlide(layout2)
55      end
56      -- Import content from the original slide
57      newSlide~importContent(slide)
58  end
```

*Figure 21 Importing slides and content.*

### 4.3.3 Changing the Slide order

After all slides were created correctly it is possible to change the order of the slides. PowerPoint memorizes the order of all slides with in a slide show Therefore the “getSlides()” method is used to get access to all available slides of the slide show. Now it is possible to select any slide needed with the “get()” method and the index of the slide as the parameter in brackets, here index three to get the fourth slide of the presentation, as featured in figure 22. The chosen slide is stored in the variable “selectedslide”. The slide order can now be changed with the “setSlideOrder()” method with the selected slide stored in the variable “selectedslide” and the index where the selected slide should be placed, as the parameters in brackets. In this example the fourth slide is placed in the front of the presentation addressing the index zero.

```
60  -- Changing slide order of the presentation
61  slides=presentation~getSlides()
62  selectedslide=slides~get(3)
63  presentation~setSlideOrder(selectedslide, 0)
```

*Figure 22 Changing slide order.*

# 5. Microsoft Excel

Excel is a spread sheet application developed and designed by Microsoft. Via the Java Api Apache POI and the XSSF format it is possible to create Excel documents and is used in all Microsoft Excel Nutshell examples featured in this paper. The sources for the coded Excel Nutshell examples are coded Java examples freely available on tutorials point and for further detailed explanation of the possibilities of Apache POI XSSF, accessible here, https://www.tutorialspoint.com/javaexamples/java_apache_poi_excel.htm. And for the third Nutshell example the HowToDoInJava website which explains how to create Excel formulars using Apache POI, https://howtodoinjava.com/java/library/readingwriting-excel-files-in-java-poi-tutorial/.

The first Nutshell example shows how to create an Excel workbook and a spreadsheet within this workbook and how to set and edit text. In addition to that, the second Nutshell example shows how to rotate the content of cells and to set various hyperlinks. Also merging cells and setting the print area of the document is featured. The third nutshell example deals with creating numbers and how to create various formulars.

## 5.1 First Nutshell Example

This example shows how to create an Excel workbook and a spreadsheet and deals with text editing and setting.

### 5.1.1 Creating Workbooks and Spreadsheets

If you want to create an Excel document, you have to instantiate two new instances, unlike the Word or PowerPoint examples where only one instance is necessary. You need one instance of the "org.apache.poi.xssf.usermodel.XSSFWorkbook" class to create a new workbook and to get access to all available methods and classes of the "XSSFWorkbook" class. Secondly, a new instance of the "org.apache.poi.xssf.usermodel.XSSFSheet" class is required to create a new

spreadsheet. Now it is necessary to create a new spreadsheet with in the previous created workbook using the “createSheet()” method, with the name of the created spreadsheet in brackets, here featured in figure 23 below the spreadsheet named “Cell Types” is created.

```
15  workbook=.bsf~new("org.apache.poi.xssf.usermodel.XSSFWorkbook")
16
17  spreadsheet=.bsf~new("org.apache.poi.xssf.usermodel.XSSFSheet")
18
19  -- Create a new sheet
20  spreadsheet=workbook~createSheet("Cell Types")
```

*Figure 23 Creating new Excel workbook and spreadsheet.*

### 5.1.2 Setting Text and Font

To edit the text of a cell you first have to create a new font, using the “createFont()” method. Now it is possible to set the size of the font in points, using the “setFontHeightInPoints” method, here the size of 20 points is chosen. The font “GOUDY STOUT” is used with the “setFontName() method and a text effect is set using the “setItalic()” method and the input parameter “.true” in brackets, to set the created font to italic. To change the colour of the font you need to get access to the so called “CreationHelper” this helper allows you to use the “createExtendedColor” method and to set the text using the “setARGBHex()” method to any hex colour needed, here the hex value “000080” is used as the parameter to set the font colour to dark blue.

```
22  -- Creating a new font
23  font=workbook~createFont()
24  font~setFontHeightInPoints(20)
25  font~setFontName("GOUDY STOUT")
26  font~setItalic(.true)
27  -- Setting font color to blue
28  colour=workbook~getCreationHelper()~createExtendedColor()
29  colour~setARGBHex("000080")
30  font~setColor(colour)
```

*Figure 24 Creating new font.*

Now, after the font is created, a custom style has to be created and stored in a variable using the "createCellStyle()" method and set your created font to the style. If you want to put some content in a cell, first a row has to be created with the "createRow()" method and the index of the created row as the parameter in brackets, here index zero is used to create the first row. It is possible to get access to any cell of the row, using the "createCell()" method, with the index of the created cell in brackets, here the index zero is used to create the first cell of the row. After creating a cell it is possible to insert content in it using the "setCellValue()" method, this method can be used to set a text or numbers into a cell. The last step is to edit the content of the cell using the "setCellStyle()" method with the variable, which contains the style of your font, here in figure 25 below the variable "style" is used.

```
31    -- Setting font into style
32    style=workbook~createCellStyle()
33    style~setFont(font)
34
35    -- Create headers
36    row=spreadsheet~createRow(0)
37    cell=row~createCell(0)
38    cell~setCellValue("Type of Cell")
39    cell~setCellStyle(style)
```

*Figure 25 Creation of a cell style and applying it to a cell.*

### 5.1.3 Setting Different Cell Formats

If a cell is created in Excel using Apache POI, the standard cell format is used for each created cell. It is possible to change the cell format manually, in this example the cell formats date, numeric and currency are featured. The creation of the cell format date is explained in detail in figure 26 below and the other formats work quite similarly and are featured in the entire code in the appendix. First a row and a cell are required, and a style has to be created the same way as explained in the section above. A helper has to be created using the "getCreationHelper()" method. Now the desired cell format can be created with the "setDataFormat()" method and three different parameters in brackets. The previous created helper and a new data format has to be created using "createDataFormat()" method. The last required parameter is the chosen format accessible through the "getFormat()" method, with the desired data format in brackets, here "m/d/yy" as the date format. The prepared style can now be applied to the created cell.

