An Introduction to Apache PDFBox Library

Bachelor's thesis at the Institute for Information Systems and Society

Tiao Wang, h0553521

Agenda

- Introduction
- The Portable Document Format
- Apache PDFBox
- Examples
- Q&A

Introduction

- A beginner's guide for the Apache PDFBox Library
- Aim of this project is to provide basic knowledge about PDF and Apache PDFBox
- A set of nutshell examples demonstrates how certain tasks can be achieved
- Writen in ooRexx language
- BSF4ooRexx is used to bridge the Java functionality to ooRexx

The Portable Document Format

- Document sharing and distribution
- Online forms and surveys
- E-books and digital publications
- Invoices and receipts
- Legal and official documents

The Portable Document Format
A brief Histroy

- In the early 1990s, Adobe Systems created PDF as a way to share and view electronic documents across different platforms and operating systems.
- The initial goal was to create a file format that would preserve the look and layout of a document, regardless of the software or hardware used to view it.
- The first version of PDF was released in 1993, and it quickly gained popularity among businesses and government agencies.
- Over the years, PDF has undergone several revisions and updates to improve its functionality and usability.
- Today, PDF is one of the most widely used file formats for electronic documents, and it is supported by a variety of software applications and platforms.

The Portable Document Format
Version History I

- PDF 1.0 (1993): The first version of PDF allowed users to create documents that could be viewed on any device or operating system.
- PDF 1.1 (1996): This version introduced several new features, including support for encryption, bookmarks, and metadata.
- PDF 1.2 (1998): PDF 1.2 added support for multimedia content, such as audio and video, as well as new font and color options.
- PDF 1.3 (2000): This version added support for transparent images, layers, and digital signatures.
- PDF 1.4 (2001): PDF 1.4 introduced support for 3D graphics and metadata, as well as improvements to accessibility and printing.

The Portable Document Format
Version History II

- PDF 1.5 (2003): This version added support for advanced transparency, improved tagging for accessibility, and enhanced support for annotations.
- PDF 1.6 (2004): PDF 1.6 introduced support for JPEG 2000 compression, enhanced security features, and improved support for PDF forms.
- PDF 1.7 (2006): The final version of PDF released by Adobe, PDF 1.7 introduced support for layers, improved tagging for accessibility, and enhancements to digital signatures.
- In 2008, Adobe released a new format called PDF/A, which is a version of PDF specifically designed for archiving purposes. PDF/A ensures that the document will look the same years from now as it does today, even as technology and software evolve.
- In 2017, Adobe released PDF 2.0, which is the latest version of PDF. PDF 2.0 includes improved support for transparency, digital signatures, and the ability to embed HTML and other web content in PDF documents.

The Portable Document Format
Benefits and Advantages

- Preserves formatting
- Security features
- Smaller file size
- Searchability
- Cross-platform compatibility
- Easy to create, edit, and share.

Apache PDFBox

- An open source Java library
- For working with PDF documents
- Provides a wide range of functionality
- Published under the Apache License v2.0

Apache PDFBox
A brief Histroy

- 2002: PDFBox was first released by Ben Litchfield with the aim of extracting text from PDF files.
- 2008: The project was moved to the Apache Software Foundation, where it became part of the Apache PDFBox project.
- 2010: PDFBox version 1.0 was released with improved functionality for working with PDF documents.
- 2012: PDFBox version 1.7 was released with support for creating PDF/A documents.
- 2015: PDFBox version 2.0 was released with a major overhaul of the codebase and significant performance improvements.
- 2021: PDFBox version 3.0 was released with improved performance, Java 11 compatibility, and better digital signature support

Apache PDFBox
Components

- PDFBox: the main part
- FontBox: handles font information
- XmpBox: handles XMP metadata
- Preflight: checks PDF files for PDF/A-1b conformity.

Apache PDFBox
Features I

- PDF Creation and Manipulation: PDFBox can create new PDF documents, manipulate existing PDF documents, and extract information from PDF documents.
- Text Extraction: PDFBox can extract text from PDF documents, including Unicode text and embedded fonts.
- Image Extraction: PDFBox can extract images from PDF documents, including JPEG, PNG, and TIFF images.
- Encryption and Decryption: PDFBox can encrypt and decrypt PDF documents using various encryption algorithms.
- Digital Signatures: PDFBox can create and verify digital signatures in PDF documents.

Apache PDFBox
Features II

- Form Filling: PDFBox can fill in interactive form fields in PDF documents.
- Metadata Manipulation: PDFBox can read, write, and modify metadata in PDF documents.
- Text Search: PDFBox can search for text in PDF documents.
- Text Highlighting: PDFBox can highlight text in PDF documents.
- Font Embedding: PDFBox can embed fonts in PDF documents.
- Watermarking: PDFBox can add watermarks to PDF documents.
- Conversion: PDFBox can convert PDF documents to other formats, such as images and HTML.

Apache PDFBox
Benefits and Advantages

- Open-source and free to use
- Cross-platform compatibility
- Comprehensive functionality
- High performance
- Support for d encryption and digital signatures
- Active community of developers and users

Examples
Required Software:

- Java
- ooRexx
- BSF4ooRexx
- Apache PDFBox components
  - pdfbox
  - xmpbox
  - preflight

Examples
Part I

- Creating a PDF Document
- Adding Text to an Existing PDF Document
- Use Different Fonts and Colors
- Extracting Text from an Existing PDF Document
- Drawing some Shapes
- Creating a Table with Content

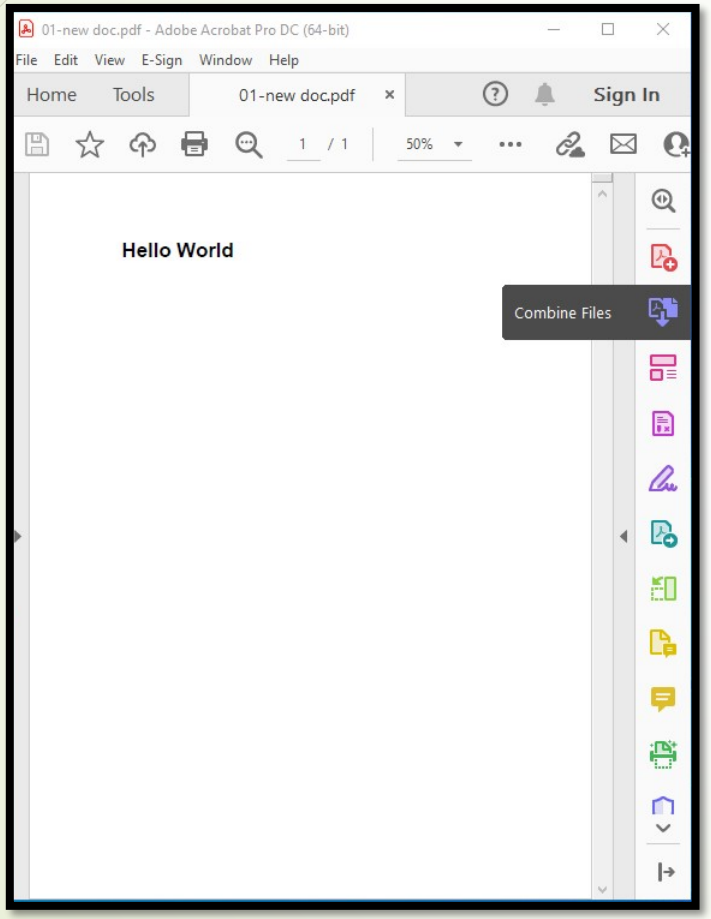
Examples
Part II

- Converting a PDF Document To Image
- Inserting Image to a PDF Document
- Adding Multiple Pages to a PDF Document
- Splitting a PDF Document with Multiple Pages
- Merging Multiple PDF Documents
- Setting the Document Metadata

