1 Overview

ComPDFKit PDF SDK is a powerful PDF library for developers to add robust PDF functionality to their applications, which ships with simple-to-use Java APIs to offer document viewing, creation, searching, annotation, and editing. With this SDK, even developers with limited knowledge of PDF can quickly build a professional PDF viewer and editor with just a few lines of code on their Java programs. ComPDFKit PDF SDK for Java could be integrated into all development platforms like mobile, desktop, web, and Linux programs, empowering the abilities of PDF processing.

1.1 ComPDFKit PDF SDK

ComPDFKit PDF SDK consists of two elements as shown in the following picture.

The element for ComPDFKit PDF SDK for Java:

- **PDF Core API**

  A parser for PDF documents (also known as PDF Data layer) doesn't include any UI components. If you just need to render a PDF document without displaying any content inside the PDF, you can just integrate this into your Java application.

1.2 Key Features

Annotations component offers:

- Create, edit and remove annotations, including Note, Link, Free Text, Line, Square, Circle, Highlight, Underline, Squiggly, Strikeout, Ink, and Stamp.
- Support for annotation appearances.
- Import and export annotations to/from XFDF.
- Support for annotation flattening.

Forms component offers:

- Create, edit and remove form fields, including Push Button, Check Box, Radio Button, Text Field, Combo Box, List Box.
- Fill PDF Forms.
- Support for PDF form flattening.

Document Editor component offers:

- PDF manipulation, including Split pages, Merge pages, and Merge files.
- Page edit, including Add pages, Delete pages, Insert pages, Move pages, Rotate pages, Replace pages, Exchange pages, Reverse pages, Scale pages, Crop pages, and Copy & Paste pages.
- Access document information.
- Extract images.

Security component offers:

- Encrypt and decrypt PDFs, including Permission setting and Password protected.

2 Get Started

ComPDFKit is a powerful PDF SDK. It is easy to embed ComPDFKit PDF SDK in your Java application with a few lines of Java code. Take just a few minutes and get started.

The following sections introduce the requirements, structure of the installation package, and how to make a PDF Reader/Editor in Java with ComPDFKit PDF SDK.

2.1 Requirements

ComPDFKit PDF SDK for Java requires apps to enable Java 8 language features to build.

- Programming Environment: Java JDK 1.8 and higher.
- Supported Platforms
    - Mac (intel & M1)
    - Linux (x86)

2.2 Java Package Structure

The package of ComPDFKit PDF SDK for Java includes the following files:

- ***"samples"*** - A folder containing Android sample projects.
    - ***"Annotations"*** - A PDF viewer with full types of annotation editing, including adding annotations, modifying annotations, annotation lists, etc.
    - ***"Forms"*** - A PDF viewer with full types of forms editing, including radio button, combo box, etc.
    - ***"DocsEditor"*** - A PDF viewer with page editing, including inserting/deleting pages, extracting pages, reordering pages, etc.
    - ***"TestFile"*** - A folder containing test files.
    - ***"lib"*** - A folder containing ***"compdfkit-1.0-SNAPSHOT"***.
      compdfkit-1.0-SNAPSHOT is PDF Core API.
- ***"api_reference_Java"*** - API reference.
- ***"developer_guide_Java.pdf"*** - Developer guide.
- ***"release_notes.txt"*** - Release information.
- ***"legal.txt"*** - Legal and copyright information.

2.3 How to Run a Demo

ComPDFKit PDF SDK for Java provides multiple demos for developers to learn how to call the SDK on Java applications. You can find them in the ***"samples"*** folder. In this guide, we take ***"DocumentInfoTest"*** as an example to show how to run it on Java applications.

1. Import the ***"samples"*** project on IDE.
2. Navigate to the ***"JAVA"*** folder which is in ***"DocumentInfoTest"*** file and run the sample with. For example `/samples/DocumentInfoTest/JAVA`.

   ```
   ./RunTest.sh
   ```

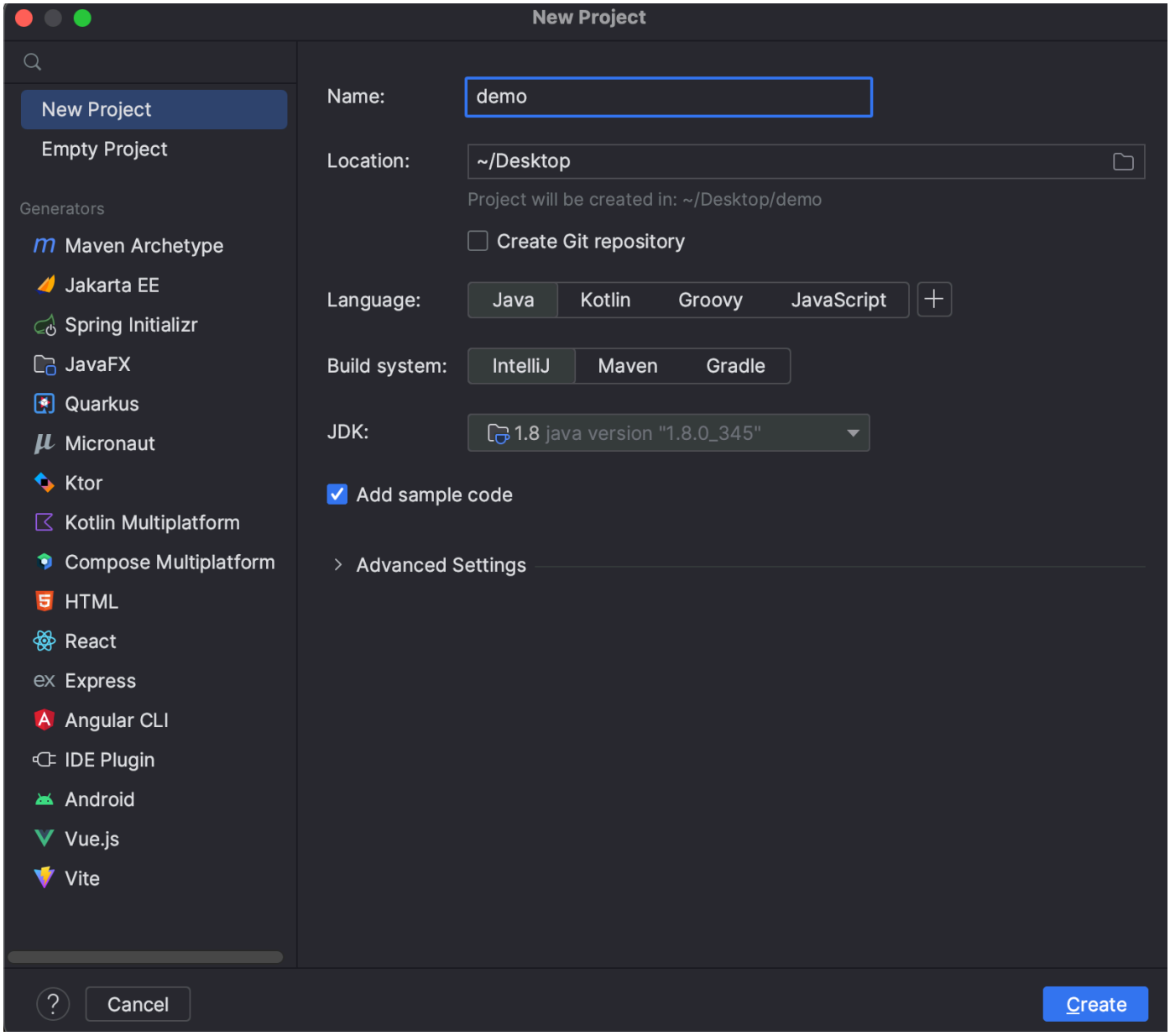
   Note: *This is a demo project, presenting completed ComPDFKit PDF SDK functions. The functions might be different based on the license you have purchased. Please check that the functions you choose work fine in this demo project.* The output files will be in `/samples/out`.

2.4 How to Use ComPDFKit PDF SDK for Java in Java Applications

This section will help you to quickly get started with ComPDFKit PDF SDK to make a Java application with step-by-step instructions.

