Foxit

DEVELOPER GUIDE

Foxit® PDF SDK

For JAVA

Microsoft® Partner
Gold Independent Software Vendor (ISV)

TABLE OF CONTENTS

1 INTRODUCTION TO FOXIT PDF SDK

Have you ever thought about building your own application that can do everything you want with PDF files? If your answer is "Yes", congratulations! You just found the best solution in the industry that allows you to build stable, secure, efficient and full-featured PDF applications.

1.1 Why Foxit is your choice

Foxit is an Amazon-invested leading software provider of solutions for reading, editing, creating, organizing, and securing PDF documents. Customers choose Foxit products for the following reasons:

- **High performance** – Very fast on PDF parsing, rendering and conversion.
- **Lightweight footprint** – Do not exhaust system resource and deploys quickly.
- **Cross-platform support** – Support Microsoft Windows, Linux etc.
- **Compatibility** – ISO 32000-1/PDF 1.7 standards compliant and compatible with other PDF products.
- **Great value/affordability** – Right features at right price with email and phone support.
- **Security** - Safeguards confidential information.

In addition, Foxit products are fully supported by our dedicated support engineers if support and maintenance are purchased. Updates are released on a regular basis. Developers may focus more on their solution building rather than spending time on PDF specification. Foxit will be the right choice if you need solutions with excellent features and low cost!

1.2 Features

Foxit PDF SDK for Java is a Software Development Kit written in Java. It enables users to develop their applications on desktop platform with Java. It allows developers to incorporate powerful PDF technology to their applications such like view, text search, adding bookmarks in PDF documents, annotating PDF documents and applying pressure sensitive ink (PSI).

Foxit PDF SDK for Java has several main features. They help application developers focus on functions that they really need and reduce the development cost.

Features	
PDF Document	Open and close files, set and get metadata
PDF Page	Parse, render, read and set the properties of a page
Render	Graphics engine created on a bitmap for platform graphics device

PDF Text Page	Text processing in a PDF document
Text Link	Extract URL formatted link
Bookmark	Directly locate and link to point of interest within a document
Image Conversion	Convert between PDF files and images(BMP,TIF,JPX,JPG,PNG), and GIF to PDF
Annotation	Create, edit and remove annotations
Form	Get and set related properties of PDF Form Fields
Reflow	Arrange page content to fit changed page size
Pressure Sensitive Ink	Incorporate pressure sensitive digital ink capabilities into PDF solutions
Watermark	Create, insert and remove watermarks
Security	Support password, certificate, DRM and custom encryption
Signature	Sign a PDF document, verify a signature, add or delete a signature field

1.3 Evaluation

Foxit PDF SDK allows users to download trial version to evaluate SDK. The trial version has no difference from a standard version except for the 30-day limitation trial period and the trail watermarks that will be generated on the PDF pages. After the evaluation period expires, customers should contact Foxit sales team and purchase licenses to continue using Foxit PDF SDK.

1.4 License

Developers should purchase licenses to use Foxit PDF SDK in their solutions. Licenses grant users permissions to release their applications based on PDF SDK libraries. However, users are prohibited to distribute any documents, sample codes, or source codes in the SDK released package to any third party without the permission from Foxit Software Incorporated.

1.5 About this guide

This guide is intended for the developers who need to integrate Foxit PDF SDK for Java into their own applications. It aims at introducing installation package structure on desktop platform with Java, basic knowledge on PDF and the usage of SDK.

2 INTRODUCTION TO PDF

2.1 History of PDF

PDF is a file format used to represent documents in a manner independent of application software, hardware, and operating systems. Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it.

While Adobe Systems made the PDF specification available for free in 1993, PDF remained a proprietary format controlled by Adobe, until July 1, 2008, when it was officially released as an open standard and published by the International Organization for Standardization as ISO 32000-1:2008. In 2008, Adobe published a Public Patent License to ISO 32000-1 granting royalty-free rights for all patents owned by Adobe that are necessary to make, use, sell and distribute PDF compliant implementations.