Examples
Part III

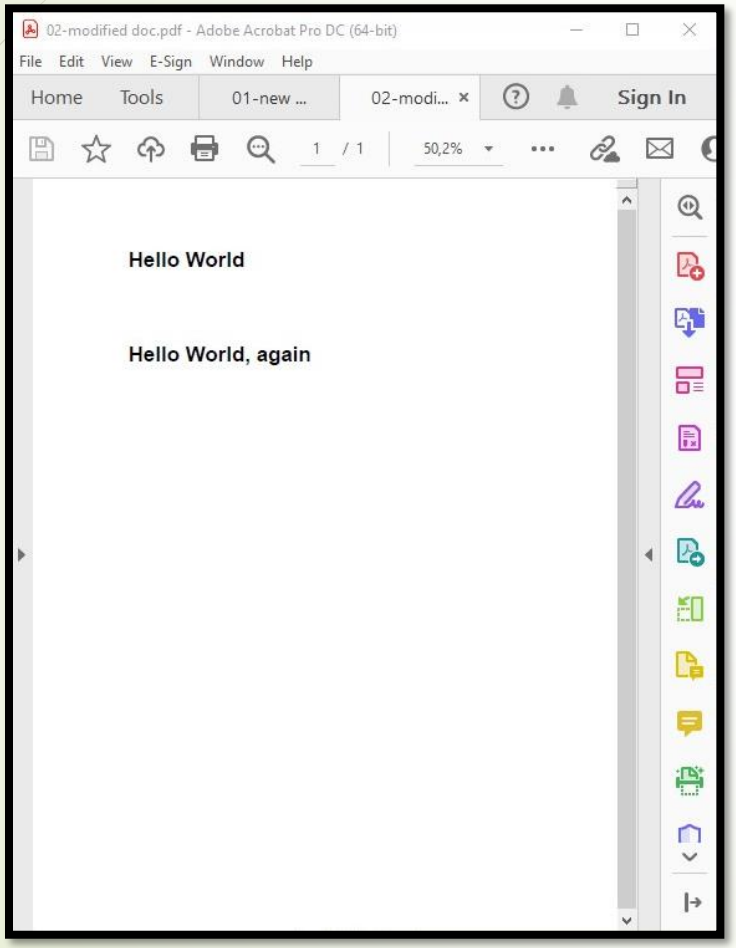
- Adding Watermark to a Document
- Encrypting a PDF Document
- Creating a PDF-A Document
- Validating a PDF-A Document
- Creating a Digital Signature
- Verifying a Digital Signature

“01. Creating a PDF Document.rex”




```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)

-- define font type

fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font =.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert content
cont~beginText
cont~setFont(font, 22)
cont~setLeading(25f)
cont~newLineAtOffset(100, 700)
cont~showText("Hello World")
cont~endText
cont~close

-- save and close the document file
doc~save("01-new doc.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

“02. Adding Text to an Existing PDF Document.rex”

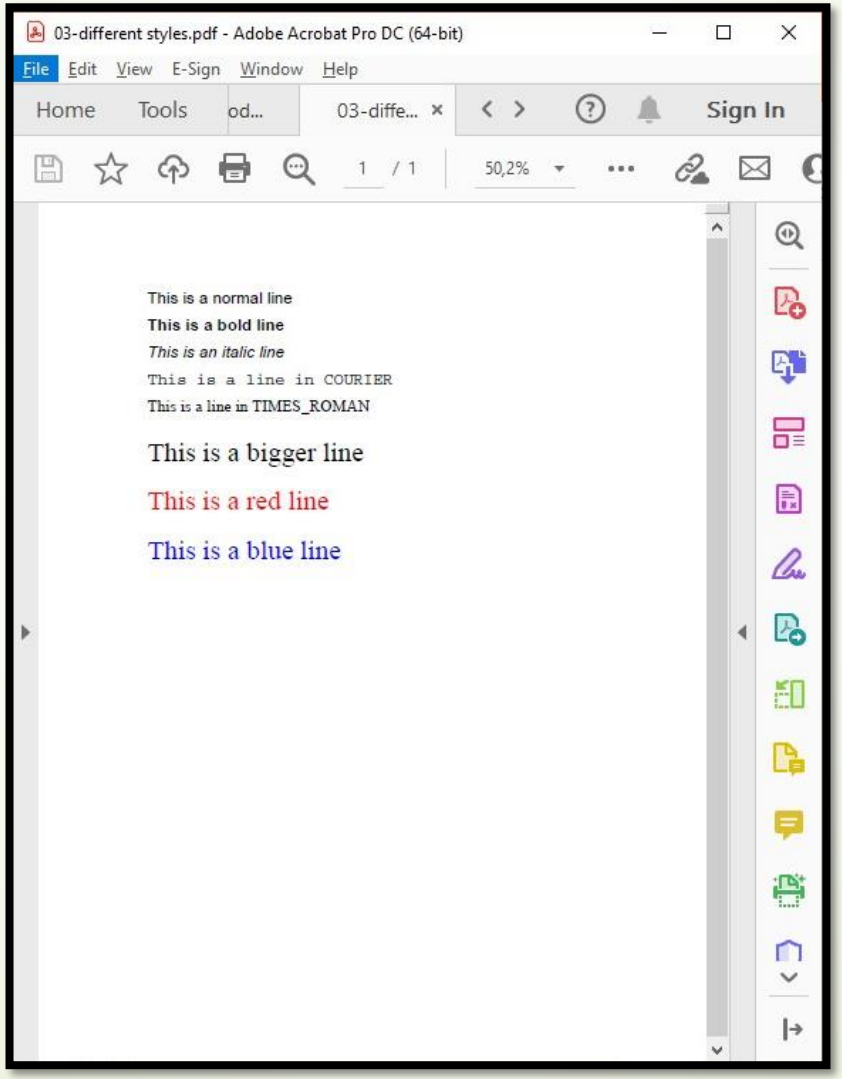



```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "01-new doc.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)
page=doc~getPage(0)

-- set the append mode
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
append=BSF.loadClass(contclass)~AppendMode~APPEND

-- create a content stream
cont=.bsf~new(contclass,doc,page,append,"true")
fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert content
cont~beginText
cont~setFont(font, 22)
cont~setLeading(25f)
cont~newLineAtOffset( 100, 600 )
cont~showText("Hello World, again")
cont~endText
cont~close

-- save and close the document file
doc~save("02-modified doc.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

“03. Use Different Fonts and Colors.rex”