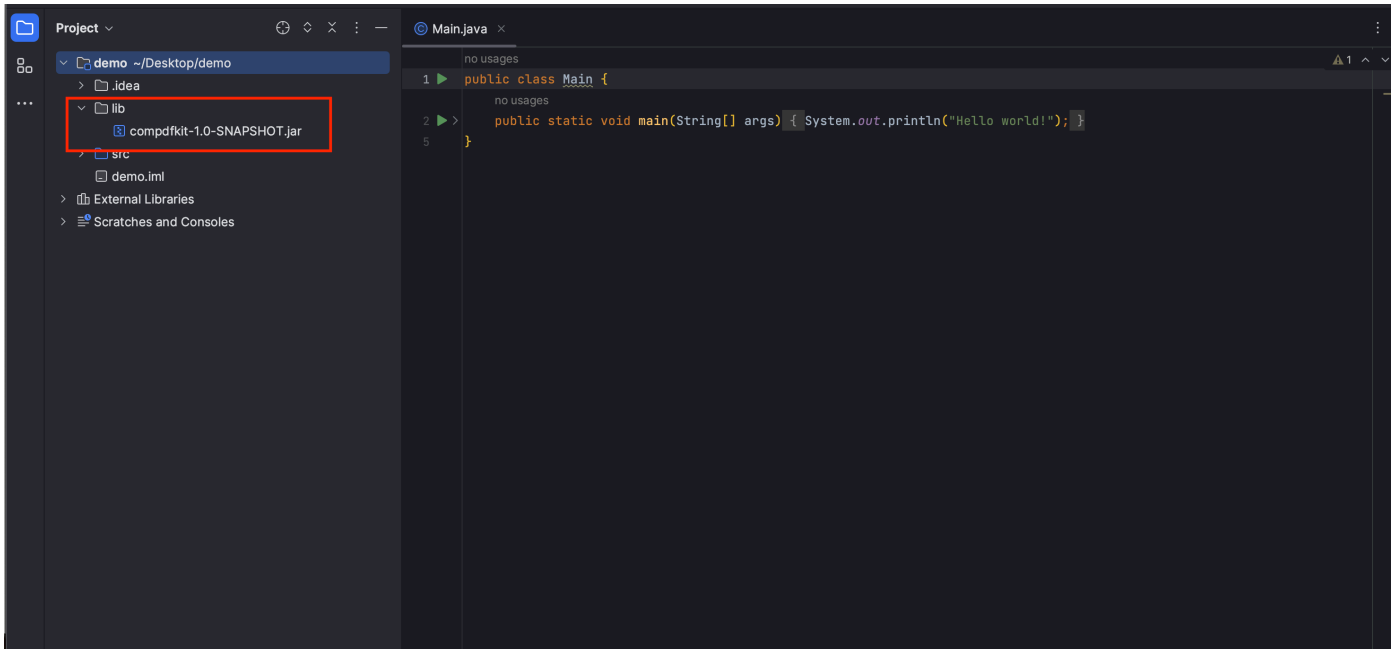
2.4.1 Create a New Project

1. Create a Java application using the IDE. Here we create a new **Java** project.

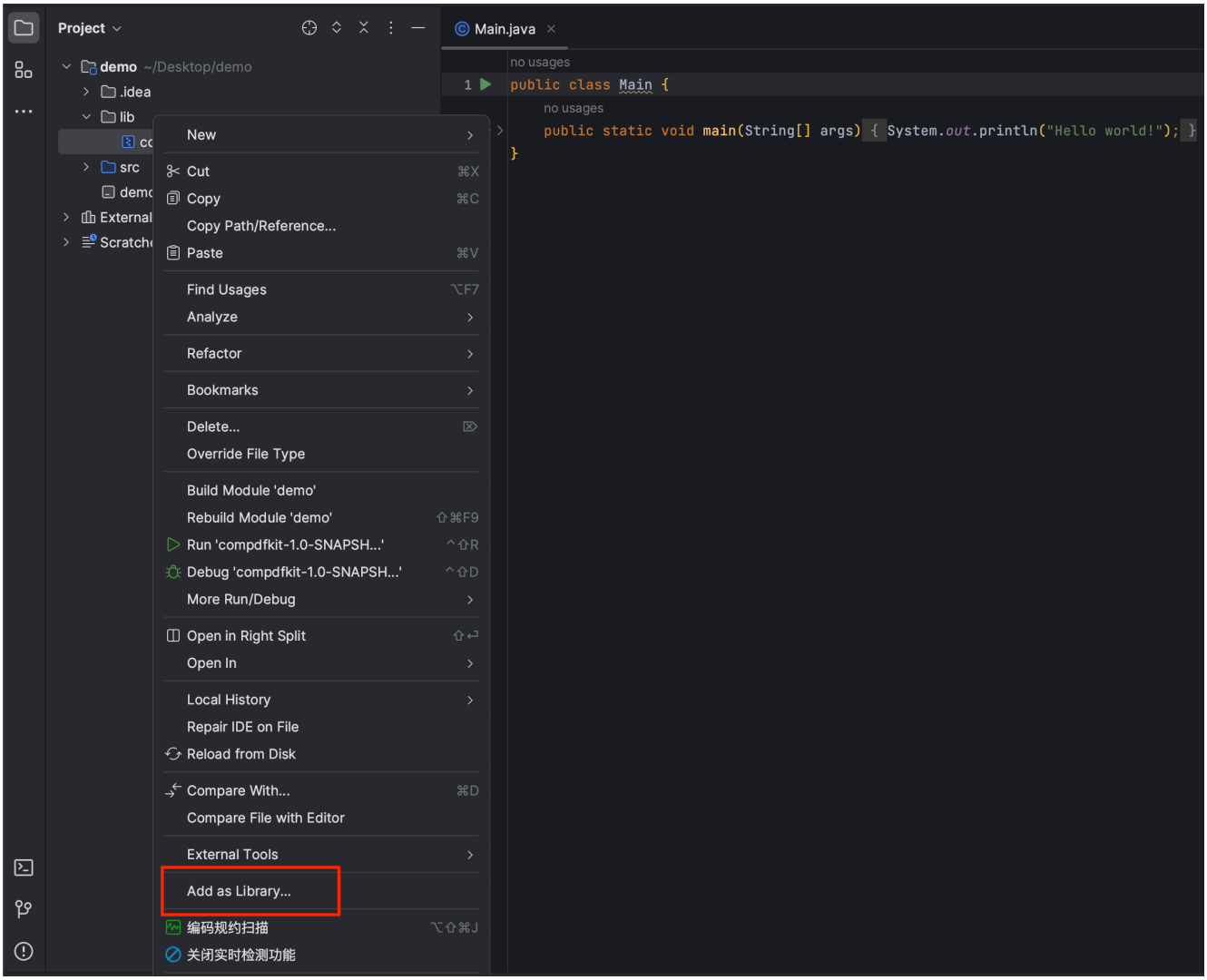


2.4.2 Add the Package of ComPDFKit PDF SDK for Java

1. The first thing we need to do is to import ComPDFKit PDF SDK for Java. Copy ***"lib"*** to the **project**.



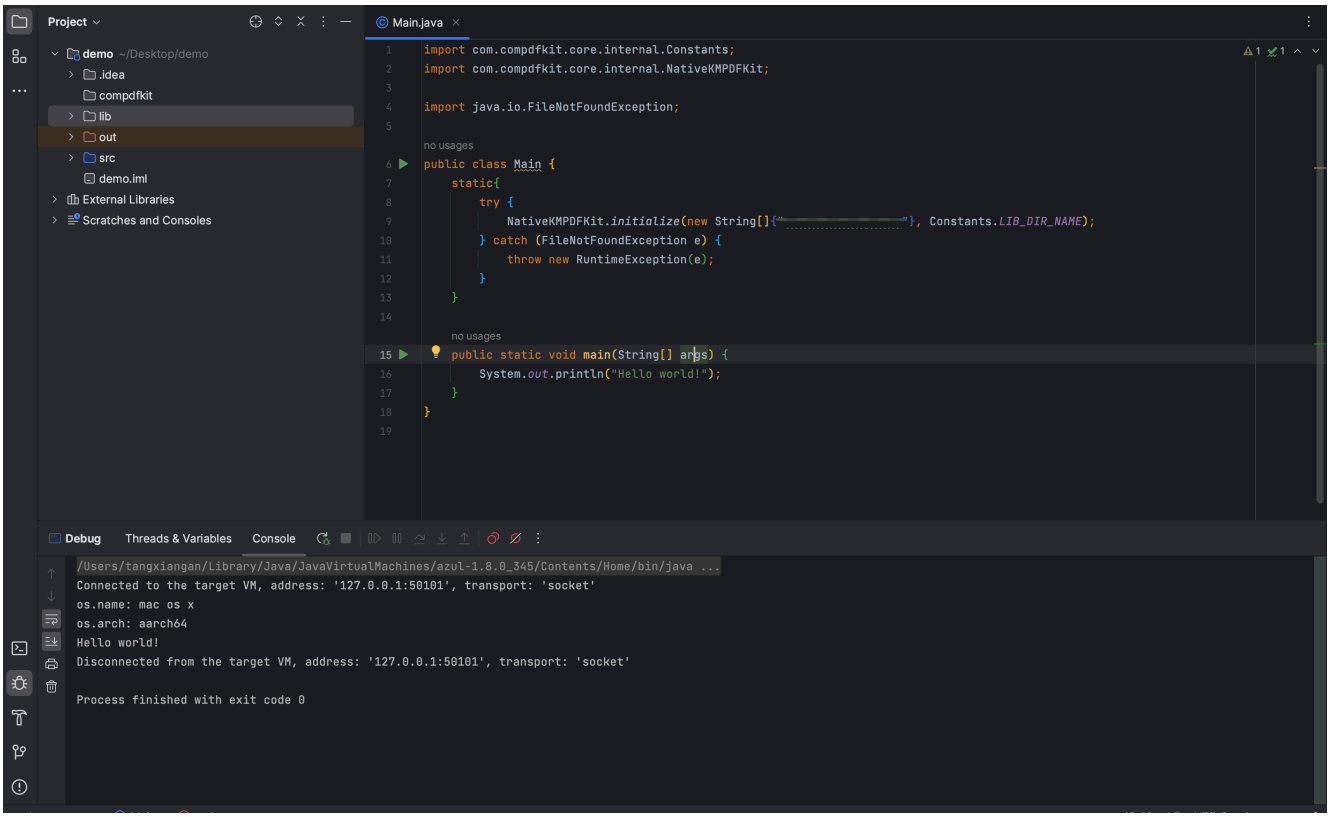
2. Right click on the SDK and select **Add as Library...**.



3. Add the following code to **Main.java**:

```
static{
    try {
       NativeKMPDFKit.initialize(new String[]{"yourKey"}, Constants.LIB_DIR_NAME);
    } catch (FileNotFoundException e) {
          throw new RuntimeException(e);
    }
}
```

4. Start the `main` method to print out the information of your computer's system version. If the information is printed successfully, you have integrated ComPDFKit PDF SDK for Java successfully!



2.4.3 Troubleshooting

2. Problems

   If you meet some other problems when integrating our ComPDFKit PDF SDK for Java, feel free to contact ComPDFKit team.

2.5 Samples

The Samples use preset parameters and documentation to call the API of ComPDFKit PDF SDK for Java for each function without UI interaction or parameter settings. This is achieved through modular code examples. The functions include creating, getting, and deleting various types of annotations and forms, extracting text and images, encrypting and decrypting documents, adding watermarks and bates numbers, and more.

These projects not only demonstrate the best practices for each function but also provide detailed introductions. The impact of each function on PDF documents can be observed in the output directory. With the help of the Samples, you can quickly learn how to use the functions you need and apply them to your projects.

Name	Description
Annotation	Print the annotation list information, set the annotations (including markup, note, ink, free text, circle, square, line annotations), and delete the annotations.
AnnotationImportExport	Export and import annotations with an xfdf file.
InteractiveForms	Print form list information, set up interactive forms (including text, checkbox, radio button, button, list, combo boxes and deleting forms), and fill out form information.
PDFPage	Manipulate PDF pages, including inserting, splitting, merging, rotating, and replacing, etc.
DocumentInfo	Display the information of PDF files like the author, created time, etc.
Encrypt	Set passwords to encrypt PDFs and set document permissions. Allow decrypting PDFs.
Watermark	Create text/image watermarks and delete watermarks.
Compare	Compare the differences between two documents, providing both content comparison and overlay comparison.

3 Guides

If you're interested in the features mentioned in Overview section, please go through our guides to quickly add PDF viewing, annotating, and editing to your application. The following sections list some examples to show you how to add document functionalities to Java apps using the APIs of ComPDFKit PDF SDK for Java.

3.1 Basic Operations

There are a few commonly used basic operations when working with documents.

3.1.1 Open a Document

Open a local file.

```
CPDFDocument document = new CPDFDocument();
// Open document
CPDFDocument.PDFDocumentError error = document.open(pdfUri);
if (error == CPDFDocument.PDFDocumentError.PDFDocumentErrorPassword) {
  // The document is encrypted and requires a password to open.