```
62      row = spreadsheet~createRow(3)
63      cell=row~createCell(0)
64      cell~setCellValue("Set cell value DATE")
65      cell~setCellStyle(style1)
66      cell = row~createCell(1)
67      cell~setCellValue(Date())
68      style2=workbook~createCellStyle
69      helper=workbook~getCreationHelper()
70      style2~setDataFormat(helper~createDataFormat()~getFormat("m/d/yy"))
71      cell~setCellStyle(style2)
```

*Figure 26 Creating and setting cell formats.*

### 5.1.4 Setting Column Size

To change the size of a column there are two possibilities, setting an auto size, using "autoSizeColumn()" method with the column index in brackets, or setting an individual column width using "setColumnWidth()" method, this method requires two parameters, the index of the column and the width of the column in points.

```
103     -- Automatically size columns
104     spreadsheet~autoSizeColumn(0)
105     spreadsheet~autoSizeColumn(1)
106     spreadsheet~setColumnWidth(2, 2000)
```

*Figure 27 Changing size of columns.*

## 5.2 Second Nutshell Example

In addition to the first Nutshell example this example explains how to create rotated texts, setting three different types of hyperlinks, merging cells, and setting the print area of an Excel spreadsheet.

### 5.2.1 Rotating Content of Cells

With in a created style, it is possible to rotate the content of a cell in an angle between 0 and 360 degrees. To apply a rotation to a text the "setRotation()" method is required with the chosen angle in degrees in brackets, here 90 is used to rotate the text in an 90 degrees angle, featured in figure 28 below. The last required step is to set the created style to any previous created cell.

```
20      -- 90 degrees angle
21      myStyle=workbook~createCellStyle()
22      myStyle~setRotation(90)
23      row=spreadsheet~createRow(0)
24      cell=row~createCell(3)
25      cell~setCellValue("90 degrees Angle")
26      cell~setCellStyle(myStyle)
```

*Figure 28 Rotating cell content.*

### 5.2.2 Setting Various Hyperlinks

In an Excel spreadsheet it is possible to set a hyperlink to an URL, to an existing file in the current directory or to another spreadsheet within the current workbook. The first step is to create a variable with the chosen hyperlink URL featured in line 30 of figure 29 below. To set a hyperlink to a file the entire name of the file is required. A hyperlink to a spreadsheet has to be defined the following way, first set a hashtag than the name of the spreadsheet an exclamation mark after and the cell which should be accessed at the other spreadsheet, in this example the spreadsheet test is linked and after clicking the link the cursor jumps to cell “A1” of the spreadsheet test. The exact description of the used hyperlinks can be found in figure 29 below.

```
42      -- Set a hyperlink formula to an URL
43      hyperlinkFormula='HYPERLINK("https://poi.apache.org/apidocs/5.0/")'
44
45      -- Set a hyperlink formula to file Second_Nutshell_Example.xlsx
46      filelinkFormula='HYPERLINK("First_Nutshell_Example.xlsx")'
47
48      -- Set a hyperlink formula in cell B3 to the "test" sheet in the same workbook
49      spreadsheetlinkFormula='HYPERLINK("#test!A1")'
```

*Figure 29 Various hyperlink possibilities.*

In the second step a row and two cells of the row are created, one cell for the description to which URL the hyperlink is linked to, and the second cell contains the hyperlink stored in the variable “hyperlinkFormula”. To connect the hyperlink to a cell, the “setCellFormula()” method is used, this method contains the variable with the hyperlink as an attribute in brackets and quotes.

```
-- Set a hyperlink formula to an URL
hyperlinkFormula='HYPERLINK("https://poi.apache.org/apidocs/5.0/")'
rowHyperlink=spreadsheet~createRow(2)
cell=rowHyperlink~createCell(0)~setCellValue("Hyperlink to Apache POI Javadocs")
cellHyperlink=rowHyperlink~createCell(1)
cellHyperlink~setCellFormula(hyperlinkFormula)
```

*Figure 30 Creating a hyperlink to a URL.*

### 5.2.3 Merging Cells and Setting Print Area

Using Apache POI allows you to merge any cells of a spreadsheet, for this the "addMergedRegion()" method is used to merge cells. This method requires a new instance of the "org.apache.poi.ss.util.CellRangeAddress" util class. The class allows you to select the cells which should be merged, furthermore the start and end row and the start and end column are needed. In this example the attributes "0, 0, 7, 8" are used to merge cell H and I of the first row. To set up the print area of a workbook the "setPrintArea()" is used with the index parameters, for which spreadsheet of the workbook the print area has to be set and which columns and rows the print area contains, in brackets. Here in figure 31 the index parameters "0, 0, 10, 0, 5" are used to set the print area of the first spreadsheet, from the first to the eleventh column and the first to the sixth row. After the print area is set up we need to get access to it using the "getPrintSetup()" method and the paper for the printer printout must be defined using the "setPaperSize()" method, with the index of the desired paper in brackets. In this example the index 9 is used to set the printer paper to A4. If you want to have the Excel gridlines on your printout, you need to set the attribute of the "setPrintGridlines()"methods in brackets to ".true".

```
-- Merging Cells
spreadsheet~addMergedRegion(.bsf~new("org.apache.poi.ss.util.CellRangeAddress", 0, 0, 7, 8))
-- Set print area with indexes
workbook~setPrintArea(0, 0, 10, 0, 5)
-- Set paper size
spreadsheet~getPrintSetup()~setPaperSize(9)
-- Set print grid lines
spreadsheet~setPrintGridlines(.true)
```

*Figure 31 Cell merging and setting print area.*

## 5.3 Third Nutshell Example

The last Excel Nutshell example deals with the creation of random numbers using a built in excel function and how to create formulars to evaluate the created numbers.

### 5.3.1 Setting Random Numbers

To set 100 random numbers in column A, a do-loop is required to iterate over 100 cells of the column. First the indexes of the rows for the iteration needs to be defined in variable “i” “3 to 103”, here it creates row 3 to row 102. In each created row the first cell of the row will be created and a random number from 0 to 1000 will be randomly created using the “setCellFormula()” method and the built in Excel formula “Randbetween(0, 1000) as the attribute in brackets. The entire loop is featured from line 30 to line 35 in figure 32 below.