2.2 PDF Document Structure

A PDF document is composed of one or more pages. Each page has its own specification to indicate its appearance. All the contents in a PDF page, such as text, image, annotation, and form, etc. are represented as PDF objects. A PDF document can be regarded as a hierarchy of objects contained in the body section of a PDF file. Displaying a PDF document in an application involves loading PDF document, parsing PDF objects, retrieving/decoding the page content and displaying/printing it on a device. Editing a PDF document requires parsing the document structure, making changes and reorganizing the PDF objects in a document. These operations could be done by a conforming PDF reader/editor or in your own applications through APIs provided by Foxit.

2.3 PDF Document Features

PDF supports a lot of features to enhance the document capability, such as document encryption, digital signatures, java script actions, form filling, layered content, multimedia support and etc. These features provide users with more flexibility in displaying, exchanging and editing documents. Foxit Java SDK supports most of these PDF features in the ISO standard. Users can use Foxit PDF SDK to fulfill these advanced features in your applications.

3 GETTING STARTED

It is very easy to setup Foxit PDF SDK and see it in action! It takes just a few minutes and we will show you how to use it in desktop platform with Java. The following sections introduce the structure of installation package, how to apply a license, how to run a demo and how to create your own project.

3.1 System Requirements

Windows:

Windows XP, Vista, 7 and 8 (32-bit, 64-bit)

Windows Server 2003, 2008 and 2012 (32-bit and 64-bit)

The release package includes a 32 bit version and native 64 bit version DLL library for windows 32/64, and .Jar SDK library.

Note: it only supports for Windows 8 classic style not for Store App.

Linux:

Linux 32-bit and Linux 64-bit OS

All Linux test cases have been tested on Centos 6.4 32-bit and Centos 6.5 64-bit.

The release package includes 32-bit, 64-bit version Linux libraries (.so files) and .Jar SDK library.

3.2 Windows

3.2.1 What is in the Package

Download Foxit PDF SDK for windows Java package and extract it to a new directory like "foxitpdfsdk_5_3_win_java". The structure of the release package is shown in Figure 3-1. One thing to note that the highlighted rectangle in the figure is the version of the SDK. Here the SDK version is 5.3, so it shows 5_3. Other highlighted rectangles have the same meaning in this guide. This package contains the following folders:

docs:	API references, developer guide
lib:	libraries and license files
samples:	sample projects and demos