  error = document.open(pdfUri, "password");
}
if (error == CPDFDocument.PDFDocumentError.PDFDocumentErrorSuccess) {
  // The document is opened successfully and data can be parsed and manipulated.
} else {
  //The PDF file is failed to open. You can refer to the API file for specific error
messages.
}
//Check if the document has been repaired during the opening process.
if (document.hasRepaired()) {
  //The repaired document needs to be saved first and then opened, otherwise any edits
or modifications made to the repaired document cannot be saved.
  //Perform a save operation
  ...
}
```

Following are the descriptions of error codes in `CPDFDocumentError`:

Error	Description
PDFDocumentErrorNoReadPermission	No reading permission.
PDFDocumentNotVerifyLicense	The license doesn't allow the permission.
PDFDocumentErrorSuccess	Open PDF file successfully.
PDFDocumentErrorUnknown	Unknown error.
PDFDocumentErrorFile	File not found or could not be opened.
PDFDocumentErrorFormat	File not in PDF format or corrupted.
PDFDocumentErrorPassword	Password required or incorrect password.
PDFDocumentErrorSecurity	Security scheme not supported.
PDFDocumentErrorPage	Page not found or content error.

Note: *This is a time-consuming process, so it needs to be executed in a sub-thread.*

3.1.2 Save a Document

Save the document.

```
//First, see whether the document has been modified. If there are any modifications, save it.
if (document.hasChanges()) {
  boolean res;
  //Save to original document
  res = document.save(CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveNoIncremental);
  //res = document.save(CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveIncremental);
  //res = document.save(CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveRemoveSecurity);
  //Save to the specified path; PDFDocumentSaveRemoveSecurity indicates whether to delete the password of the document.
  //res = document.save(uri, CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveRemoveSecurity);
}
//Close the document and release resources
document.close();
```

Note: *This is a time-consuming process, so it needs to be executed in a sub-thread. Once the process is completed, the closing process of the document should be executed in the main thread, such as destroying the Activity.*

Incremental saving, is the strategy that ComPDFKit uses to ensure saving is as fast as possible. This strategy always appends changes to the end of the document, and it never deletes anything. However, even though this is the fastest way to save a document, it does come with the cost of increasing the file size with each save. You can read more about incremental saving and full saving of PDFs here.

In most cases, the increase in file size is negligible. However, in some cases, you may want to prioritize file size over saving performance. Below you'll find some strategies to prevent file size from growing unnecessarily when saving changes to a document.

If the document is saved in a new path (Not the original storage path), there is going to trigger a non-incremental document save, which will rewrite the entire document instead of appending changes at the end.

3.2 Annotations

PDF annotations refer to adding, editing, and sharing various multimedia contents on PDF documents, such as note, link, free text, shapes, markup, ink, stamps, etc. By adding annotations, users can more conveniently mark, annotate, revise, comment, sign, and share PDF documents.