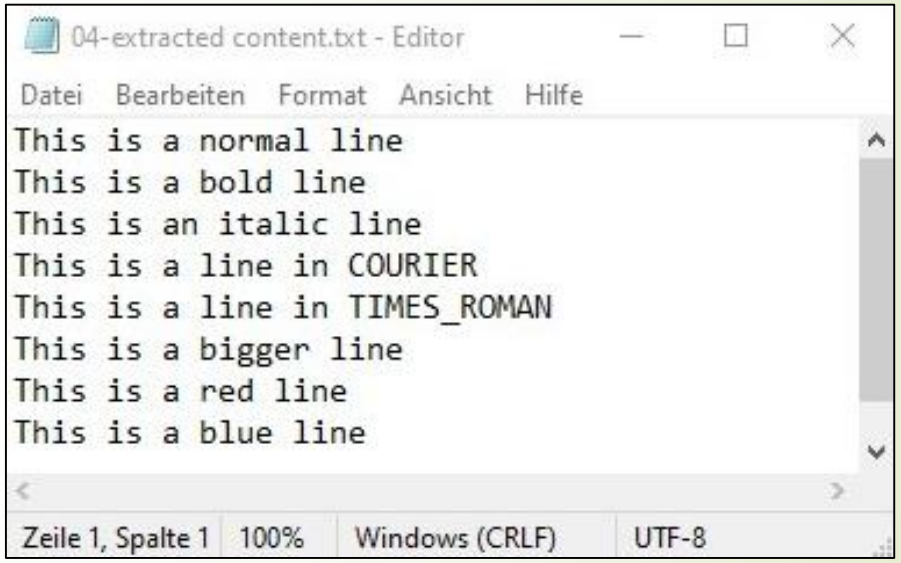
```
-- change directory to program location

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

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)

-- define font type
fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert content
cont~beginText
cont~setFont(font, 15)
cont~setLeading(25f)
cont~newLineAtOffset( 100, 700 )
cont~showText("This is a normal line")
cont~newLine

-- use a new font type
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)
cont~setFont(font, 15)
cont~showText("This is a bold line")
cont~newLine

-- use a new font type
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_OBLIQUE
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)
cont~setFont(font, 15)
cont~showText("This is an italic line")
cont~newLine

-- use a new font type
fname = BSF.loadClass(fontclass)~FontName~COURIER
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)
cont~setFont(font, 15)
cont~showText("This is a line in COURIER")
cont~newLine

-- use a new font type
fname = BSF.loadClass(fontclass)~FontName~TIMES_ROMAN
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)
cont~setFont(font, 15)
cont~showText("This is a line in TIMES_ROMAN")
cont~setLeading(45f)
cont~newLine

-- use a bigger font size
cont~setFont(font, 25)
cont~showText("This is a bigger line")
cont~newLine

-- use a new font color
col=BSF.loadClass("java.awt.Color")~red
cont~setNonStrokingColor(col)
cont~showText("This is a red line")
cont~newLine

-- use a new font color
col=BSF.loadClass("java.awt.Color")~blue
cont~setNonStrokingColor(col)
cont~showText("This is a blue line")
cont~endText
cont~close

-- save and close the document file
doc~save("03-different styles.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

“03. Use Different Fonts and Colors.rex”



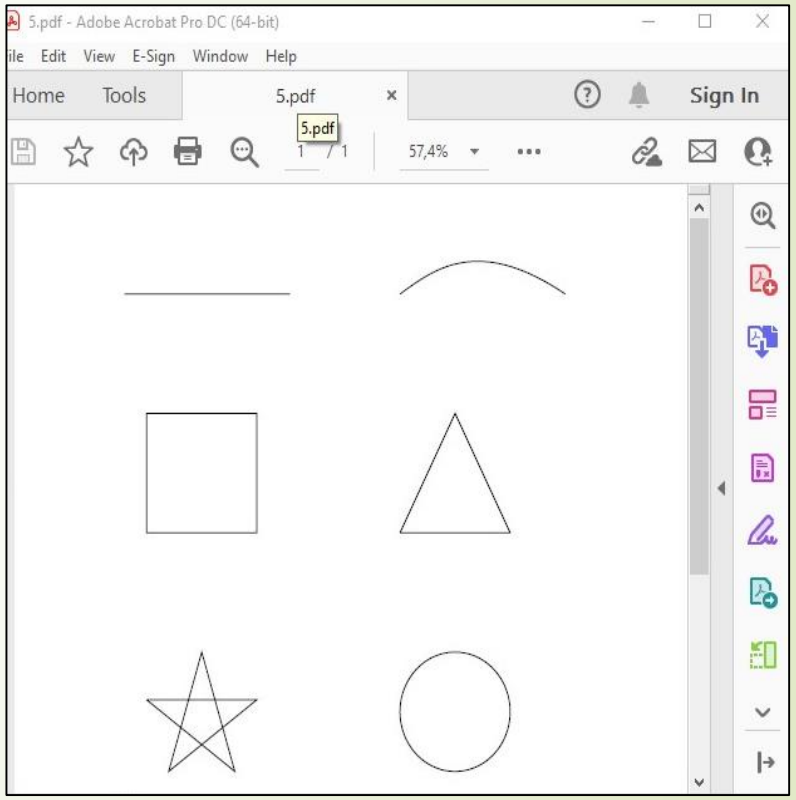
“04. Extracting Text from an Existing PDF Document.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "03-different styles.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)

-- use the stripper to get content
stripper=.bsf~new("org.apache.pdfbox.text.PDFTextStripper")
content=stripper~getText(doc)

-- write the content to the destination file
writer=bsf.import("java.io.FileWriter")
file= writer~new("04-extracted content.txt")
file~write(content)
file~close()

-- get java support
::requires "BSF.CLS"
```

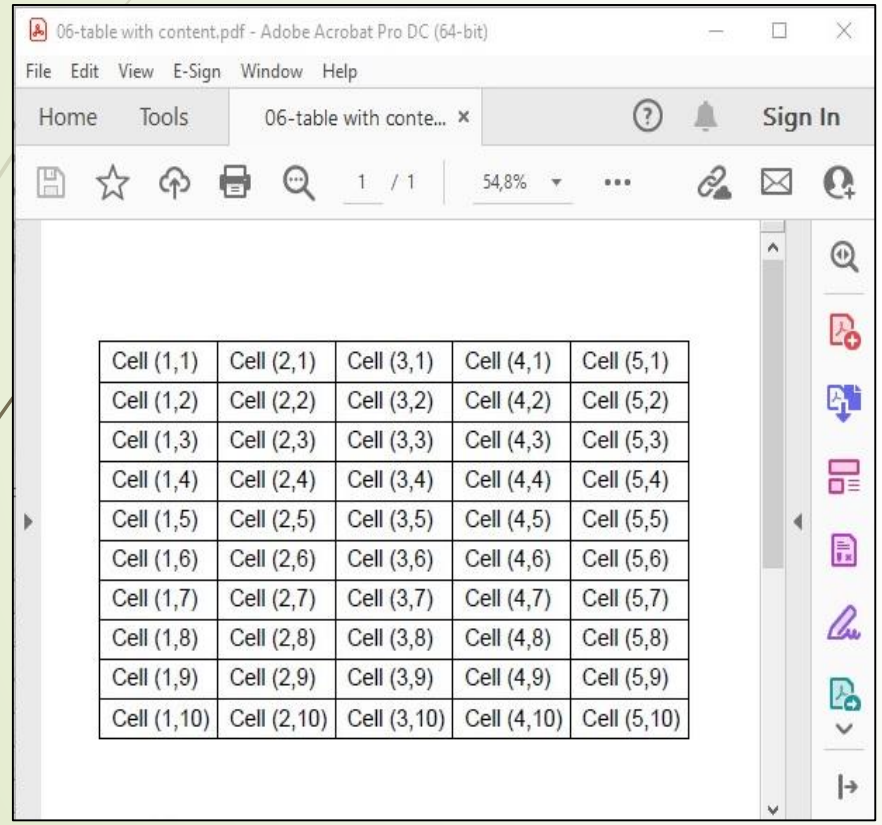


“05. Drawing some Shapes.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- create a content stream
contclass =
"org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)
cont~setLineWidth(1)