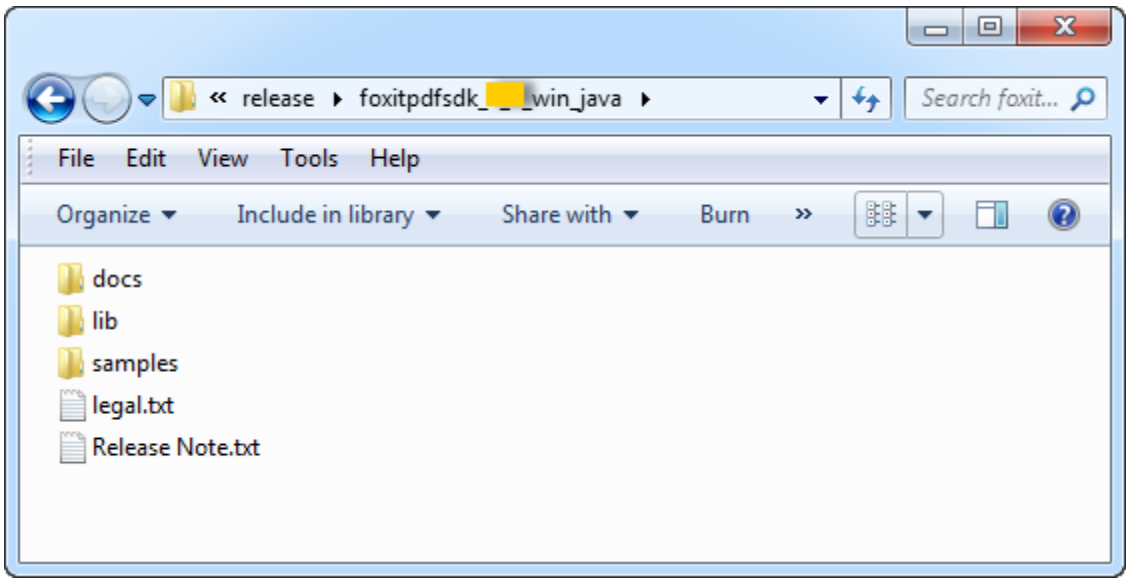

Figure 3-1

Foxit PDF SDK provides “fsdk.jar” file in directory “lib”, it contains 12 packages that are shown in Figure 3-2. In each package, there are java classes listed in Table 3-1.

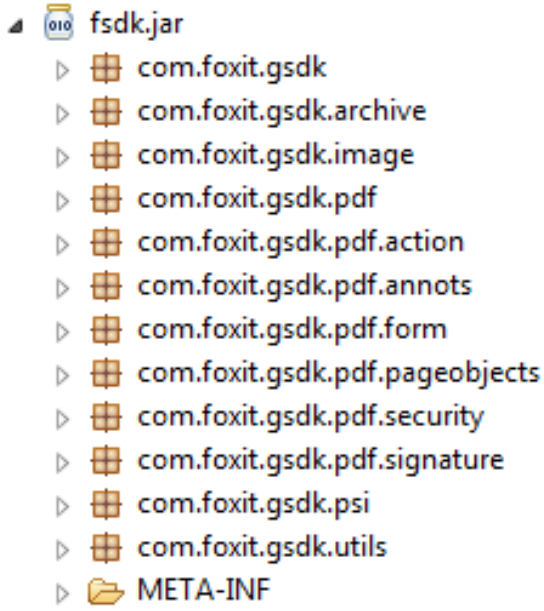

Figure 3-2

Table 3-1

Java Package	Java Class
com.foxit.gsdk	IInvalidate.class PDFException.class PDFLibrary.class
com.foxit.gsdk.archive	Archive.class
com.foxit.gsdk.image	Bitmap.class Image.class ImageFile.class
com.foxit.gsdk.pdf	BookmarkPos.class DefaultAppearance.class Font.class PDFAttachment.class PDFAttachments.class PDFBookmarkIterator.class PDFDocument.class PDFMetadata.class PDFPage.class PDFPath.class PDFReflowPage.class PDFTextLink.class PDFTextPage.class PDFTextSearch.class PDFTextSelection.class PDFWatermark.class Progress.class RenderColorOption.class RenderContext.class Renderer.class RenderOption.class

Java Package	Java Class
com.foxit.gsdk.pdf.action	PDFAction.class PDFDestination.class PDFEmbeddedGotoAction.class PDFEmbeddedGotoActionTarget.class PDFGotoAction.class PDFHideAction.class PDFImportDataAction.class PDFJavascriptAction.class PDFLaunchAction.class PDFNamedAction.class PDFRemoteGotoAction.class PDFResetFormAction.class PDFSubmitFormAction.class PDFURIAction.class
com.foxit.gsdk.pdf.annots	Annot.class Annot3D.class AnnotIconProvider.class Caret.class Circle.class FileAttachment.class FreeText.class Highlight.class Ink.class Line.class Link.class Markup.class Movie.class Polygon.class Polyline.class Popup.class PrinterMark.class PSInk.class RubberStamp.class Screen.class Sound.class Square.class Squiggly.class StrikeOut.class