PDF annotations are very useful for document communication, review, and revision, and can improve work efficiency and document quality. At the same time, ComPDFKit PDF SDK for Java provides different annotation features and tools, and developers can choose suitable features and tools according to their needs.

3.2.1 Annotation Types

ComPDFKit PDF SDK supports all common annotation types:

Type	Description	Class
Note	Add notes to PDF documents to mark important text, tables, images, and more, making it convenient for you and other readers to read and understand.	`CPDFTextAnnotation`
Link	Add hyperlinks to PDF documents to link to web pages, emails, or specific locations in other documents.	`CPDFLinkAnnotation`
Free Text	Add text and comments to PDF documents.	`CPDFFreetextAnnotation`
Shapes: Square, Circle, and Line, and Arrow	Add graphic elements like shapes, lines, arrows, or images to PDF documents to emphasize or illustrate certain content.	`CPDFSquareAnnotation` `CPDFCircleAnnotation` `CPDFLineAnnotation`
Markup: Highlight, Underline, Strikeout, and Squiggly	Add markups to PDF documents to highlight, emphasize, or illustrate specific content, such as important paragraphs, lines or words, keywords or tables, etc.	`CPDFHighlightAnnotation` `CPDFUnderlineAnnotation` `CPDFStrikeoutAnnotation` `CPDFSquigglyAnnotation`
Stamp	Add stamps to PDF documents to identify and verify the source and authenticity of documents, using the formats of numbers or images to represent signers.	`CPDFStampAnnotation`
Ink	Draw the brush trails freely anywhere in PDF documents, making it easy for users to add doodles, charts, sketches, signatures, or other custom content.	`CPDFInkAnnotation`

ComPDFKit PDF SDK for Java supports most annotation types defined in PDF Reference and provides APIs for annotation creation, properties access and modification, appearance setting, and drawing. These standard annotations can be read and written by many apps, such as Adobe Acrobat and Apple Preview.

3.2.2 Access Annotations

`CPDFAnnotation` is the base class for all annotations. It has no practical use and cannot instantiate objects, only subclasses such as `CPDFCircleAnnotation` and `CPDFTextAnnotation` are useful. Any unknown or unsupported annotations will be filtered out when parsing a PDF.

Access the list of annotation by using the following method:

```
CPDFDocument document = new CPDFDocument();
document.open("fileUrl");
int pageCount = document.getPageCount();
if (pageCount > 0){
  for (int pageIndex = 0; pageIndex < pageCount; pageIndex++) {
    List<CPDFAnnotation> annotations =
document.pageAtIndex(pageIndex).getAnnotations();
    if (annotations != null && !annotations.isEmpty() ){
      for (CPDFAnnotation annotation:annotations){
        if (annotation == null || !annotation.isValid()){
           continue;
        }
        // Do other annotation processings
      }
    }
  }
}
```

The elements of the array will most likely be typed into subclasses of the `CPDFPage` class.

3.2.3 Create & Edit Annotations

ComPDFKit PDF SDK for Java includes a wide variety of standard annotations, and each of them is added to the project in a similar way.

- Note

  Add a sticky note (text annotation) to a PDF Document page by using the following method.

  ```
  // Open PDF document
  CPDFDocument document = new CPDFDocument();
  CPDFDocument.PDFDocumentError error = document.open("pdfPath");
  if (error == CPDFDocument.PDFDocumentError.PDFDocumentErrorSuccess) {
    //The document is opened successfully and data can be parsed and manipulated.
    if (document.getPageCount() > 0){
      //Insert page number
      int pageNumber = 0;
      //get page instance of pdf
      CPDFPage page = document.pageAtIndex(pageNumber);
  ```

```
    CPDFTextAnnotation textAnnotation = (CPDFTextAnnotation)
page.addAnnot(CPDFAnnotation.Type.TEXT);
    //Get the actual size of the page you want to insert
    RectF insertRect = new RectF(0,0,50,50);
    textAnnotation.setRect(insertRect);
    textAnnotation.setContent("Hello world!");
    textAnnotation.updateAp();
  }
} else {
  //The PDF file failed to open. You can refer to the API file for specific error
messages.
}
```

- Link

  Add a hyperlink or intra-document link annotation to a PDF Document page by using the following method.

```
CPDFLinkAnnotation linkAnnotation = (CPDFLinkAnnotation)
page.addAnnot(CPDFAnnotation.Type.LINK);
RectF insertRect = new RectF(0,0,100,100);
//Coordinate conversion
linkAnnotation.setRect(insertRect);

//Destination
CPDFGoToAction goToAction = new CPDFGoToAction();
CPDFDestination destination = new
CPDFDestination(pageNumber,0,pageSize.height(),1f);
goToAction.setDestination(document, destination);
linkAnnotation.setLinkAction(goToAction);

//Website
CPDFUriAction uriAction = new CPDFUriAction();
uriAction.setUri("website");
linkAnnotation.setLinkAction(uriAction);
linkAnnotation.updateAp();
```

- Free Text

  Add a free text annotation to a PDF Document page by using the following method.

```
CPDFFreetextAnnotation freetextAnnotation = (CPDFFreetextAnnotation)
page.addAnnot(CPDFAnnotation.Type.FREETEXT);
RectF insertRect = new RectF(0,0,100,100);
freetextAnnotation.setRect(insertRect);
freetextAnnotation.setContent("Hello World!");
freetextAnnotation.setFreetextAlignment(CPDFFreetextAnnotation.Alignment.ALIGNMENT_
LEFT);
freetextAnnotation.updateAp();
```

For the text annotations being edited, ComPDFKit also provides an API to modify the font and font color properties. These do not need to end the editing state of text annotation, and you can directly modify the properties in the editing process to change the annotation properties.

```
CPDFTextAttribute textAttribute = freetextAnnotation.getFreetextDa();
textAttribute.setColor(Color.YELLOW.getRGB());
textAttribute.setFontSize(14);
textAttribute.setFontName("fontName");
freetextAnnotation.setFreetextDa(textAttribute);
freetextAnnotation.updateAp();
```

- Shapes

  Add a shape annotation to a PDF Document page by using the following method.

```
// Square
CPDFSquareAnnotation squareAnnotation = (CPDFSquareAnnotation)
page.addAnnot(CPDFAnnotation.Type.SQUARE);
RectF insertRect = new RectF(0,0,100,100);
squareAnnotation.setRect(insertRect);
squareAnnotation.setBorderColor(Color.YELLOW.getRGB());
CPDFBorderStyle borderStyle = new
CPDFBorderStyle(CPDFBorderStyle.Style.Border_Solid, 10, new float[]{8.0F, 0F});
borderStyle.setBorderWidth(10F);
squareAnnotation.setBorderStyle(borderStyle);
squareAnnotation.setBorderAlpha(255);
squareAnnotation.setFillColor(Color.RED.getRGB());
squareAnnotation.setFillAlpha(255);

// Circle
CPDFCircleAnnotation circleAnnotation = (CPDFCircleAnnotation)
page.addAnnot(CPDFAnnotation.Type.CIRCLE);
RectF insertRect = new RectF(0,0,100,100);
circleAnnotation.setRect(insertRect);
circleAnnotation.setBorderColor(Color.YELLOW.getRGB());
CPDFBorderStyle borderStyle = new
CPDFBorderStyle(CPDFBorderStyle.Style.Border_Solid, 10, new float[]{8.0F, 0F});
borderStyle.setBorderWidth(10F);
circleAnnotation.setBorderStyle(borderStyle);
circleAnnotation.setBorderAlpha(255);
circleAnnotation.setFillColor(Color.RED.getRGB());
circleAnnotation.setFillAlpha(255);

// Line
CPDFLineAnnotation lineAnnotation = (CPDFLineAnnotation)
page.addAnnot(CPDFAnnotation.Type.LINE);
float lineWidth = 10f;
PointF startPoint = new PointF(0,0);
PointF endPoint = new PointF(200,200);
```

```
RectF area = new RectF(0,0,200,200);
lineAnnotation.setRect(area);
lineAnnotation.setLinePoints(startPoint, endPoint);
lineAnnotation.setLineType(CPDFLineAnnotation.LineType.LINETYPE_NONE,
CPDFLineAnnotation.LineType.LINETYPE_ARROW);
CPDFBorderStyle borderStyle = new
CPDFBorderStyle(CPDFBorderStyle.Style.Border_Solid,lineWidth,null);
lineAnnotation.setBorderStyle(borderStyle);
lineAnnotation.setBorderAlpha(255);
lineAnnotation.setBorderColor(Color.RED.getRGB())
```

Note: *`CPDFLineAnnotation` properties (`startPoint`, `endPoint`) point is specified in page-space coordinates. Page space is a coordinate system with the origin at the lower-left corner of the current page.*

For each line, users can choose separate styles for the start and the end. The styles are defined by the `CPDFLineStyle` enumeration.