```
30    -- Creating a formula to generate random numbers in column A
31    do i=3 to 102
32        row=spreadsheet~createRow(i - 1)
33        cell=row~createCell(0)
34        cell~setCellFormula("Randbetween(0, 1000)")
35    end
```

*Figure 32 Creating and setting random numbers.*

### 5.3.2 Basic Evaluation of Numbers using Formulars

In Excel it is possible to set various formulars to deal with numbers, for example sum up values of a desired area, count nonblank cells or count cells with a specific value. Just like as creating hyperlinks, you need the “setCellFormula() method to use all the mentioned Excel built in formulars featured in this example in figure 33 below. You have to use the English names of the formulas in the code, no matter in which language you use Excel, because Excel will translate the formulars in the standard language Excel is used on your pc. If you want to sum all random created values up, the sum formular is the way to go. The “setCellFormula()” method requires two attributes “SUM(A3:A102)”, “SUM” is the name of the Excel formula and (A3:A102) is the range of cells which should be summed up. Two different ways to count cells are featured in this example, the “COUNT” formula which counts all cells with a numeric value and the “COUNTIF” formula which counts cells that fulfil a certain criterion. The criterion here is to count all cells with a minimum value of 500, described as “>=500” in line 34 of figure 33.

```
37      -- Summing up the values in column A
38      row1=spreadsheet~createRow(1)
39      formulaCellSum=row1~createCell(1)
40      formulaCellSum~setCellFormula("Sum(A3:A102)")
41
42      -- Creating formula for counting non-empty cells in column A
43      formulaCellCount=row1~createCell(2)
44      formulaCellCount~setCellFormula("Count(A3:A102)")
45
46      -- Creating formula for counting values in column A at least 500
47      formulaCellCountAtLeast500=row1~createCell(3)
48      formulaCellCountAtLeast500~setCellFormula("Countif(A3:A102, ""=500"")")
```

*Figure 33 Basic Excel formulars, Sum, Count and Countif.*

### 5.3.3 Complex Evaluation of Numbers using Formulars

This part of the example deals with statistical evaluation of the random created numbers. The Excel formulars used are “Min(A3:A102)” to get the lowest number, or “Max(A3:A102)” to get the highest number of the given cell range. To get the arithmetic average of the given cell range, the formula “Average(A3:A102)” is used. Furthermore, the median of the values in column A is determined by the use of the “Median(A3:A102) formula. Last formula featured in this example is “Stdev(A3:A102)”, this formula calculates the standard deviation of the cells A3 to A102.

```
50      -- Creating formula for the minimum value in column A
51      formulaCellMin=row1~createCell(4)
52      formulaCellMin~setCellFormula("Min(A3:A102)")
53
54      -- Creating formula for the maximum value in column A
55      formulaCellMax=row1~createCell(5)
56      formulaCellMax~setCellFormula("Max(A3:A102)")
57
58      -- Creating formula for the average value in column A
59      formulaCellAvg=row1~createCell(6)
60      formulaCellAvg~setCellFormula("Average(A3:A102)")
61
62      -- Creating formula for the mean (median) value in column A
63      formulaCellMedian=row1~createCell(7)
64      formulaCellMedian~setCellFormula("Median(A3:A102)")
65
66      -- Creating formula for calculating the standard deviation of column A
67      formulaCellStdDev=row1~createCell(8)
68      formulaCellStdDev~setCellFormula("Stdev(A3:A102)")
```

*Figure 34 Statistical evaluation using Min, Max, Average, Median and Stdev.*

# 6. Conclusion

This chapter summarizes the paper and explains some of the difficulties occurred during the programming process of the Nutshell examples relating to the Apache POI Java API.

The goal of this seminar paper is to act as a cookbook for readers, how to use a Java API in a program coded in ooRexx, using BSF4ooRexx. In the first chapter a general introduction is given what the paper is about and why the used Java API and the programming language ooRexx were chosen. A very vital part for a programming cookbook is, what software components are required, how to download and install them, that all the required software components work correctly together, this is explained in detail in chapter two installation. Main part of the paper are the coded Nutshell examples, which create Microsoft Office documents, the featured Microsoft Office applications are Word, PowerPoint, and Excel. Each application contains an individual chapter and within each chapter there is given a detailed explanation of how to create documents using selectively chosen code snippets. The entire code of all featured Nutshell examples can be found in the appendix section at the end of the paper.

The provided classes and methods from the Apache POI are easy to understand for what reason they are used for and if anything was unclear, I studied the official documentation of the API. In combination with coded Java examples, it was not very difficult to code useful ooRexx Nutshell examples, which explains how to reproduce them for anyone.

For the created Word Nutshell examples the XWPF component of the Apache POI was used, this component provides all the necessary classes and methods. This component provides the least possibilities for creating and manipulating documents compared to the other two components, in the featured examples, only the “org.apache.poi.xwpf.usermodel” package was used. Originally planned was to use the “org.apache.poi.xwpf.extractor” package, which features classes and methods to extract text from an existing word document, but there occurred some errors which I could not resolve and the decision was made to access an existing word document and modify it.

In the PowerPoint Nutshell examples the XSLF component and the “org.apache.poi.xslf.usermodel” package was used. This package features all the necessary classes and methods for creating the slideshows in the featured examples. The third Nutshell example was quite challenging, because I had to figure out how to merge two individual presentations in to one presentation using the “java.io.FileInputStream” class.

The XSSF component of the API for Excel provides in the “org.apache.poi.xssf.usermodel” by far the most possibilities for creating  Excel documents. It is possible to deal and edit text, to insert various types of hyperlinks and to use and create Excel formulars with in your code. Another possible package is “org.apache.poi.xssf.usermodel.helpers” this package provides classes and methods for protecting created Excel workbooks, but I was not able to implement that in a correct working way and refused to use it.

For this paper the latest available binary distributed version of Apache POI, 5.2.3 was used. On the official homepage there is mentioned that the source package Apache POI 5.2.5 is available for download, but this package is only required for creating the API and not for using it in your code.

# References


Apache Software Foundation. (2023, November 2) *The Java API for accessing Microsoft Open XML documents.* Retrieved Oktober 12, 2023 from https://poi.apache.org/components/oxml4j/index.html.

Apache Software Foundation 1. (2023, November 2) *Apache POI component overview.* Retrieved Oktober 12, 2023 from https://poi.apache.org/components/.

Apache Software Foundation 2. (2023, November 2) *Api documentation for version 5.*.* Retrieved Oktober16, 2023 from https://poi.apache.org/apidocs/5.0/.

Tutorials Point. (2023, Oktober 1) *Apache POI Java tutorial for Word.* Retrieved Oktober 21, 2023 from https://www.tutorialspoint.com/apache_poi_word/apache_poi_word_quick_guide.htm.

Mkyong. (2023, Oktober 5) *Reading and writing Microsoft Word documents with Apache POI.* Retrieved Oktober 23, 2023 from https://mkyong.com/java/java-read-and-write-microsoft-word-with-apache-poi/#document-header-and-footer.

Tutorials Point 1. (2023, Oktober 1) *Apache POI Java tutorial for Word.* Retrieved November 6, 2023 from https://www.tutorialspoint.com/apache_poi_ppt/apache_poi_ppt_quick_guide.htm.

Baeldung. (2022, May 26) *Apache POI Slideshow example.* Retrieved November 6, 2023 from https://www.baeldung.com/apache-poi-slideshow.

HowToDoInJava. (2023, August 27) *Reading writing Excel files in Java POI tutorial.* Retrieved November 22, 2023 from https://howtodoinjava.com/java/library/readingwriting-excel-files-in-java-poi-tutorial/.

Tutorials Point 2. (2023, Oktober 1) *Apache POI Java tutorial for Excel.* Retrieved November 22, 2023 from https://www.tutorialspoint.com/javaexamples/java_apache_poi_excel.htm.

Sourceforge. (2023, May 17) *Downloading ooRexx 5.0.0.* Retrieved Oktober 20, 2023 from https://sourceforge.net/projects/oorexx/files/oorexx/.

Open Object Rexx. (2015) *Explanation of ooRexx.* Retrieved November 30, 2023 from https://www.oorexx.org/about.html.

Rexx Language Association. (2023) *Homepage of the Rexx Language Association.* Retrieved Oktober 20, 2023 from https://www.rexxla.org/.

Azul.com (2023) *Downloading page for Zulu Java packages.* Retrieved Oktober 20, 2023 from https://www.azul.com/downloads/?package=jdk#zulu.

JetBrains. (2023) *Downloading page for IntelliJ IDEA IDE.* Retrieved Oktober 20, 2023 from https://www.jetbrains.com/idea/.

Sourceforge 1. (2021, July 20) *Downloading ooRexx plugin for IntelliJ IDEA.* Retrieved Oktober 20, 2023 from https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/GA/.

Sourceforge 2. (2023, June 6) *Downloading BSF4ooRexx850 beta.* Retrieved Oktober 20, 2023 from https://sourceforge.net/projects/bsf4oorexx/files/beta/20221004/.

Apache Software Foundation 3. (2023, November 2) *Downloading distributed binary version 5.2.3.* Retrieved Oktober 20, 2023 from https://archive.apache.org/dist/poi/release/bin/.

Apache Software Foundation 4. (2004, January) *Apache POI license page.* Retrieved December 2, 2023 from https://www.apache.org/licenses/LICENSE-2.0.

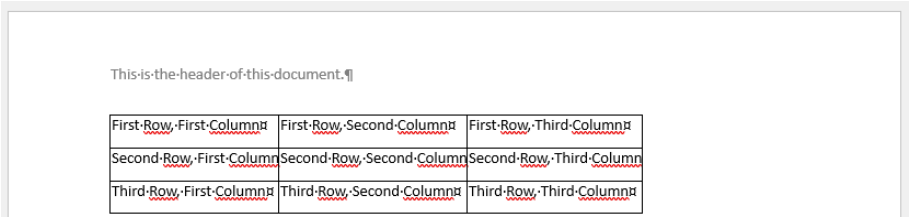
# Appendix

First Word Nutshell example output + code:



```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

document=.bsf~new("org.apache.poi.xwpf.usermodel.XWPFDocument")

.bsf~bsf.importClass("org.apache.poi.xwpf.usermodel.ParagraphAlignment", "AligningParagraph")

-- Titel Paragraph
titlePara=document~createParagraph()
titleRun=titlePara~createRun()
titleRun~setBold(1)
titleRun~setFontSize(30)
titleRun~setFontFamily("Bahnschrift")
titleRun~setText("Enter Your Title Here")

--Aligning Titelparagraph
titlePara~setSpacingAfter(500)
titlePara~setAlignment(.AligningParagraph~CENTER)

-- First Paragraph
p=document~createParagraph()
p~setAlignment(.AligningParagraph~RIGHT)
r=p~createRun()
r~setFontSize(12)
r~setFontFamily("Times New Roman")
r~setText("This is a standard Text!")

-- Next Page
p2=document~createParagraph()
p2~setWordWrapped(1)
p2~setPageBreak(1)
r2=p2~createRun()
r2~setFontSize(40)
r2~setItalic(1)
r2~setFontFamily("Arial Black")
r2~setText("New Page")

-- Saving the output document
parse source . . a
call directory filespec('L', a)
filename=directory()"/First_Nutshell_Example.docx"
filename=qualify(filename)
output=.bsf~new("java.io.FileOutputStream", filename)
document~write(output)

::REQUIRES BSF.CLS
```

Second Word Nutshell example output + code:



This·is·the·header·of·this·document.¶

| | | |
|---|---|---|
| First·Row,·First·Column¤ | First·Row,·Second·Column¤ | First·Row,·Third·Column¤ |
| Second·Row,·First·Column | Second·Row,·Second·Column | Second·Row,·Third·Column |
| Third·Row,·First·Column¤ | Third·Row,·Second·Column¤ | Third·Row,·Third·Column¤ |



This·is·the·footer·of·this·document.¶

```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

document=.bsf~new("org.apache.poi.xwpf.usermodel.XWPFDocument")
.bsf~bsf.loadClass("org.apache.poi.wp.usermodel.HeaderFooterType", "HeaderFooterType")

-- Creating Header
header=document~createHeader(.HeaderFooterType~DEFAULT)
headerParagraph=header~createParagraph()
-- Add text to the header
headerText="This is the header of this document."
run=headerParagraph~createRun()
run~setText(headerText)