Java Package	Java Class
	Text.class TextMarkup.class TrapNet.class UnderLine.class Watermark.class Widget.class
com.foxit.gsdk.pdf.form	PDFForm.class PDFFormControl.class PDFFormField.class
com.foxit.gsdk.pdf.pageobjects	ImageObject.class PageObject.class PageObjects.class TextObjects.class TextState.class
com.foxit.gsdk.pdf.security	CertificateEncryptionParams.class CertificateHandler.class CustomEncryptionParams.class EncryptionParams.class FoxitDRMEncryptionParams.class FoxitDRMHandler.class PasswordEncryptionParams.class SecurityHandler.class
com.foxit.gsdk.pdf.signature	Signature.class SignatureHandler.class
com.foxit.gsdk.psi	PSI.class
com.foxit.gsdk.utils	DateTime.class FileHandler.class Matrix.class Rect.class RectF.class Size.class SizeF.class

3.2.2 How to apply a license

It is necessary for applications to initialize and unlock Foxit PDF SDK license before calling any APIs. The function ***unlock*** *(sn, key)* is provided in PDFLibrary.java. An example of applying a license with hardcode

method is shown below. The parameter “sn_xxx” can be found in the “gsdk_sn.txt” (the string after “SN=”) and the “password_xxx” can be found in the “gsdk_key.txt” (the string after “Sign=”).

```
//load the PDF SDK library. Here we assume your system is 64-bit.
static{
	System.load(System.getProperty("user.dir") + "//lib//fsdk_java_win64.dll")
}

PDFLibrary pdfLibrary = PDFLibrary.getInstance();
try {
	pdfLibrary.initialize(30*1024*1024, true);
	pdfLibrary.unlock(“sn_xxx”, “password_xxx”);

} catch (PDFException e) {
	e.printStackTrace();
}
```

3.2.3 How to run a demo

1) **Demo Environment**

Foxit PDF SDK provides useful examples for developers to learn how to call SDK APIs. The followings are the components for the development environments:

- lib/fsdk_java_win64.dll (fsdk_java_win32.dll)– A dynamic link library using Java Native Interface (JNI) to expose native C/C++ functions to the Java project in a cross compilation environment. The advantage of .dll (dynamic link library) is that they are linked during the runtime.

- SDK Library jar file (fsdk.jar) – operates on the Java layer. They provide all the classes and functionalities of our PDF library.

2) **Setting up and running demo project**

Download and install Eclipse IDE (http://www.eclipse.org/) in Windows platform.

In “samples”, there are three demos illustrating how to implement PDF document application with Foxit PDF SDK. The demos are shown in Figure 3-3.

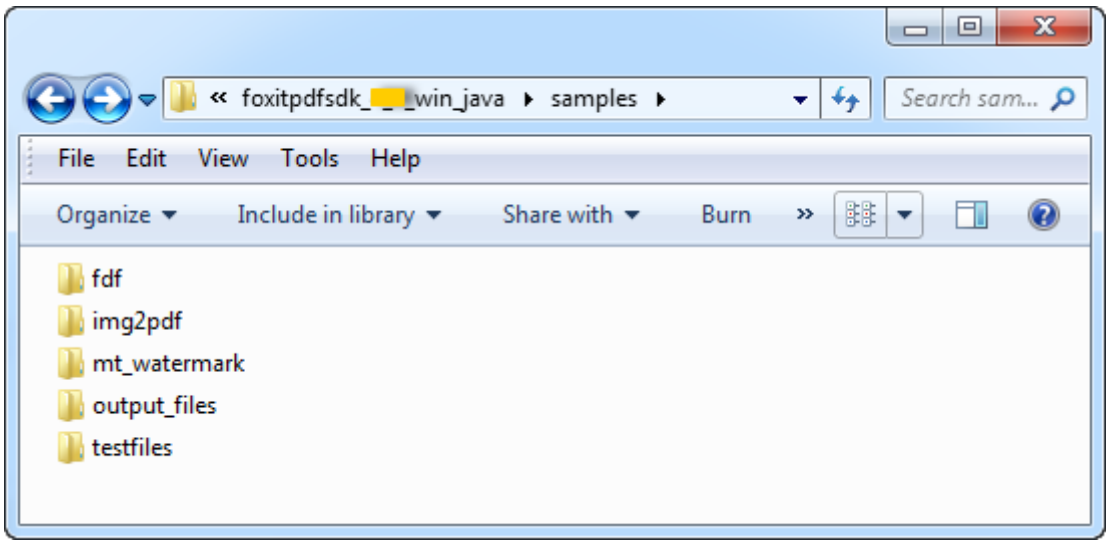

Figure 3-3

To build and run these demos, there are two options: in Eclipse or in command line.