Name	Description
LINETYPE_NONE	No special line ending.
LINETYPE_ARROW	Two short lines meeting in an acute angle to form an open arrowhead.
LINETYPE_CLOSEDARROW	Two short lines meeting in an acute angle as in the LINETYPE_ARROW style and connected by a third line to form a triangular closed arrowhead filled with the annotation's interior color.
LINETYPE_SQUARE	A square filled with the annotation's interior color.
LINETYPE_CIRCLE	A circle filled with the annotation's interior color.
LINETYPE_DIAMOND	A diamond shape filled with the annotation's interior color.

- Markup

  Add a highlight annotation to a PDF Document page by using the following method, and add other markup annotations in a similar way.

```
// First, get array of TextSelection from the specific area on a page.
CPDFPage pdfPage = douc.pageAtIndex(1);
CPDFTextPage pdfTextPage = pdfPage.getTextPage();
RectF selectRect = new RectF(0f, 0f, 500f, 500f);
// Then, add a highlight annotation for the specific area.
highlightAnnotation.setQuadRects(selectRect);
highlightAnnotation.setMarkedText("text");
highlightAnnotation.setRect(annotRect);
highlightAnnotation.updateAp();
```

Note: *Many annotation types are defined as markup annotations because they are used primarily to mark up PDF documents. Markup annotations may be divided into the following: highlight, underline, strikeout, and squiggly.*

- Ink

  Add a Ink annotation to a PDF Document page by using the following method.

```
// Insert the ink annotation into the first page of the PDF document.
CPDFPage page1 = document.pageAtIndex(0);
ArrayList<ArrayList<PointF>> mDrawing = new ArrayList<>();
ArrayList<PointF> arc1 = new ArrayList<>();
arc1.add(new PointF(100, 100));
arc1.add(new PointF(110, 110));
arc1.add(new PointF(120, 120));
mDrawing.add(arc1);
ArrayList<PointF> arc2 = new ArrayList<>();
arc2.add(new PointF(115, 115));
arc2.add(new PointF(130, 130));
arc2.add(new PointF(160, 160));
mDrawing.add(arc2);
float scaleValue = 1F;
float borderWidth = 5F;
CPDFInkAnnotation inkAnnotation = (CPDFInkAnnotation)
page1.addAnnot(CPDFInkAnnotation.Type.INK);
inkAnnotation.setColor(Color.RED.getRGB());
inkAnnotation.setAlpha(255);
inkAnnotation.setBorderWidth(borderWidth);
RectF rect = null;
RectF size = document.getPageSize(0);
if (size.isEmpty()) {
     return;
   }
int lineCount = mDrawing.size();
PointF[][] path = new PointF[lineCount][];
for (int lineIndex = 0; lineIndex < lineCount; lineIndex++) {
   ArrayList<PointF> line = mDrawing.get(lineIndex);
   int pointCount = line.size();
   PointF[] linePath = new PointF[pointCount];
   for (int pointIndex = 0; pointIndex < pointCount; pointIndex++) {
        PointF point = line.get(pointIndex);
     // Calculate the smallest Rect that the Path is surrounded by
        if (rect == null) {
          rect = new RectF(point.x, point.y, point.x, point.y);
           } else {
              rect.union(point.x, point.y);
       }
    // Calculate the coordinate points that are converted to the Page and stored in
the linePath collection
        linePath[pointIndex] = point;
```

```
  }
   path[lineIndex] = linePath;
}
float dx = borderWidth / scaleValue / 2;
rect.inset(-dx, -dx);
rect.set(rect);
inkAnnotation.setInkPath(path);
inkAnnotation.setRect(rect);
inkAnnotation.updateAp();
mDrawing.clear();
```

- Stamp

  Add standard, text, and image stamps to a PDF document page by using the following method.

```
// Add standard stamp annotation
int yOffset = 50;
float lastOffset = 0;
for (int i = 0; i < CPDFStampAnnotation.StandardStamp.values().length; i++) {
    CPDFPage page = document.pageAtIndex(4);
    CPDFStampAnnotation.StandardStamp standardStamp =
CPDFStampAnnotation.StandardStamp.values()[i];
    if (standardStamp == null || standardStamp ==
CPDFStampAnnotation.StandardStamp.UNKNOWN) {
        continue;
    }
    // Add Standard stamp
    CPDFStampAnnotation standard = (CPDFStampAnnotation)
page.addAnnot(CPDFAnnotation.Type.STAMP);
    if (standard == null) {
        continue;
    }
    standard.setStandardStamp(standardStamp);
    RectF pageSize = page.getSize();
    RectF insertRect = standard.getRect();
    float defaultWidth = 100F;
    int x = 50;
    if (i == 10) {
        lastOffset = 50;
    }
    if (i >= 10) {
        x = 150;
    }
    yOffset = (int) lastOffset + 10;
    PointF vertex = new PointF(x, yOffset);
    insertRect.set(vertex.x, vertex.y, vertex.x + defaultWidth, vertex.y +
defaultWidth * Math.abs(insertRect.height() / insertRect.width()));
    lastOffset = insertRect.bottom;
    standard.setRect(insertRect);
```

```
    standard.updateAp();
}

// Add text stamp annotations
CPDFPage page = document.pageAtIndex(4);
CPDFStampAnnotation stampAnnotation = (CPDFStampAnnotation)
page.addAnnot(CPDFAnnotation.Type.STAMP);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String date = df.format(new Date());
stampAnnotation.setTextStamp(new CPDFStampAnnotation.TextStamp(
        "ComPDFKit", date, CPDFStampAnnotation.TextStampShape.TEXTSTAMP_RECT.id,
        CPDFStampAnnotation.TextStampColor.TEXTSTAMP_GREEN.id));
RectF insertRect = stampAnnotation.getRect();
insertRect.set( insertRect);
float defaultWidth = 150f;
PointF vertex = new PointF(300, 50);
insertRect.set(vertex.x, vertex.y, vertex.x + defaultWidth, vertex.y + defaultWidth
* Math.abs(insertRect.height() / insertRect.width()));
stampAnnotation.setRect(insertRect);
stampAnnotation.updateAp();

// Add image stamp annotations
CPDFStampAnnotation standard = (CPDFStampAnnotation)
page.addAnnot(CPDFAnnotation.Type.STAMP);
String imagePath = rootDir + "/TestFiles/ComPDFKit.png";
RectF imageInsertRect = new RectF(457, 626.5f, 513, 570.5f);
standard.setRect(imageInsertRect);
if (imagePath.endsWith("png")) {
    File imageFile = new File(imagePath);
    BufferedImage image = null;
    try {
        image = ImageIO.read(imageFile);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    int[] pixel = getPixel(image);
    standard.updateApWithBitmap(pixel, image.getWidth(), image.getHeight());
} else {
    standard.setImageStamp(imagePath);
    standard.updateAp();
}
```

ComPDFKit PDF SDK for Java provides APIs to modify the color appearance of properties in annotations. Only the corresponding RGB values will be used, and additional values are required for transparency to take effect. The following APIs will introduce how to set the color values and transparency of related properties.