-- draw a line
cont~moveTo(100,700)
cont~lineTo(250,700)

-- draw a curve
cont~moveTo(350,700)
cont~curveTo(380,720,420,750,500,700)

-- draw a rectangle
cont~addRect(120,600,100,-100)

-- draw a triangle
cont~moveTo(350,500)
cont~lineTo(450,500)
cont~lineTo(400,600)
cont~lineTo(350,500)

-- draw a star shape
cont~moveTo(120,360)
cont~lineTo(220,360)
cont~lineTo(140,300)
cont~lineTo(170,400)
cont~lineTo(200,300)
cont~lineTo(120,360)

-- draw a circle
cx=400
cy=350
r=50
k=0.552284749831
cont~moveTo(cx-r,cy)
cont~curveTo(cx-r,cy+k*r,cx-k*r,cy+r,cx,cy+r)
cont~curveTo(cx+k*r,cy+r,cx+r,cy+k*r,cx+r,cy)
cont~curveTo(cx+r,cy-k*r,cx+k*r,cy-r,cx,cy-r)
cont~curveTo(cx-k*r,cy-r,cx-r,cy-k*r,cx-r,cy)

-- make all the drawed shapes visible
cont~stroke
cont~close

-- save and close the document file
doc~save("05-shapes.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```



“06. Creating a Table with Content.rex”

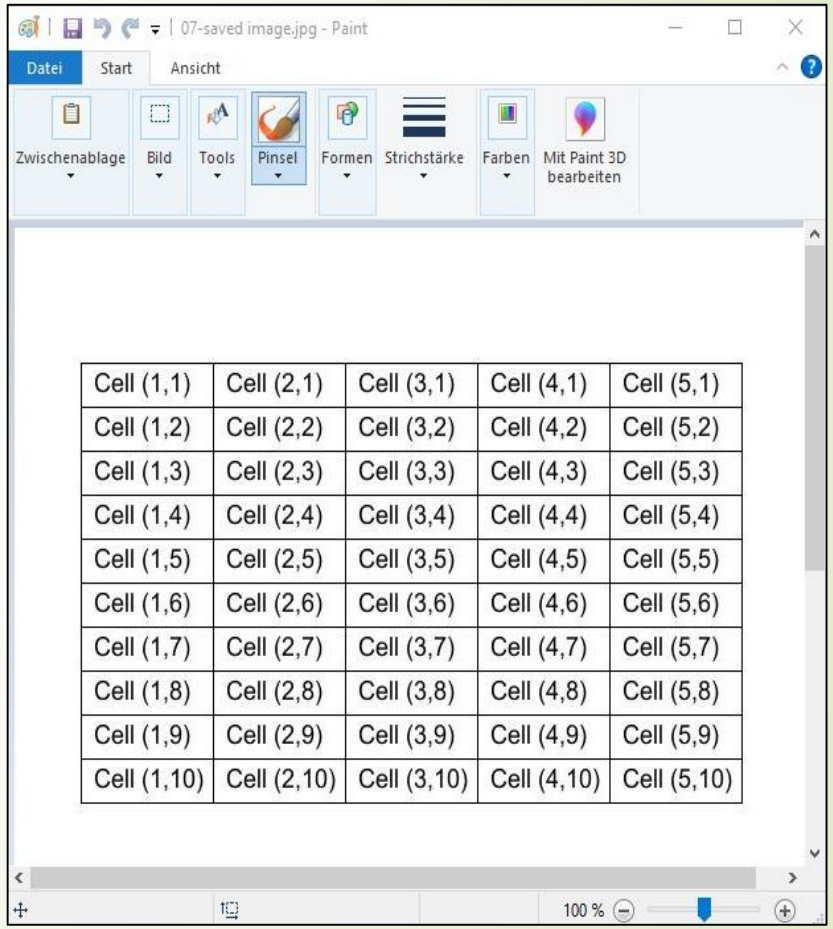


Cell (1,1)	Cell (2,1)	Cell (3,1)	Cell (4,1)	Cell (5,1)
Cell (1,2)	Cell (2,2)	Cell (3,2)	Cell (4,2)	Cell (5,2)
Cell (1,3)	Cell (2,3)	Cell (3,3)	Cell (4,3)	Cell (5,3)
Cell (1,4)	Cell (2,4)	Cell (3,4)	Cell (4,4)	Cell (5,4)
Cell (1,5)	Cell (2,5)	Cell (3,5)	Cell (4,5)	Cell (5,5)
Cell (1,6)	Cell (2,6)	Cell (3,6)	Cell (4,6)	Cell (5,6)
Cell (1,7)	Cell (2,7)	Cell (3,7)	Cell (4,7)	Cell (5,7)
Cell (1,8)	Cell (2,8)	Cell (3,8)	Cell (4,8)	Cell (5,8)
Cell (1,9)	Cell (2,9)	Cell (3,9)	Cell (4,9)	Cell (5,9)
Cell (1,10)	Cell (2,10)	Cell (3,10)	Cell (4,10)	Cell (5,10)

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- define font type
fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)
cont~setLineWidth(1)
cont~setFont(font, 18)

-- define the cell and table
initx=50
inity=700
cheight=30
cwidth=100
col=5
row= 10

-- start the loops to draw the table and insert contents
do i=0 to row-1
do j=0 to col-1
cont~addRect(initx+j*cwidth,inity-i*cheight,cwidth,-cheight)
cont~beginText
cont~newLineAtOffset(initx+j*cwidth+10,inity-i*cheight-cheight+10)
cont~showText("Cell" "(" || j+1 || "," || i+1 || ")")
cont~endText
end
end

-- make the table visible
cont~stroke
cont~close

-- save and close the document file
doc~save("06-table with content.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

“07. Converting a PDF Document To Image.rex”

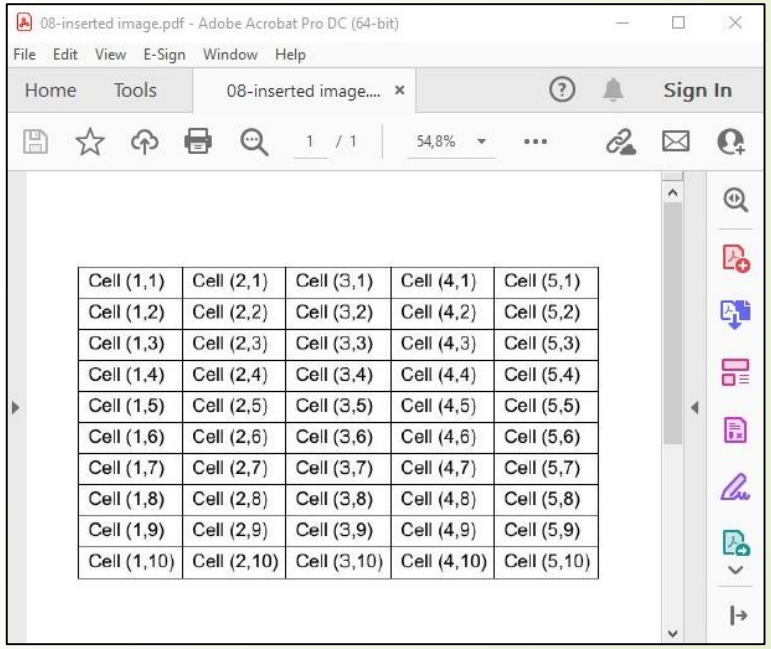
```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "06-table with content.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)

-- use the renderer to buffer the document as an image
renderer=.bsf~new("org.apache.pdfbox.rendering.PDFRenderer",doc)
image=renderer~renderImage(0)

-- write the buffered image to the destination file
writer=bsf.import("javax.imageio.ImageIO")
file=.bsf~new("java.io.File","07-saved image.jpg")
writer~write(image, "jpeg", file)

-- get java support
::requires "BSF.CLS"
```