For example, to run the “img2pdf” demo in Eclipse, you can follow the steps below:

a) Launch the Eclipse, import the project into Eclipse following “File->Import-> General/Existing Project into Workspace”, and choose the directory where the demo was extracted by “Browse”. If there is an exclamation mark in the project, please click on “Project->Clean” or right click the project and click on “Refresh”. The directory structure of the demo will be like Figure 3-4.

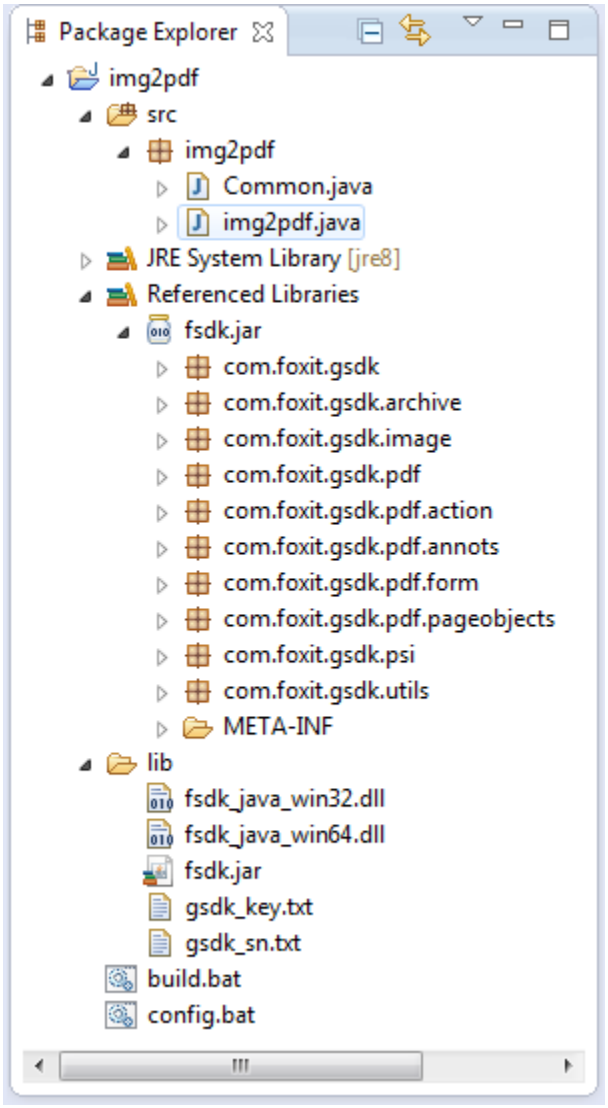

Figure 3-4

One point is important to note that if you check the "Copy projects into workspace" when importing the demo project into Eclipse, you should manually copy the "fsdk.jar", "fsdk_java_win64.dll" and "fsdk_java_win32.dll" files in directory "foxitpdfsdk_5_3_win_java/lib" to "img2pdf/lib" in the workspace.

b) Right-click the demo project in package explorer, then choose "Run As → Java Application" to run the demo. The input files are put in directory "foxitpdfsdk_5_3_win_java/samples/testfiles", and the output files are generated in "foxitpdfsdk_5_3_win_java/samples/output_files/img2pdf".

To run the "img2pdf" demo in command line, start "cmd.exe", navigate to "foxitpdfsdk_5_3_win_java/samples/img2pdf" directory and run "build.bat". The terminal output is

shown in Figure 3-5. Then, you can find the output files in "foxitpdfsdk_5_3_win_java/samples/output_files/img2pdf" directory.