-- Creating Footer
footer=document~createFooter(.HeaderFooterType~DEFAULT)
footerParagraph=footer~createParagraph()
-- Add text to the footer
footerText="This is the footer of this document."
run1=footerParagraph~createRun()
run1~setText(footerText)

-- Creating Table
table=document~createTable()
-- Addressing first row cells
row1=table~getRow(0)
row1~getCell(0)~setText("First Row, First Column")
row1~addNewTableCell()~setText("First Row, Second Column")
row1~addNewTableCell()~setText("First Row, Third Column")
row2=table~createRow()
row2~getCell(0)~setText("Second Row, First Column")
row2~getCell(1)~setText("Second Row, Second Column")
row2~getCell(2)~setText("Second Row, Third Column")
row3=table~createRow()
row3~getCell(0)~setText("Third Row, First Column")
row3~getCell(1)~setText("Third Row, Second Column")
row3~getCell(2)~setText("Third Row, Third Column")

-- Saving the output document
parse source . . b
call directory filespec('L', b)
filename=directory()"/Second_Nutshell_Example.docx"
filename=qualify(filename)
output=.bsf~new("java.io.FileOutputStream", filename)
document~write(output)
::REQUIRES BSF.CLS
```

Third Word Nutshell example output of additionally content + code:

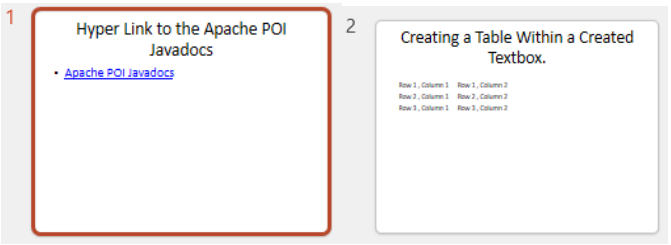

```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

parse source . . c
call directory filespec('L', c)
filename1=directory()"/First_Nutshell_Example.docx"
filename1=qualify(filename1)

olddocument=.bsf~new("java.io.FileInputStream", filename1)

-- Create a new XWPFDocument for the output
document=.bsf~new("org.apache.poi.xwpf.usermodel.XWPFDocument", olddocument)

-- Add a new paragraph with text
newParagraph=document~createParagraph()
newRun=newParagraph~createRun()
newRun~setText("It was added after accessing the previously created Word document.")

.bsf~bsf.loadClass("org.apache.poi.xwpf.usermodel.Document", "Document")
.bsf~bsf.importClass("org.apache.poi.util.Units", "Units")

-- Create a new paragraph for the image
imageParagraph=document~createParagraph()
imageRun=imageParagraph~createRun()

-- Specify the path to the image file
imagePath=directory()"/java.png"
imagePath=qualify(imagePath)
image=.bsf~new("java.io.FileInputStream", imagePath)

-- Add the image to the document
imageRun~addPicture(image, .Document~PICTURE_TYPE_PNG, "java.png", .Units~toEMU(250), .Units~toEMU(250))

-- Saving the output document
parse source . . d
call directory filespec('L', d)
filename=directory()"/Third_Nutshell_Example.docx"
filename=qualify(filename)
output = .bsf~new("java.io.FileOutputStream", filename)
document~write(output)

::REQUIRES BSF.CLS
```

First PowerPoint Nutshell example output + code:


```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

presentation=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow")

.bsf~bsf.importClass("org.apache.poi.xslf.usermodel.SlideLayout", "SlideLayout")

-- Creating slide layouts
template=presentation~getSlideMasters()~get(0) -- retrieves all master slides
layout=template~getLayout("title slide") -- Set Layout
layout1=template~getLayout("title only")
layout2=template~getLayout("title and content")

.bsf~bsf.importClass("org.apache.poi.xslf.usermodel.XSLFTextParagraph", "XSLFTextParagraph")

-- Creating title slide
titleslide=presentation~createSlide(layout)
titleTextbox=titleslide~getPlaceholder(0)
titleTextbox~setText("BSF4ooRexx/Java: Apache POI")

-- Editing text
titleParagraph=titleTextbox~getTextParagraphs()~get(0)
titleRun=titleParagraph~getTextRuns()~get(0)
titleRun~setFontFamily("Algerian")
titleRun~setFontSize(50.0)
titleRun~setBold(1)

subtitleTextbox=titleslide~getPlaceholder(1)~setText("First PowerPoint Nutshell Example")

firstslide=presentation~createSlide(layout1)

.bsf~bsf.importClass("java.awt.Rectangle", "Rectangle")

-- Creating custom textbox
textbox=firstslide~createTextBox()
textbox~setAnchor(.Rectangle~new(50, 150, 400, 200))
paragraph=textbox~addNewTextParagraph()
run=paragraph~addNewTextRun()
run~setText("Text in a custom created textbox.")
paragraph1=textbox~addNewTextParagraph
run1=paragraph1~addNewTextRun()
run1~setText("Second paragraph of the textbox.")

firstTextbox=firstslide~getPlaceholder(0)
firstTextbox~setText("This is a Slide With Just a Title and a Custom Textbox")

```

```
secondslide=presentation~createSlide(layout2)
secondTitleTextbox=secondslide~getPlaceholder(0)
secondTitleTextbox~setText("Here is a Slide With Bullet Points")

ContentTextbox=secondslide~getPlaceholder(1)
ContentTextbox~setText("First Text")

newParagraph=ContentTextbox~addNewTextParagraph()
leftMargin=100
newParagraph~setLeftMargin(leftMargin)
newRun=newParagraph~addNewTextRun()
newRun~setText("This is a new paragraph in the placeholder.")