Set the draw color of note:

```
/**
 * Method to get / set the opacity for the annotation.
 */
public boolean setAlpha(int alpha);
public int getAlpha();

/**
 * Method to get / set the color for the annotation.
 * @discussion For many annotations ("Circle", "Square") the stroke color. Used for
other annotations as well.
 */
public boolean setColor(int color);
public int getColor();
```

Set the font styles of free text (free text):

```
/**
 * Sets Freetext font style, eg: fontName, fontSize, fontColor.
 */
public boolean setFreetextDa(CPDFTextAttribute freetextDa);
public CPDFTextAttribute getFreetextDa();

/**
 * Sets Freetext font opacity.
 */
public boolean setAlpha(int alpha);
public int getAlpha();
```

Set the stroke color and fill color of shapes (square, circle, line, and arrow):

```
/**
 * Method to get / set the border opacity for the annotation.
 */
public boolean setBorderAlpha(int lineAlpha);
public int getBorderAlpha();

/**
 * Method to get / set the border color for the annotation.
 * @discussion For many annotations ("Circle", "Square") the stroke color. Used for
other annotations as well.
 */
public boolean setBorderColor(int color);
public int getBorderColor();

/**
 * Method to get / set the fill opacity used for drawing the annotation.
 */
public boolean setFillAlpha(int fillAlpha);
```

```
public int getFillAlpha();

/**
 * Method to get / set the fill color used for drawing the annotation.
 */
public boolean setFillColor(int color);
public int getFillColor();
```

Set the color of markup (highlight, underline, strikeout, and squiggly):

```
/**
 * Method to get / set the opacity for the annotation.
 */
public boolean setAlpha(int alpha);
public int getAlpha();

/**
 * Method to get / set the color for the annotation.
 * @discussion For many annotations ("Circle", "Square") the stroke color. Used for
other annotations as well.
 */
public boolean setColor(int color);
public int getColor();
```

3.2.4 Delete Annotations

The code below shows how to remove an annotation from a document.

```
CPDFPage page = document.pageAtIndex(0);
CPDFAnnotation annotation = page.getAnnotations().get(0);
page.deleteAnnotation(annotation);
```

3.2.5 Annotation Appearances

Annotations may contain properties that describe their appearance — such as annotation color or shape. However, these don't guarantee that the annotation will be displayed the same in different PDF viewers. To solve this problem, each annotation can define an appearance stream that should be used for rendering the annotation.

When you modify the annotation properties, you must call the `updateAp()` method in the `CPDFAnnotation` class.

```
public boolean updateAp();
```

3.2.6 Import & Export Annotations

XFDF is an XML-based standard from Adobe XFDF for encoding annotations. An XFDF file will contain a snapshot of a PDF document's annotations and forms. It's compatible with Adobe Acrobat and several other third-party frameworks. ComPDFKit supports both reading and writing XFDF.

- Import from XFDF

  You can import annotations and form fields from an XFDF file to a document like this:

```
CPDFDocument document = readerView.getPDFDocument();
//the path of xfdf
String xfdfPath = "/xfdf/exportXFDF.xfdf";
//the transfer path of audio and video files
String tmpxfdfPath =  "/tmp";
document.importAnnotations(xfdfPath,tmpxfdfPath);
```

- Export to XFDF

  You can export annotations and form fields from a document to an XFDF file like this:

```
//the path of xfdf
String xfdfPath = "/xfdf/exportXFDF.xfdf";
//the transfer path of audio and video files
String tmpxfdfPath = "/tmp";
document.exportAnnotations(xfdfPath,tmpxfdfPath);
```

3.2.7 Flatten Annotations

Annotation flattening refers to the operation that changes annotations into a static area that is part of the PDF document, just like the other text and images in the document. When flattening an annotation, the annotation is removed from the document, while its visual representation is kept intact. A flattened annotation is still visible but is no longer editable by your users or by your app.

Annotations in a PDF document can be flattened in the ComPDFKit by calling `flattenAllPages` and `save` function.

```
// Open document from file path
CPDFDocument document = new CPDFDocument();
document.open(pdfPath, passwod);
// Flatten all pages from document
boolean result =
document.flattenAllPages(CPDFPage.PDFFlattenOption.FLAT_NORMALDISPLAY);
// Save document
if (result) {
  document.save(CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveIncremental);
}
```

3.3 Forms

An interactive form (sometimes referred to as an AcroForm) is a collection of fields for gathering information interactively from the user. Under the hood, PDF form fields are a type of PDF annotation called widget annotations.

ComPDFKit PDF SDK for Java fully supports reading, filling, creating, and editing PDF forms and provides utility methods to make working with forms simple and efficient.

3.3.1 Supported Form Fields

ComPDFKit PDF SDK for Java supports all form types specified by the PDF specification, including:

Type	Description	Annotation Object
Check Box	Select one or more options.	`CPDFCheckboxWidget`
Radio Button	Select one option from the predefined options.	`CPDFRadiobuttonWidget`
Push Button	Create custom buttons on the PDF document that will perform an action when pressed.	`CPDFPushbuttonWidget`
List Box	Select one or more options from the predefined options, similar to the Combo Box.	`CPDFListboxWidget`
Combo Box	Select one option from a drop-down list of available text options.	`CPDFComboboxWidget`
Text	Enter text content such as name, address, email, etc.	`CPDFTextWidget`

`CPDFWidget` is the base class for all form fields and a subclass of `CPDFAnnotation`. Any unknown or unsupported form fields will be filtered out when parsing the PDF.

3.3.2 Create & Edit Form Fields

Create form fields works the same as adding any other annotation, as can be seen in the guides for programmatically creating annotations.

```
int pageNumber = 0;
RectF insertRect = new RectF(0,0,100,100);
CPDFCheckboxWidget checkboxWidget = (CPDFCheckboxWidget)
document.pageAtIndex(pageNumber).addFormWidget(CPDFWidget.WidgetType.Widget_CheckBox);
checkboxWidget.setRect(insertRect);
checkboxWidget.setChecked(true);
checkboxWidget.updateAp();
```

3.3.3 Delete Form Fields

Deleting form fields works the same as deleting annotations, and check deleting annotations in the guides to see more.

```
int page = 0;
document.pageAtIndex(page).deleteAnnotation(annotation);
```

3.3.4 Fill Form Fields

The following example that demonstrates how form fields can be queried and filled with code:

```
int pageNumber = 0;
CPDFPage pdfPage = document.pageAtIndex(pageNumber);
List<CPDFAnnotation> annotations = pdfPage.getAnnotations();
for (CPDFAnnotation annotation:annotations) {
  if (annotation.getType() == CPDFAnnotation.Type.WIDGET) {
    if (((CPDFWidget) annotation).getWidgetType() ==
CPDFWidget.WidgetType.Widget_CheckBox) {
      ((CPDFCheckboxWidget) annotation).setChecked(true);
    }

    if (((CPDFWidget) annotation).getWidgetType() ==
CPDFWidget.WidgetType.Widget_TextField) {
      ((CPDFTextWidget) annotation).setText("Hello world");
    }
  }
}
```

3.3.5 Flatten PDF Forms

PDF Form flattening works the same as annotation flattening, and refer to annotation flattening in the guides to see more.

3.4 Document Editor

ComPDFKit PDF SDK for Java provides a wide range of APIs for document editing operations. These are mostly available through the `CPDFDocument` and `CPDFPage` classes.

The Document Editor of ComPDFKit PDF SDK for Java allows the following PDF processings:

- PDF Manipulation
  - Split pages
  - Merge pages
  - Merge files
- Page Edit
  - Add pages (Add a blank page)
  - Delete pages
  - Extract pages
  - Reverse pages
  - Insert pages (Choose page or image from another document)
  - Move pages
  - Crop pages
  - Scale pages
  - Rotate pages
  - Replace pages
  - Copy pages
  - Paste pages
- Access Document Information
- Extract Images

3.4.1 PDF Manipulation

- Split Pages

  `CPDFDocument` can extract range of pages from one document and put them into another document. If you run this operation multiple times with different page indexes, you can effectively split a PDF into as many documents as you require.