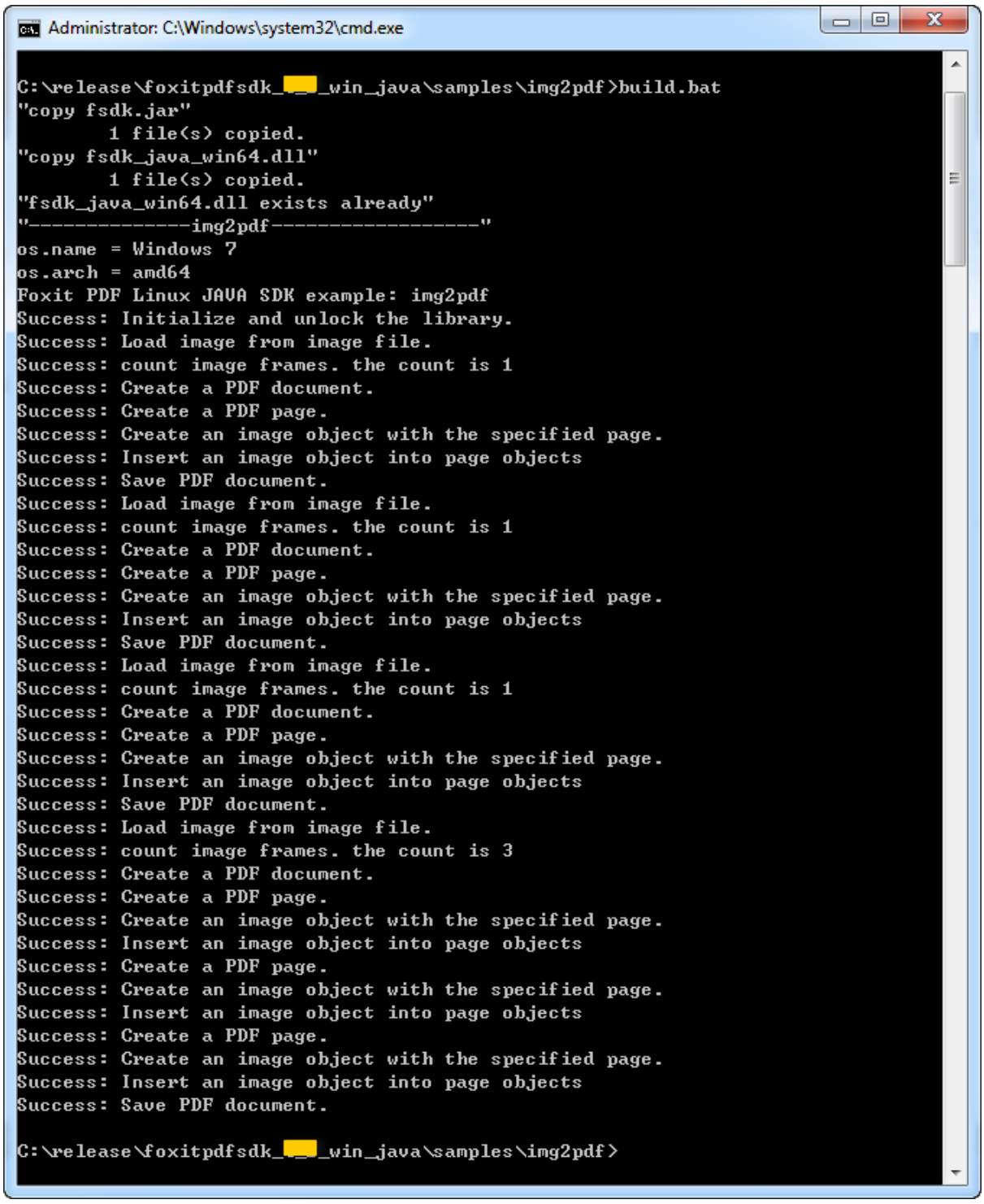

Figure 3-5

3.2.4 How to create your own project

In this section, we will show you how to create your own project by using Foxit PDF SDK APIs. Create a Java project in Eclipse called “test”. Copy “lib” folder from the download package to the project folder, and then refresh the project. The structure of the test project is shown in Figure 3-6.