-- Save the File
parse source . . e
call directory filespec('L', e)
filename=directory()"/First_Nutshell_Example.pptx"
filename=qualify(filename)
output=.bsf~new("java.io.FileOutputStream", filename)
presentation~write(output)

::REQUIRES BSF.CLS
```

Second PowerPoint Nutshell example output + code:



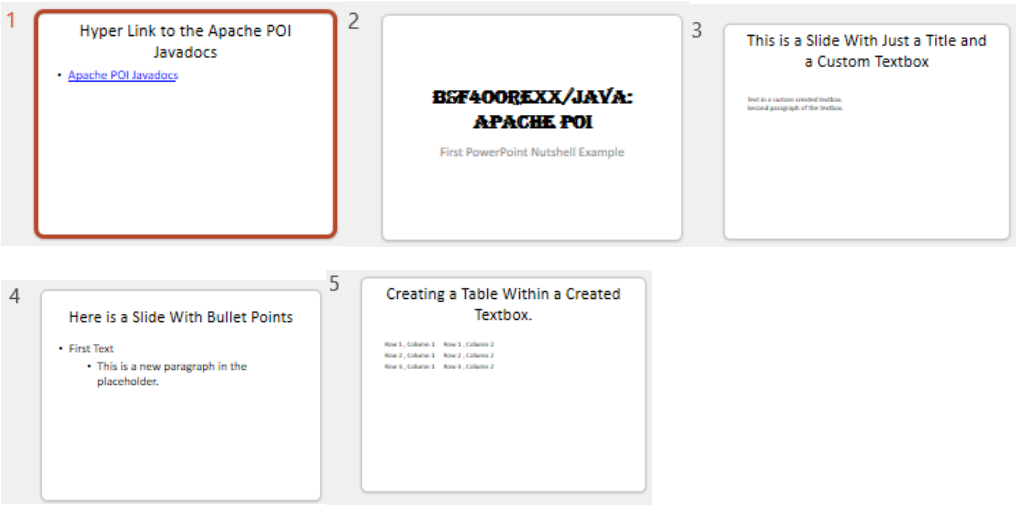

```
/*Copyright [2023] [Florian Frcena]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.*/

presentation=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow")

.bsf~bsf.importClass("org.apache.poi.xslf.usermodel.SlideLayout", "SlideLayout")
.bsf~bsf.importClass("org.apache.poi.xslf.usermodel.XSLFSlide", "XSLFSlide")

template=presentation~getSlideMasters()~get(0) -- retrieves all master slides
layout1=template~getLayout("title and content")
layout2=template~getLayout("title only")

-- Hyperlink at Slide 1
firstslide=presentation~createSlide(layout1)
Textbox=firstslide~getPlaceholder(0)
Textbox~setText("Hyper Link to the Apache POI Javadocs")

Textbox1=firstslide~getPlaceholder(1)
Textbox1~clearText()
textRun=Textbox1~addNewTextParagraph()~addNewTextRun()
textRun~setText("Apache POI Javadocs")
link=textRun~createHyperlink()
link~setAddress("https://poi.apache.org/apidocs/5.0/")

secondslide=presentation~createSlide(layout2)
Textbox=secondslide~getPlaceholder(0)
Textbox~setText("Creating a Table Within a Created Textbox.")

.bsf~bsf.importClass("java.awt.Rectangle", "Rectangle")

-- Insert a table on slide1
table=secondslide~createTable()
table~setAnchor(.Rectangle~new(50, 150, 500, 200))

-- Add rows and cells to the table
rowCount=3
colCount=2
do row=0 to rowCount -1
    tableRow=table~addRow()
    do col=0 to colCount -1
        tableCell=tableRow~addCell()
        tableCell~setText("Row" row+1 ", Column" col+1)
        table~setColumnWidth(col, 150)
    end
end

```

```
58  -- Save the File
59  parse source . . f
60  call directory filespec('L', f)
61
62  filename=directory()"\Second_Nutshell_Example.pptx"
63  filename=qualify(filename)
64
65  output=.bsf~new("java.io.FileOutputStream", filename)
66  presentation~write(output)
67
68  ::REQUIRES  BSF.CLS
```

Third PowerPoint Nutshell example code + output:

```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

parse source . . g
call directory filespec('L', g)

firstfilename=directory()"/First_Nutshell_Example.pptx"
firstfilename=qualify(firstfilename)
firstoldpresentation=.bsf~new("java.io.FileInputStream", firstfilename)

secondfilename=directory()"/Second_Nutshell_Example.pptx"
secondfilename=qualify(secondfilename)
secondoldpresentation=.bsf~new("java.io.FileInputStream", secondfilename)

-- Create an ArrayList to hold the slides
slidesList=.bsf~new("java.util.ArrayList")

-- Create XMLSlideShow instances for each input file
xmlSlideShow1=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow", firstoldpresentation)
xmlSlideShow2=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow", secondoldpresentation)

-- Add slides from the first presentation to the list
slidesList~addAll(xmlSlideShow1~getSlides())

-- Add slides from the second presentation to the list
slidesList~addAll(xmlSlideShow2~getSlides())

-- Create a new XMLSlideShow instance
presentation=.bsf~new("org.apache.poi.xslf.usermodel.XMLSlideShow")

-- Retrieve slide master and layouts
template=presentation~getSlideMasters()~get(0)
layout=template~getLayout("title slide")
layout1=template~getLayout("title only")
layout2=template~getLayout("title and content")

```

```
-- Add the slides from the list to the new presentation with specific layouts
do slide over slidesList
    select
        when slide==slidesList~get(0) then newSlide=presentation~createSlide(layout)
        when slide==slidesList~get(1) then newSlide=presentation~createSlide(layout1)
        when slide==slidesList~get(3) then newSlide=presentation~createSlide(layout1)
        otherwise newSlide=presentation~createSlide(layout2)
    end
    -- Import content from the original slide
    newSlide~importContent(slide)
end

-- Changing slide order of the presentation
slides=presentation~getSlides()
selectedslide=slides~get(3)
presentation~setSlideOrder(selectedslide, 0)

-- Save the reordered file
parse source . . h
call directory filespec('L', h)

filename=directory()"/Third_Nutshell_Example.pptx"
filename=qualify(filename)

output=.bsf~new("java.io.FileOutputStream", filename)
presentation~write(output)

::REQUIRES BSF.CLS
```



First Excel Nutshell example code + output:

```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

workbook=.bsf~new("org.apache.poi.xssf.usermodel.XSSFWorkbook")
spreadsheet=.bsf~new("org.apache.poi.xssf.usermodel.XSSFSheet")
-- Create a new sheet
spreadsheet=workbook~createSheet("Cell Types")
-- Creating a new font
font=workbook~createFont()
font~setFontHeightInPoints(20)
font~setFontName("GOUDY STOUT")
font~setItalic(.true)
-- Setting font color to blue
colour=workbook~getCreationHelper()~createExtendedColor()
colour~setARGBHex("0000B0")
font~setColor(colour)
-- Setting font into style
style=workbook~createCellStyle()
style~setFont(font)