  To split a PDF document into multiple pages, please use the following method:

  1. Create a blank PDF document.

```
CPDFDocument newDocument = CPDFDocument.createDocument();
```

2. Open PDF document that contains the pages you want to split.

```
// Open document from file path
CPDFDocument document = new CPDFDocument();
document.open(pdfPath);
```

3. Extract specific pages from PDF document that you just opened, and import it into the blank PDF document.

```
// Pages that need to be split, e.g. 2 to 5 pages, start insert index is 0
newDocument.importPages(document, 2, 5, 0);
```

4. Save the document.

```
// Save path, don't remove password of pdf file
newDocument.save(savePath,
CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveNoIncremental);
```

- Merge Pages

  ComPDFKit allows you to instantiate multiple `CPDFDocument`, and you can use the `CPDFDocument` API to merge multiple PDF files into a single one.

  To merge PDF documents into one file, please use the following method:

  1. Create a blank PDF document.

```
CPDFDocument mergeDocument = CPDFDocument.createDocument();
```

  2. Open the PDF documents that contain the pages you want to merge.

```
// Open document1 from file path1
CPDFDocument document1 = new CPDFDocument();
document.open(pdfPath1);

// Open document2 from file path2
CPDFDocument document2 = new CPDFDocument();
document2.open(pdfPath2);
```

  3. Merge all the pages from the documents you just opened, and import it into the blank PDF document.

```
mergeDocument.importPages(document1, start, end, mergeDocument.pageCount);
mergeDocument.importPages(document2, start, end, mergeDocument.pageCount);
```

4. Save the document.

```
// Save path, don't remove password of pdf file
mergeDocument.save(file.getAbsolutePath(),
CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveNoIncrem
```

The sample code above allows you to merge all the pages from the two documents. If you're looking to merge or add specific pages from one document to another, you can use `importPages` of `CPDFDocument.importPages(CPDFDocument otherDocument, int[] pages, int insertPosition)` to set specific pages.

- Extract Pages

  `CPDFDocument` can extract range of pages from one document and put them into a blank document. If you run this operation, you can effectively extract a PDF as you require. Refer to split pages for more details.

3.4.2 Page Edit

Page manipulation is the ability to perform changes to pages.

- To delete pages from a PDF document, use the function `CPDFDocument.removePages(int[] pageIndexs)`.
- To insert a blank page into a PDF document, use the function `CPDFDocument.insertBlankPage(int pageIndex, float width, float height)`.
- To insert an image as an entire page into a PDF document, use the function `CPDFDocument.insertPageWithImagePath(int pageIndex, float width, float height, String imagePath)`.
- To insert a specific page from one document to another, use the function `CPDFDocument.importPages(CPDFDocument otherDocument, int start, int end, int insertPosition)`.
- To move a page to a new location, use the function `CPDFDocument.movePage(int fromIndex, int toIndex)`.
- To exchange the location of two document pages, use the function `CPDFDocument.exchangePage(int firstIndex, int secondIndex)`.
- To replace original document pages with new pages from a different document, use function `CPDFDocument.removePages(int[] pageIndexs)` and `CPDFDocument.importPages(CPDFDocument otherDocument, int start, int end, int insertPosition)`.
- To rotate a page in a PDF document, refer to the following method in the `CPDFPage` class.

```
// Rotation on a page. Must be 0, 90, 180 or 270 (negative rotations will be
"normalized" to one of 0, 90, 180 or 270).
// Some PDF's have an inherent rotation and so -[rotation] may be non-zero when a
PDF is first opened.
CPDFPage pdfPage = pdfDocument.pageAtIndex(pageIndex);
int rotation = pdfPage.getRotation();
pdfPage.setRotation(rotation + 90);
```

3.4.3 Document Information

To edit document information, refer to the following method in the `CPDFDocument` class.

```
public CPDFInfo getInfo() {
    //...
}

public boolean setInfo(CPDFInfo info){
  //...
}

public class CPDFInfo {
    // Document title.
    private String title;
    // Document author.
    private String author;
    // Document subject.
    private String subject;
    // Name of app that created document.
    private String creator;
    // Name of app that produced PDF data.
    private String producer;
    // Document creation date.
    private String creationDate;
    // Document modification date.
    private String modificationDate;
    // Document keywords.
    private String keywords;
}
```

3.4.4 Extract Images

To extract images from a PDF document, use the function `CPDFPage.extractImages(String filePath)`.

Extracting images from a page is time-consuming, and you are advised to perform this operation asynchronously.

The code below will grab all images from the first page of the given PDF document:

```
File folder = new File(imageFolder);
if(!folder.exists()){
    folder.mkdir();
}
CPDFPage pdfPage = document.pageAtIndex(0);
try {
    pdfPage.extractImages(folder.getParent());
} catch (IOException e) {
    e.printStackTrace();
}
```

3.5 Security

ComPDFKit PDF SDK for Java is used to protect the content of PDF documents from unauthorized access such as copying or printing. It allows developers to encrypt and decrypt PDFs, add passwords, insert watermarks, and more. To control document security in ComPDFKit PDF SDK, security handlers perform user authorization and set various permissions for PDF documents.

3.5.1 Encrypt

ComPDFKit's `CPDFDocument` API can generate a password-protected document. You can use `CPDFDocument` to create a new password-protected PDF document on disk based on a current document. The user password prevents users from viewing the PDF. If you specify it, you also need to specify an owner password.

For example, you can configure a owner password and the "printing" permission when saving a document to make sure that users who don't know that owner password can only print the document, but not modify it.

```
// Open document from file path
CPDFDocument document = new CPDFDocument();
document.open(pdfPath);
// Set user password
document.setUserPassword(user_password);
//set owner password
document.setOwnerPassword(owner_password);
```

```
// Set permission info
CPDFDocumentPermissionInfo info = document.getPermissionsInfo();
info.setAllowsPrinting(true);
info.setAllowsHighQualityPrinting(false);
info.setAllowsCopying(false);
info.setAllowsDocumentChanges(false);
info.setAllowsDocumentAssembly(false);
info.setAllowsCommenting(false);
info.setAllowsFormFieldEntry(false);
document.setPermissionsInfo(info);
// Save document
document.save(outPath,CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveNoIncremental);
```

3.5.2 Decrypt

ComPDFKit fully supports reading secured and encrypted PDF documents.

To check if a document requires a password:

```
// Open document from file path
CPDFDocument document = new CPDFDocument();
CPDFDocument.PDFDocumentError error = document.open(pdfPath, passwod);
if (error == CPDFDocument.PDFDocumentError.PDFDocumentErrorPassword) {
    // Password required
}
```

To read a PDF document with password protection, use the function `CPDFDocument.isEncrypted()`. If the password is correct, this method returns `true`. Once unlocked, you cannot use this function to relock the document.

To remove PDF security, call the `CPDFDocument.save()` method:

```
// Open document from file path
CPDFDocument document = new CPDFDocument();
document.open(pdfPath, passwod);
// Remove password

try {
  document.save(CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveRemoveSecurity);
} catch (CPDFDocumentException e) {
  e.printStackTrace();
}
```

3.5.3 PDF Permission

A PDF file can have two different passwords sets, a permissions or owner password and an open or user password.

A PDF user password is used to secure access to PDF documents and requires the correct password to view the content. It is commonly used to protect confidential reports and financial documents. The PDF reader also displays document rights such as copying and printing permissions when a user password is set. It is different from the owner's password which controls full access to the document, including modifying content and adding comments.

A PDF permission password, also called an owner or master password, protects the permissions of a PDF document. It restricts actions such as making changes or comments and allows control over copying, printing, and modification. Permission passwords ensure integrity and allow management of advanced editing and security settings. They protect the user's copyright and differ from user passwords.