Cell (1,1)	Cell (2,1)	Cell (3,1)	Cell (4,1)	Cell (5,1)
Cell (1,2)	Cell (2,2)	Cell (3,2)	Cell (4,2)	Cell (5,2)
Cell (1,3)	Cell (2,3)	Cell (3,3)	Cell (4,3)	Cell (5,3)
Cell (1,4)	Cell (2,4)	Cell (3,4)	Cell (4,4)	Cell (5,4)
Cell (1,5)	Cell (2,5)	Cell (3,5)	Cell (4,5)	Cell (5,5)
Cell (1,6)	Cell (2,6)	Cell (3,6)	Cell (4,6)	Cell (5,6)
Cell (1,7)	Cell (2,7)	Cell (3,7)	Cell (4,7)	Cell (5,7)
Cell (1,8)	Cell (2,8)	Cell (3,8)	Cell (4,8)	Cell (5,8)
Cell (1,9)	Cell (2,9)	Cell (3,9)	Cell (4,9)	Cell (5,9)
Cell (1,10)	Cell (2,10)	Cell (3,10)	Cell (4,10)	Cell (5,10)

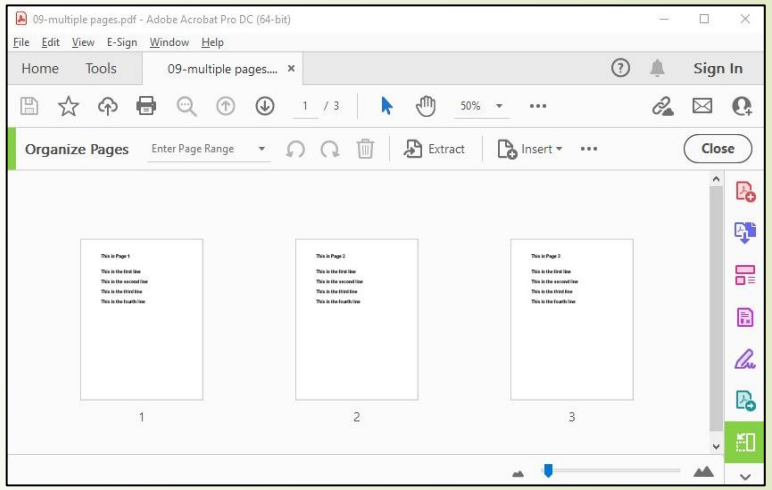
“08. Inserting Image to a PDF Document.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- import the image file
xclass = "org.apache.pdfbox.pdmodel.graphics.image.PDImageXObject"
xobject=BSF.loadClass(xclass)
image=xobject~createFromFile("07-saved image.jpg",doc)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)

-- use the content stream to insert content
cont~drawImage(image,0,0)
cont~close

-- save and close the document file
doc~save("08-inserted image.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```



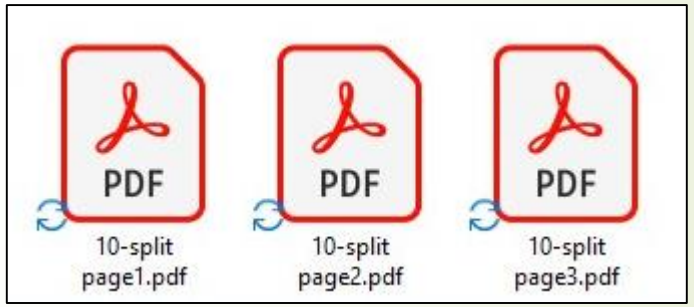
“09. Adding Multiple Pages to a PDF Document.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- create a new document
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")

-- start the loop for each page
num= 3
do i=1 to num

-- add a blank page the document
page.i=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page.i)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page.i)

-- define font type
fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert content
cont~beginText
cont~setFont(font, 22)
cont~setLeading(25f)
cont~newLineAtOffset( 100, 700 )
cont~showText("This is Page" i)
cont~newLine
cont~newLine
cont~newLine
cont~showText("This is the first line")
cont~newLine
cont~newLine
cont~showText("This is the second line")
cont~newLine
cont~newLine
cont~showText("This is the third line")
cont~newLine
cont~newLine
cont~showText("This is the fourth line")
cont~endText
cont~close
end

-- save and close the document file
doc~save("09-multiple pages.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```



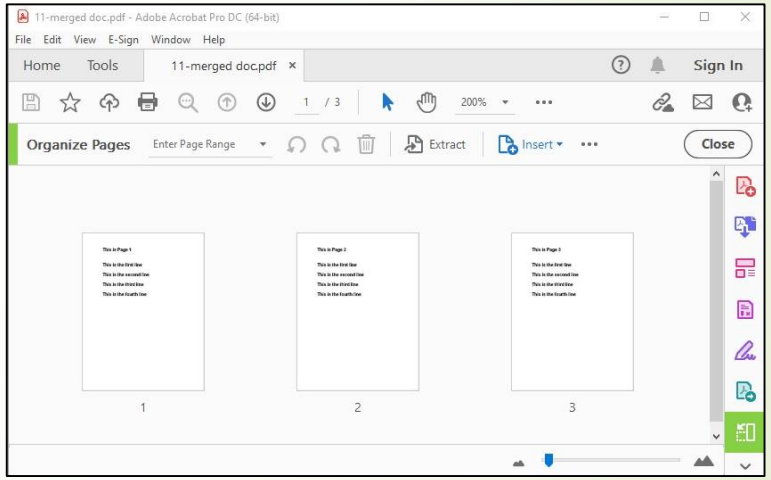
"10. Splitting a PDF Document with Multiple Pages.rex"

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "09-multiple pages.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)

-- use the splitter to split the document
spl=.bsf~new("org.apache.pdfbox.multipdf.Splitter")
pages=spl~split(doc)

-- create an iterator to navigate through the pages
iterator=Pages~listIterator

-- save each page as a new document
do i= 1 to pages~size
pd=iterator~next
pd~save("10-split page"||i||".pdf")
pd~close
end