-- Create headers
row=spreadsheet~createRow(0)
cell=row~createCell(0)
cell~setCellValue("Type of Cell")
cell~setCellStyle(style)
font1=workbook~createFont()
font1~setFontHeightInPoints(20)
font1~setFontName("BAHNSCHRIFT")
font1~setItalic(.true)
style1=workbook~createCellStyle()
style1~setFont(font1)
cell=row~createCell(1)
cell~setCellValue("Cell value")
cell~setCellStyle(style)

-- Create rows with different cell types
row=spreadsheet~createRow(1)
cell=row~createCell(0)
cell~setCellValue("Set cell value BLANK")
cell~setCellStyle(style1)
row~createCell(1)
row = spreadsheet~createRow(2)
cell=row~createCell(0)
cell~setCellValue("Set cell value BOOLEAN")
cell~setCellStyle(style1)
row~createCell(1)~setCellValue(.false)
```

```
row~createCell(2)~setCellValue(.true)
row = spreadsheet~createRow(3)
cell=row~createCell(0)
cell~setCellValue("Set cell value DATE")
cell~setCellStyle(style1)
cell = row~createCell(1)
cell~setCellValue(Date())
style2=workbook~createCellStyle
helper=workbook~getCreationHelper()
style2~setDataFormat(helper~createDataFormat()~getFormat("m/d/yy"))
cell~setCellStyle(style2)
row = spreadsheet~createRow(4)
cell=row~createCell(0)
cell~setCellValue("Set cell value NUMERIC")
cell~setCellStyle(style1)

-- Create a cell style for numeric
style3 = workbook~createCellStyle()
style3~setDataFormat(helper~createDataFormat()~getFormat("0"))
-- Set cell type numeric
cell = row~createCell(1)
cell~setCellValue(123.45)
cell~setCellStyle(style3)
row = spreadsheet~createRow(5)
cell=row~createCell(0)
cell~setCellValue("Set cell value STRING")
cell~setCellStyle(style1)
cell = row~createCell(1)
cell~setCellValue("This is a string.")
row = spreadsheet~createRow(6)
cell=row~createCell(0)
cell~setCellValue("Set cell value CURRENCY")
cell~setCellStyle(style1)
cell = row~createCell(1)
-- Set a numeric value for currency
cell~setCellValue(895.68)
-- Create a cell style for currency format
currencyStyle = workbook~createCellStyle()
currencyStyle~setDataFormat(helper~createDataFormat()~getFormat("[$$-409]#,##0.00"))

-- Apply the currency format
cell~setCellStyle(currencyStyle)
-- Automatically size columns
spreadsheet~autoSizeColumn(0)
spreadsheet~autoSizeColumn(1)
spreadsheet~setColumnWidth(2, 2000)
--Save the file
parse source . . i
call directory filespec('L', i)
filename=directory()"/First_Nutshell_Example.xlsx"
filename=qualify(filename)
output=.bsf~new("java.io.FileOutputStream" , filename)
workbook~write(output)

::REQUIRES  BSF.CLS
```

| | A | B | C |
|---|---|---|---|
| 1 | TYPE OF CELL | CELL VALUE | |
| 2 | Set cell value BLANK | | |
| 3 | Set cell value BOOLEAN | 0 | 1 |
| 4 | Set cell value DATE | 16 Dec 2023 | |
| 5 | Set cell value NUMERIC | 123.45 | |
| 6 | Set cell value STRING | This is a string. | |
| 7 | Set cell value CURRENCY | 895.68 | |

Cell Types

Second Excel Nutshell example code + output:

```
/*Copyright [2023] [Florian Frcena]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.*/

workbook=.bsf.new("org.apache.poi.xssf.usermodel.XSSFWorkbook")

spreadsheet=workbook.createSheet("Text direction")
spreadsheet1=workbook.createSheet("Test")

-- 90 degrees angle
myStyle=workbook.createCellStyle()
myStyle.setRotation(90)
row=spreadsheet.createRow(0)
cell=row.createCell(3)
cell.setCellValue("90 degrees Angle")
cell.setCellStyle(myStyle)

-- 120 degrees angle
myStyle=workbook.createCellStyle()
myStyle.setRotation(180)
cell=row.createCell(5)
cell.setCellValue("180 degrees Angle")
cell.setCellStyle(myStyle)

-- 270 degrees
myStyle=workbook.createCellStyle()
myStyle.setRotation(270)
cell=row.createCell(7)
cell.setCellValue("270 degrees Angle")
cell.setCellStyle(myStyle)

-- Set a hyperlink formula to an URL
hyperlinkFormula='HYPERLINK("https://poi.apache.org/apidocs/5.0/")'
rowHyperlink=spreadsheet.createRow(2)
cell=rowHyperlink.createCell(0).setCellValue("Hyperlink to Apache POI Javadocs")
cellHyperlink=rowHyperlink.createCell(1)
cellHyperlink.setCellFormula(hyperlinkFormula)

-- Set a hyperlink formula to file Second_Nutshell_Example.xlsx
filelinkFormula='HYPERLINK("First_Nutshell_Example.xlsx")'
rowFilelink=spreadsheet.createRow(3)
cell=rowFilelink.createCell(0).setCellValue('Hyperlink Excel File First_Nutshell_Example.xlsx")
cellFilelink=rowFilelink.createCell(1)
cellFilelink.setCellFormula(filelinkFormula)
```

```
-- Set a hyperlink formula in cell B3 to the "test" sheet in the same workbook
spreadsheetlinkFormula='HYPERLINK("#test!A1")'
rowSpreadsheetlink=spreadsheet~createRow(4)
cell=rowSpreadsheetlink~createCell(0)~setCellValue("Hyperlink to the Test Spreadsheet")
cellSpreadsheetlink=rowSpreadsheetlink~createCell(1)
cellSpreadsheetlink~setCellFormula(spreadsheetlinkFormula)

spreadsheet~autoSizeColumn(0)
spreadsheet~setColumnWidth(1, 8600)