Description and permissions description about PDF user password and permissions password:

- When the document does not have a permission password nor a user password, the permission is `PermissionsNone`.
- When there is a permission password without a user password, the permission is `PermissionsNone` before the permission password is entered, and the correct permission is `PermissionsOwner` after input.
- When there is no permission password and there is a user password, the permission is `PermissionsNone` before the password is entered, and the correct permission is `PermissionsUser` after the password is entered.
- When there is a permission password and an open password, the permission is `PermissionsNone` before any user password is entered, the correct permission is `PermissionsOwner` after entering, and the correct open password permission is `PermissionsUser`.
- When the permission password is the same as the user password, the permission becomes `PermissionsOwner` after entering the password.

If you want to open a document with a user password programmatically, you can use the `CPDFDocument.UnlockWithPassword()` API.

The PDF specification defines the permissions shown below:

- Printing — print the document.
- High-quality printing — print the document in high fidelity.
- Copying — copy content from the document.
- Document changes — modify the document contents except for document attributes.
- Document assembly — insert, delete, and rotate pages.
- Commenting — create or modify document annotations, including form field entries.
- Form field entry — modify form field entries even if you can't edit document annotations.

To access the corresponding permissions, refer to following methods in the `CPDFDocument` class.

```
public CPDFDocumentPermissionInfo getPermissionsInfo() {
    //...
}
```

```
public class CPDFDocumentPermissionInfo {
    /****** Printing the document ******/
    private boolean allowsPrinting;
    /****** The document allows printing in high fidelity ******/
    private boolean allowsHighQualityPrinting;
    /****** Extract content (text, images, etc.) ******/
    private boolean allowsCopying;
    /****** Modify the document contents except for page management (document
attributes) ******/
    private boolean allowsDocumentChanges;
    /****** Page management: insert, delete, and rotate pages ******/
    private boolean allowsDocumentAssembly;
    /****** Create or modify annotations, including form field entries ******/
    private boolean allowsCommenting;
    /****** Modify form field entries, even if allowsCommenting is T_FALSE ******/
    private boolean allowsFormFieldEntry;
}
```

3.5.4 Watermark

A PDF watermark is an element such as a backward layer of transparent text or image added to a PDF document to maintain confidentiality and copyright protection of the document and to highlight information about the company or group to which it belongs. Watermarks can be text, images, shapes, a fixed logo, or dynamically generated text or images.

Adding a watermark to ComPDFKit PDF SDK is very simple and can be done through the API provided by `CPDFWatermark` to add a text watermark or an image watermark. You can set the watermark position, color, transparency, font, size, and other parameters to achieve different needs.

- To get the the `CPDFWatermark` at the specified index in this list, use the function `CPDFDocument.getWatermark(int index)`.
- To add a watermark, use the function `CPDFDocument.createWatermark(CPDFWatermark.Type type)`.
- To remove the watermark, use the function `CPDFWatermark.clear()`.
- To update the watermark, use the function `CPDFWatermark.update()`.

How to generate a PDF with a watermark on all its pages using the `CPDFDocument` API:

```
CPDFWatermark watermark = (CPDFWatermark)
document.createWatermark(CPDFWatermark.Type.WATERMARK_TYPE_TEXT);
if(watermark.getType() == CPDFWatermark.Type.WATERMARK_TYPE_TEXT){
  watermark.setText("Hello World");//The text for the watermark (image watermark does
not work).
  watermark.setFontName("Helvetica");//The text font for the watermark (image watermark
does not work).
  watermark.setTextRGBColor(Color.decode(color).getRGB);//The text color for the
watermark (image watermark does not work).
```

```
  watermark.setFontSize(30);//Default Font : Helvetica 24.
}else{
  watermark.setImage(bitmap, 100, 200);//Set images.
}
watermark.setOpacity(0.5f);//Watermark transparency, the range of 0~1, with the default
of 1.
watermark.setFront(true);//Set watermark to locate in front of the content.
watermark.setVertalign(CPDFWatermark.Vertalign.WATERMARK_VERTALIGN_CENTER);//Vertical
alignment of the watermark.
watermark.setHorizalign(CPDFWatermark.Horizalign.WATERMARK_HORIZALIGN_CENTER);//Horizon
tal alignment of the watermark.
watermark.setVertOffset(0);//The translation relative to the vertical position.
Positive numbers are shifted to the right, negative numbers are shifted to the left.
watermark.setHorizOffset(0);//The translation relative to the horizontal position.
Positive numbers are shifted downwards, negative numbers are shifted upwards.
watermark.setScale(1.3f);//Watermark zoom ratio, with default 1. If it is a image
watermark, 1 means the original size of the image, if it is a text watermark, 1 means
the textFont font size.
watermark.setPages("3,4,5");//Set page range of watermark. Page range, such as:1,2,3 or
1-3.
watermark.setFullScreen(true);//Set tiled watermark for the page (image watermark does
not work).
watermark.setHorizontalSpacing(100);//Set the horizontal spacing for the tiled
watermark.
watermark.setVerticalSpacing(100);//Set the vertical spacing for the tiled watermark.
watermark.update();
watermark.release();
```

3.6 Compare Documents

ComPDFKit provides two methods to compare documents:

- Overlay Comparison
- Content Comparison

3.6.1 Overlay Comparison

Overlay Comparison is used to visually compare pages of different documents. It's helpful for things such as construction plans and detailed drawings, as well as other content that requires precise placement.

This can be done in ComPDFKit using `CPDFCompareDrawings`. The process of preparing documents and comparing them involves a few steps to specify how the comparison should happen. Follow the code below to compare two PDFs with overlay comparison.

```
CPDFDocument baseDocument = new CPDFDocument();
baseDocument.open(rootDir + "/TestFiles/comparefile.pdf");
```

```
CPDFDocument comDocument = new CPDFDocument();
comDocument.open(rootDir + "/TestFiles/comparefile1.pdf");
printDividingLine();

CPDFCompareDrawings compareDrawings = baseDocument.createCompareDrawings(comDocument);
compareDrawings.setBlendMode(4);
compareDrawings.setColorCompare(Color.decode("#FBBDBF").getRGB());
compareDrawings.setColorCompared(Color.decode("#93B9FD").getRGB());
compareDrawings.setFillAlphaCompare(50);
compareDrawings.setFillAlphaCompared(50);
compareDrawings.compareContent();
CPDFDocument document = compareDrawings.getDocGenerate();
document.save(rootDir + "/out/compareTestOverlay.pdf",
CPDFDocument.PDFDocumentSaveType.PDFDocumentSaveNoIncremental);
```

3.6.2 Content Comparison

Quickly pinpoint changes by comparing two versions of a PDF file.

This can be done in ComPDFKit using `CPDFCompare`. The process of preparing documents and comparing them involves a few steps to specify how the comparison should happen. Follow the code below to compare two PDFs with content comparison.

```
CPDFDocument baseDocument = new CPDFDocument();
baseDocument.open(rootDir + "/TestFiles/comparefile.pdf");
CPDFDocument comDocument = new CPDFDocument();
comDocument.open(rootDir + "/TestFiles/comparefile1.pdf");
printDividingLine();

CPDFCompare compare = baseDocument.createCompare(baseDocument, comDocument);
compare.setReplaceResColor(Color.decode("#93B9FD").getRGB());
compare.setInsertResColor(Color.decode("#C0FFEC").getRGB());
CPDFCompareResults result = compare.doCompare(0, 0, CPDFCompare.CompType.COMPAREALL.id,
true);
compare.generateNewDoc(rootDir + "/out/compareTest.pdf");
```

4 Support

4.1 Reporting Problems

Thank you for your interest in ComPDFKit PDF SDK, the only easy-to-use but powerful development solution to integrate high quality PDF rendering capabilities to your applications. If you encounter any technical questions or bug issues when using ComPDFKit PDF SDK for Android, please submit the problem report to the ComPDFKit team. More information as follows would help us to solve your problem:

- ComPDFKit PDF SDK product and version.
- Your operating system and IDE version.
- Detailed descriptions of the problem.
- Any other related information, such as an error screenshot.

4.2 Contact Information

Website:

- Home Page: https://www.compdf.com

Contact ComPDFKit:

- Contact Sales: https://api.compdf.com/contact-us
- Technical Issues Feedback: https://www.compdf.com/support
- Contact Email: support@compdf.com

Thanks,

The ComPDFKit Team