-- get java support
::requires "BSF.CLS"
```



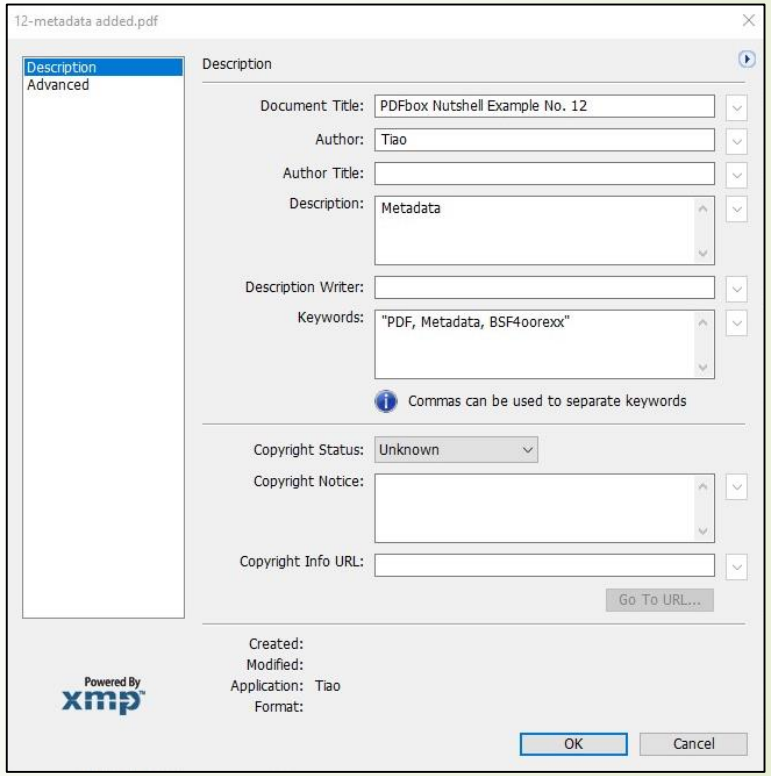
“11. Merging Multiple PDF Documents.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source files
file1=.bsf~new("java.io.File", "10-split page1.pdf")
file2=.bsf~new("java.io.File", "10-split page2.pdf")
file3=.bsf~new("java.io.File", "10-split page3.pdf")

-- import the merger
merger=.bsf~new("org.apache.pdfbox.multipdf.PDFMergerUtility")

-- set the destination file
merger~setDestinationFileName("11-merged doc.pdf")

-- add the source files and save as a new document
merger~addSource(file1)
merger~addSource(file2)
merger~addSource(file3)
mus=BSF.loadClass("org.apache.pdfbox.io.MemoryUsageSetting")
merger~mergeDocuments(mus~setupMainMemoryOnly)

-- get java support
::requires "BSF.CLS"
```



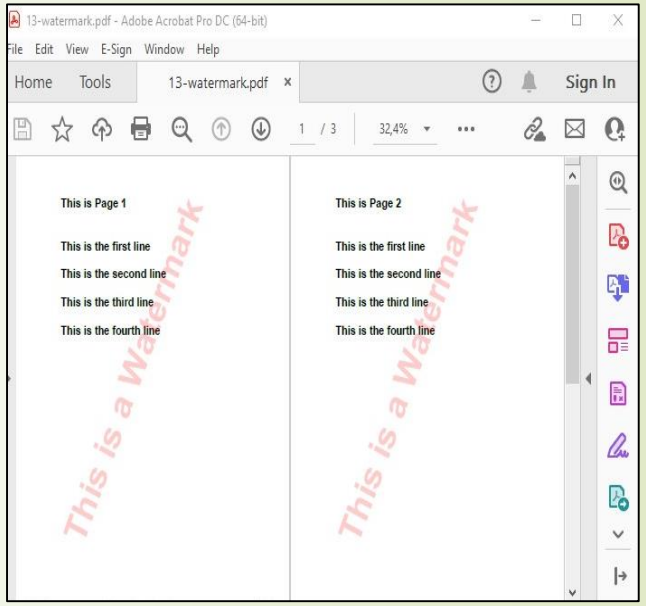
“12. Setting the Document Metadata.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "11-merged doc.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)

-- load the document information and set different
metadata
info=doc~getDocumentInformation
info~setAuthor("Tiao")
info~setTitle("PDFbox Nutshell Example No. 12")
info~setCreator("Tiao")
info~setSubject("Metadata")
info~setKeywords("PDF, Metadata, BSF4oorexx")

-- save and close the document file
doc~save("12-metadata added.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```



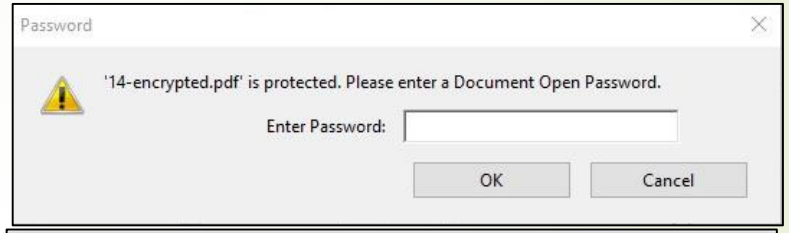
“13. Adding Watermark to a Document.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "12-metadata added.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)

-- count the number of pages
pages=doc~getDocumentCatalog~getPages

-- start the loop for each page
do i= 1 to doc~getNumberOfPages
page=pages~get(i-1)

-- set the append mode
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
append=BSF.loadClass(contclass)~AppendMode~APPEND

-- create a content stream
cont=.bsf~new(contclass,doc,page,append,"true")

-- define font type

fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert content
cont~beginText
cont~setLeading(25f)
cont~setFont(font, 70)

-- set the transparency of the watermark
gsclass="org.apache.pdfbox.pdmodel.graphics.state.PDExtendedGraphicsState"
gs=.bsf~new(gsclass)
gs~setNonStrokingAlphaConstant(0.2)
cont~setGraphicsStateParameters(gs)

-- set the color of the watermark
col=BSF.loadClass("java.awt.Color")~red
cont~setNonStrokingColor(col)

-- set the rotation of the watermark
matrix=.bsf~new("org.apache.pdfbox.util.Matrix")
cont~setTextMatrix(matrix~getRotateInstance(20,150,100))

-- insert the watermark
cont~showText("This is a Watermark")
cont~endText
cont~close
end

-- save and close the document file
doc~save("13-watermark.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```



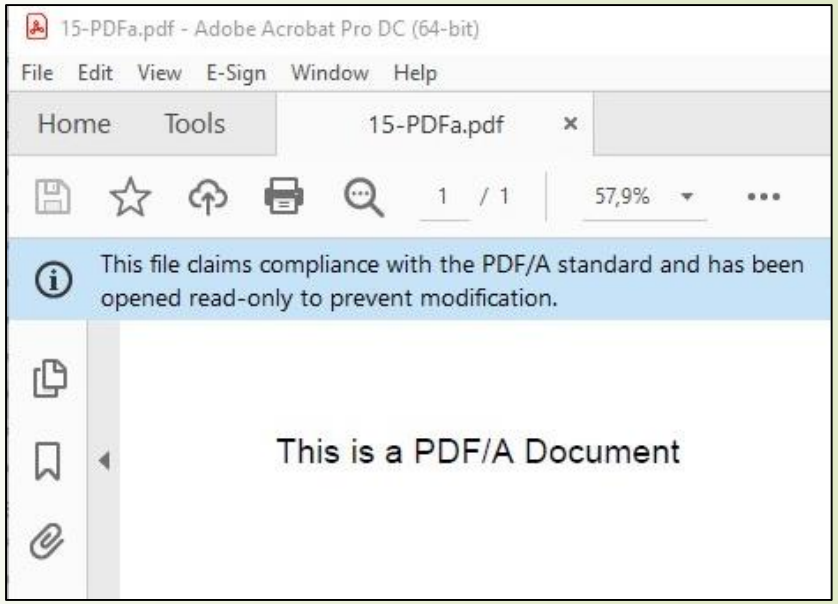
“14. Encrypting a PDF Document.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "13-watermark.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(source)