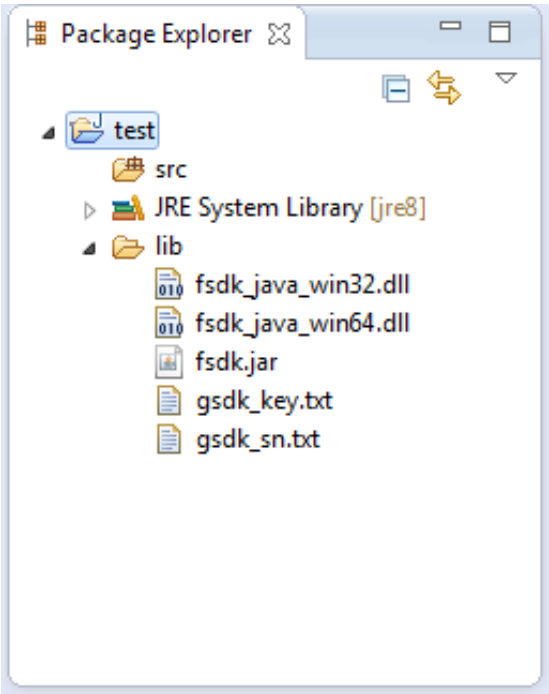

Figure 3-6

To run the test project, follow the steps below:

a) Add the “fsdk.jar” to the project. Right click the test project, select “Build Path→ Configure Build Path→ Libraries→ Add JARs”, and choose the “fsdk.jar” in “test/lib” as shown in Figure 3-7.

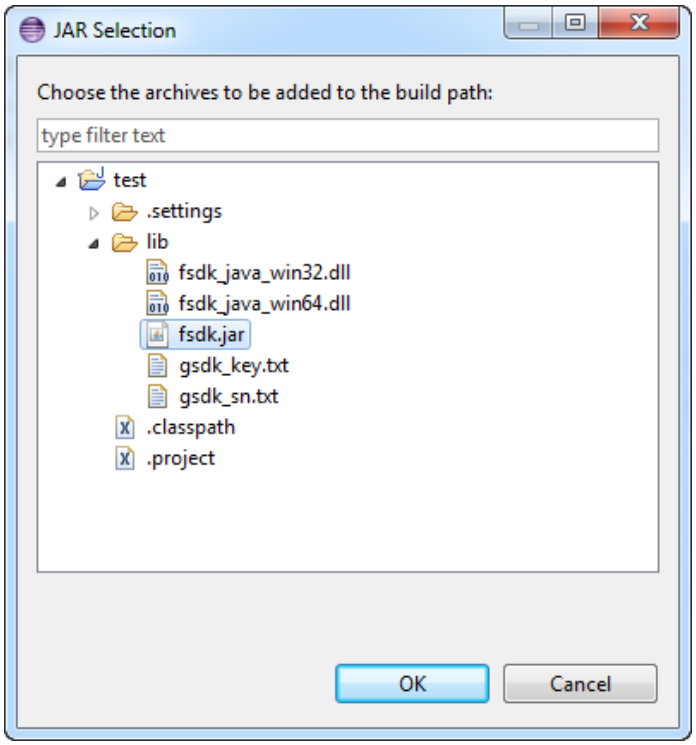

Figure 3-7

b) Create a new class file called Test.java under “test/src/test” directory. The structure of the test project will be like Figure 3-8.