-- Merging Cells
spreadsheet~addMergedRegion(.bsf~new("org.apache.poi.ss.util.CellRangeAddress", 0, 0, 7, 8))
-- Set print area with indexes
workbook~setPrintArea(0, 0, 10, 0, 5)
-- Set paper size
spreadsheet~getPrintSetup()~setPaperSize(9)
-- Set print grid lines
spreadsheet~setPrintGridlines(.true)

--Save the file
parse source . . j
call directory filespec('L', j)
filename=directory()"/Second_Nutshell_Example.xlsx"
filename=qualify(filename)
output=.bsf~new("java.io.FileOutputStream" , filename)
workbook~write(output)

::REQUIRES  BSF.CLS
```

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | 90 degrees Angle | | 180 degrees Angle | | 270 degrees Angle | | | |
| 2 | | | | | | | | | | | |
| 3 | Hyperlink to Apache POI Javadocs | https://poi.apache.org/apidocs/5.0/ | | | | | | | | | |
| 4 | Hyperlink Excel File First_Nutshell_Example.xlsx | First_Nutshell_Example.xlsx | | | | | | | | | |
| 5 | Hyperlink to the Test Spreadsheet | #test!A1 | | | | | | | | | |
| 6 | | | | | | | | | | | |

Text direction | Test | +

Third Excel Nutshell example + output of first 15 generated numbers:

```
/*Copyright [2023] [Florian Frcena]

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.*/

workbook=.bsf~new("org.apache.poi.xssf.usermodel.XSSFWorkbook")

spreadsheet=workbook~createSheet("Random Numbers and Excel Functions")

row=spreadsheet~createRow(0)
cell=row~createCell(0)~setCellValue("Random Numbers")
cell=row~createCell(1)~setCellValue("Sum")
cell=row~createCell(2)~setCellValue("Count")
cell=row~createCell(3)~setCellValue("Count if value >=500")
cell=row~createCell(4)~setCellValue("Minimum")
cell=row~createCell(5)~setCellValue("Maximum")
cell=row~createCell(6)~setCellValue("Average")
cell=row~createCell(7)~setCellValue("Median")
cell=row~createCell(8)~setCellValue("Standard Deviation")

-- Creating a formula to generate random numbers in column A
do i=3 to 102
    row=spreadsheet~createRow(i - 1)
    cell=row~createCell(0)
    cell~setCellFormula("Randbetween(0, 1000)")
end

-- Summing up the values in column A
row1=spreadsheet~createRow(1)
formulaCellSum=row1~createCell(1)
formulaCellSum~setCellFormula("Sum(A3:A102)")

-- Creating formula for counting non-empty cells in column A
formulaCellCount=row1~createCell(2)
formulaCellCount~setCellFormula("Count(A3:A102)")

-- Creating formula for counting values in column A at least 500
formulaCellCountAtLeast500=row1~createCell(3)
formulaCellCountAtLeast500~setCellFormula("Countif(A3:A102, ">=500")")

-- Creating formula for the minimum value in column A
formulaCellMin=row1~createCell(4)
formulaCellMin~setCellFormula("Min(A3:A102)")

-- Creating formula for the maximum value in column A
formulaCellMax=row1~createCell(5)
formulaCellMax~setCellFormula("Max(A3:A102)")

```

```
-- Creating formula for the average value in column A
formulaCellAvg=row1~createCell(6)
formulaCellAvg~setCellFormula("Average(A3:A102)")

-- Creating formula for the mean (median) value in column A
formulaCellMedian=row1~createCell(7)
formulaCellMedian~setCellFormula("Median(A3:A102)")

-- Creating formula for calculating the standard deviation of column A
formulaCellStdDev=row1~createCell(8)
formulaCellStdDev~setCellFormula("Stdev(A3:A102)")

spreadsheet~autoSizeColumn(0)
spreadsheet~setColumnWidth(1, 1700)
spreadsheet~autoSizeColumn(2)
spreadsheet~autoSizeColumn(3)
spreadsheet~autoSizeColumn(4)
spreadsheet~autoSizeColumn(5)
spreadsheet~autoSizeColumn(6)
spreadsheet~autoSizeColumn(7)
spreadsheet~autoSizeColumn(8)

/* Save the file */
parse source . . k
call directory filespec('L', k)

filename=directory()"/Third_Nutshell_Example.xlsx"
filename=qualify(filename)

output=.bsf~new("java.io.FileOutputStream" , filename)
workbook~write(output)
output~close()

::REQUIRES  BSF.CLS
```

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Random Numbers | Sum | Count | Count if value >=500 | Minimum | Maximum | Average | Median | Standard Deviation |
| 2 | | 52649 | 100 | 54 | 2 | 980 | 526,49 | 571,5 | 296,4646059 |
| 3 | 606 | | | | | | | | |
| 4 | 298 | | | | | | | | |
| 5 | 530 | | | | | | | | |
| 6 | 786 | | | | | | | | |
| 7 | 422 | | | | | | | | |
| 8 | 939 | | | | | | | | |
| 9 | 783 | | | | | | | | |
| 10 | 921 | | | | | | | | |
| 11 | 798 | | | | | | | | |
| 12 | 52 | | | | | | | | |
| 13 | 18 | | | | | | | | |
| 14 | 968 | | | | | | | | |
| 15 | 292 | | | | | | | | |
| 16 | 40 | | | | | | | | |

Random Numbers and Excel Functi

Apache POI 5.2.3 .jar files

poi-5.2.3.jar
poi-examples-5.2.3.jar
poi-excelant-5.2.3.jar
poi-javadoc-5.2.3.jar
poi-ooxml-5.2.3.jar
poi-ooxml-full-5.2.3.jar
poi-ooxml-lite-5.2.3.jar
poi-scratchpad-5.2.3.jar
commons-codec-1.15.jar
commons-collections4-4.4.jar
commons-io-2.1.1.0.jar
commons-math3-3.6.1.jar
commons-compress-1.21.jar
commons-logging-1.2.jar
curvesapi-1.07.jar
log4j-api-2.18.0.jar
SparseBitSet-1.2.jar
jakarta.activation-2.0.1.jar
jakarta.xml.bind-api-3.0.1.jar
slf4j-api-1.7.36.jar
xmlbeans-5.1.1.jar