-- set access permissions
ap=.bsf~new("org.apache.pdfbox.pdmodel.encryption.AccessPermission")
ap~setcanModify(.false)
ap~setCanExtractContent(.false)
ap~setCanPrint(.false)

-- define the standard protection policyencrypt the document
sppclass = "org.apache.pdfbox.pdmodel.encryption.StandardProtectionPolicy"
spp=.bsf~new(sppclass,"ownerpw","userpw",ap)
spp~setEncryptionKeyLength(128)
spp~setPermissions(ap)

--encrypt the document
doc~protect(spp)

-- save and close the document file
doc~save("14-encrypted.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

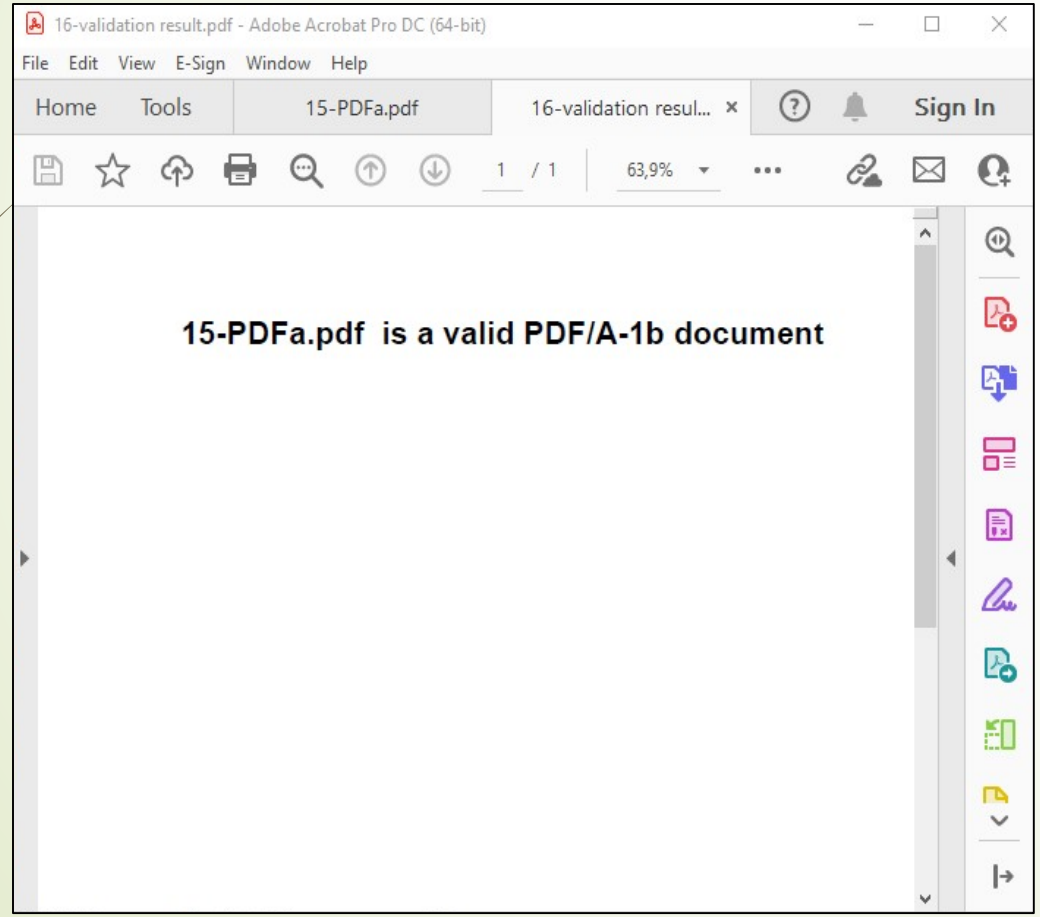


“15. Creating a PDF-A Document.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- load the font file
fontfile=.bsf~new("java.io.File", "resources\arial.ttf")
fontclass = "org.apache.pdfbox.pdmodel.font.PDType0Font"
font=BSF.loadClass(fontclass)~load(doc,fontfile)

-- set the xmp metadata
xmp=BSF.loadClass("org.apache.xmpbox.XMPMetadata")~creat
eXMPMetadata
dc=xmp~createAndAddDublinCoreSchema
dc~setTitle("15-PDFa.pdf");
id=xmp~createAndAddPDFAIdentificationSchema
id~setPart(1);
id~setConformance("B");
serializer=.bsf~new("org.apache.xmpbox.xml.XmpSerializer
")
baos=.bsf~new("java.io.ByteArrayOutputStream")
serializer~serialize(xmp, baos,"true")
metadata=.bsf~new("org.apache.pdfbox.pdmodel.common.PDMe
tadata",doc)
metadata~importXMPMetadata(baos~toByteArray)
doc~getDocumentCatalog~setMetadata(metadata)

-- define the color profile
colorprofile=.bsf~new("java.io.FileInputStream",
"resources\sRGB.icc")
intentclass =
"org.apache.pdfbox.pdmodel.graphics.color.PDOutputInte
nt"
intent=.bsf~new(intentclass,doc,colorprofile)
intent~setInfo("sRGB IEC61966-2.1")
intent~setOutputCondition("sRGB IEC61966-2.1")
intent~setOutputConditionIdentifier("sRGB IEC61966-
2.1")
intent~setRegistryName("http://www.color.org")
doc~getDocumentCatalog~addOutputIntent(intent)

-- create a content stream
cont=.bsf~new("org.apache.pdfbox.pdmodel.PDPageContent
Stream",doc,page)

-- use the content stream to insert content
cont~beginText
cont~setFont(font, 22)
cont~setLeading(25f)
cont~newLineAtOffset( 100, 700 )
cont~showText("This is a PDF/A Document")
cont~endText
cont~close
nocclass="org.apache.pdfbox.pdfwriter.compress.Compres
sParameters"
noc=BSF.loadClass(nocclass)

-- save and close the document file
doc~save("15-PDFa.pdf",noc~NO_COMPRESSION)
doc~close

-- get java support
::requires "BSF.CLS"
```



“16. Validating a PDF-A Document.rex”

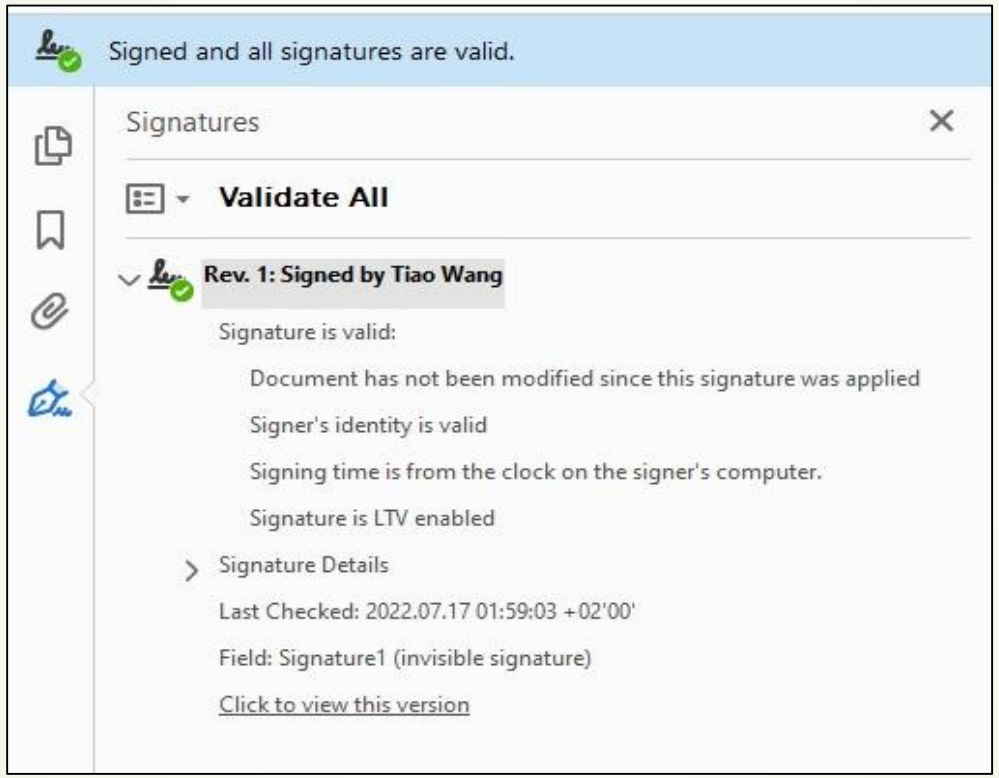



```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file
file="15-PDFa.pdf"

-- use the parser to validate the document and save the result
pclass = "org.apache.pdfbox.preflight.parser.PreflightParser"
parser=.bsf~new(pclass,file)
doc=parser~parse
if doc~validate~isValid=1
then content=file " is a valid PDF/A-1b document"
else content=file " is not a valid PDF/A-1b document"

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)

-- define font type
fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert validation result
cont~beginText
cont~setFont(font, 22)
cont~setLeading(25f)
cont~newLineAtOffset(100, 700)
cont~showText(content)
cont~endText
cont~close

-- save and close the document file
doc~save("16-validation result.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

“17. Creating a Digital Signature.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
infile=.bsf~new("java.io.File", "16-validation result.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
doc=importdoc~loadPDF(infile)

-- configure the signature settings
sigc="org.apache.pdfbox.pdmodel.interactive.digitalsignature.PDSignature"
sig=.bsf~new(sigc)
filter=BSF.loadClass(sigc)~FILTER_ADOBE_PPKLITE
sf=BSF.loadClass(sigc)~SUBFILTER_ADBE_PKCS7_DETACHED
sig~setFilter(filter)
sig~setSubFilter(sf)
sig~setName("Tiao Wang")
sig~setLocation("Vienna, Austria")
sig~setReason("Testing")
cal=BSF.loadClass("java.util.Calendar")~getInstance
sig~setSignDate(cal)
doc~addSignature(sig)

-- prepare the signature generator
password=.bsf~new("java.lang.String", "123456")~toCharArray
keystore=BSF.loadClass("java.security.KeyStore")~getInstance("PKCS12")
ksfis=.bsf~new("java.io.FileInputStream", "resources\keystore.p12")
keystore~load(ksfis,password)
pk=keystore~getKey(test,password)
certchain=keystore~getCertificateChain(test)
certList=BSF.loadClass("java.util.Arrays")~asList(certchain)
cert =certlist~get(0)
certstore=.bsf~new("org.bouncycastle.cert.jcajce.JcaCertStore",certlist)

-- create the signature generator
gen=.bsf~new("org.bouncycastle.cms.CMSSignedDataGenerator")
sbclass="org.bouncycastle.operator.jcajce.JcaContentSignerBuilder"
sha1signer=.bsf~new(sbclass,"SHA256WithRSA")~build(pk)
pbclass="org.bouncycastle.operator.jcajce.JcaDigestCalculatorProviderBuilder"
calc=.bsf~new(pbclass)~build
igclass="org.bouncycastle.cms.jcajce.JcaSignerInfoGeneratorBuilder"
infogen=.bsf~new(igclass,calc)~build(sha1signer, cert)
gen~addSignerInfoGenerator(infogen)
gen~addCertificates(certstore)

-- set the destination file
outfile=.bsf~new("java.io.File", "17-signed.pdf")
fos=.bsf~new("java.io.FileOutputStream", outfile)
exsign=doc~saveIncrementalForExternalSigning(fos)

-- convert the signing data to the cms format
x=exsign~getContent
io=BSF.loadClass("org.apache.commons.io.IOUtils")
ba= io~toByteArray(x)
cmsdata=.bsf~new("org.bouncycastle.cms.CMSProcessableByteArray",ba)

-- generate the signature
signed=gen~generate(cmsdata,"false")
cmssig=signed~getEncoded

-- add the signature to the document
exsign~setSignature(cmssig)

-- get java support
::requires "BSF.CLS"
```

“17. Creating a Digital Signature.rex”



“18. Verifying a Digital Signature.rex”

```
-- change directory to program location
parse source  . . pgm
call directory filespec('L', pgm)

-- define the source file and import it
source=.bsf~new("java.io.File", "17-signed.pdf")
importdoc=BSF.loadClass("org.apache.pdfbox.Loader")
fis =.bsf~new("java.io.FileInputStream","17-signed.pdf")
doc=importdoc~loadPDF(source)

-- extract the signature and the signed content
sig=doc~getSignatureDictionaries~get(0)
sigdata=sig~getContents
signeddata=sig~getSignedContent(fis)

-- convert the signed content to the cms format
pbclass="org.bouncycastle.cms.CMSProcessableByteArray"
cmsdata=.bsf~new(pbclass,signeddata)
cms=.bsf~new("org.bouncycastle.cms.CMSSignedData",cmsdata,sigdata)

-- load the certificate
signerinfo=cms~getSignerInfos~getSigners~iterator~next
cert=cms~getCertificates~getMatches(signerInfo~getSID)~iterator~next

-- create the verifier
vbclass="org.bouncycastle.cms.jcajce.JcaSimpleSignerInfoVerifierBuilder"
vbuilder=.bsf~new(vbclass)
provider=.bsf~new("org.bouncycastle.jce.provider.BouncyCastleProvider")
vbuilder~setProvider(provider)
verifier=vbuilder~build(cert)

-- verify the signature and save the result
result=signerinfo~verify(verifier)
if result=1 then content=source~getName "has a valid signature"
else content=source~getName "has a invalid signature"

-- create a new document and add a blank page
doc=.bsf~new("org.apache.pdfbox.pdmodel.PDDocument")
page=.bsf~new("org.apache.pdfbox.pdmodel.PDPage")
doc~addPage(page)

-- create a content stream
contclass = "org.apache.pdfbox.pdmodel.PDPageContentStream"
cont=.bsf~new(contclass,doc,page)

-- define font type
fontclass = "org.apache.pdfbox.pdmodel.font.Standard14Fonts"
fname = BSF.loadClass(fontclass)~FontName~HELVETICA_BOLD
font=.bsf~new("org.apache.pdfbox.pdmodel.font.PDType1Font",fname)

-- use the content stream to insert the verify result
cont~beginText
cont~setFont(font, 22)
cont~setLeading(25f)
cont~newLineAtOffset(100, 700)
cont~showText(content)
cont~endText
cont~close

-- save and close the document file
doc~save("18-signature result.pdf")
doc~close

-- get java support
::requires "BSF.CLS"
```

Examples
Demonstration

Questions and Answers