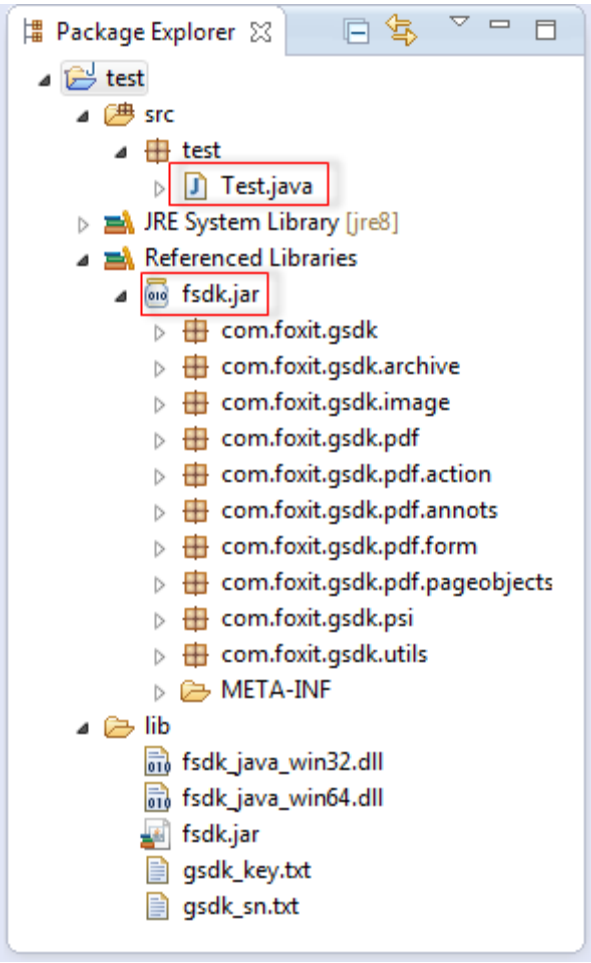

Figure 3-8

c) Open the "Test.java" file, import the classes that you need to use in "fsdk.jar". Here, we just import the classes as follows:

```
import com.foxit.gsdk.PDFException;
import com.foxit.gsdk.PDFLibrary;
```

d) Load the PDF SDK library. The project can automatically load the proper library for your system with the following code. If your system is 64 bit, the libfsdk_java_linux64.so library will be loaded, or loading the libfsdk_java_linux32.so library.

```
static {
	try {
		String arch = System.getProperty("os.arch");
		if (arch.contains("64"))
			System.load(System.getProperty("user.dir")
					+ "//lib//fsdk_java_win64.dll");
		else {
			System.load(System.getProperty("user.dir")
					+ "//lib//fsdk_java_win32.dll");
		}
```

```
        } catch (UnsatisfiedLinkError e) {
                System.out.println("Native code library failed to load.\n" + e);
                System.exit(1);
        }
}
```

e) Construct the code to build a PDF application. The necessary functions and the structure of the code are as follows. Here we do not elaborate details on how to apply a license (initLib() function), which can be referred in section 3.2.2.

```
public void initLib() {
                // The implementation of initiating SDK library manager and applying
license goes here
        }

        public void pdfOperation() {
                // The implementation of pdf operation goes here
        }

        public void release() {
                PDFLibrary pdfLibrary = PDFLibrary.getInstance();
                pdfLibrary.destroy();
        }

        public static void main(String[] args) {
                Test test = new Test();
                test.initLib();
                test.pdfOperation();
                test.release();
                System.out.println("are you ready to go on your application?");
        }
```

f) Build and Run the project. Also, there are two ways to build and run the test project: in Eclipse or in command line.

   i. In Eclipse, please right-click the test project in package explorer, and then choose “Run As → Java Application” to run it. The screenshot of the running result is shown in Figure 3-9.

   ii. In command line, start “cmd.exe”, navigate to “test” folder, input the command “*javac -cp .;src\test;lib\fsdk.jar src\test*.java*” to generate the Test.class file which is placing under “test/src/test” directory. Then, run it by using “*java -cp .;src\;lib\fsdk.jar test.Test*”. This is shown in Figure 3-10.

Now, you are ready to go on your application!

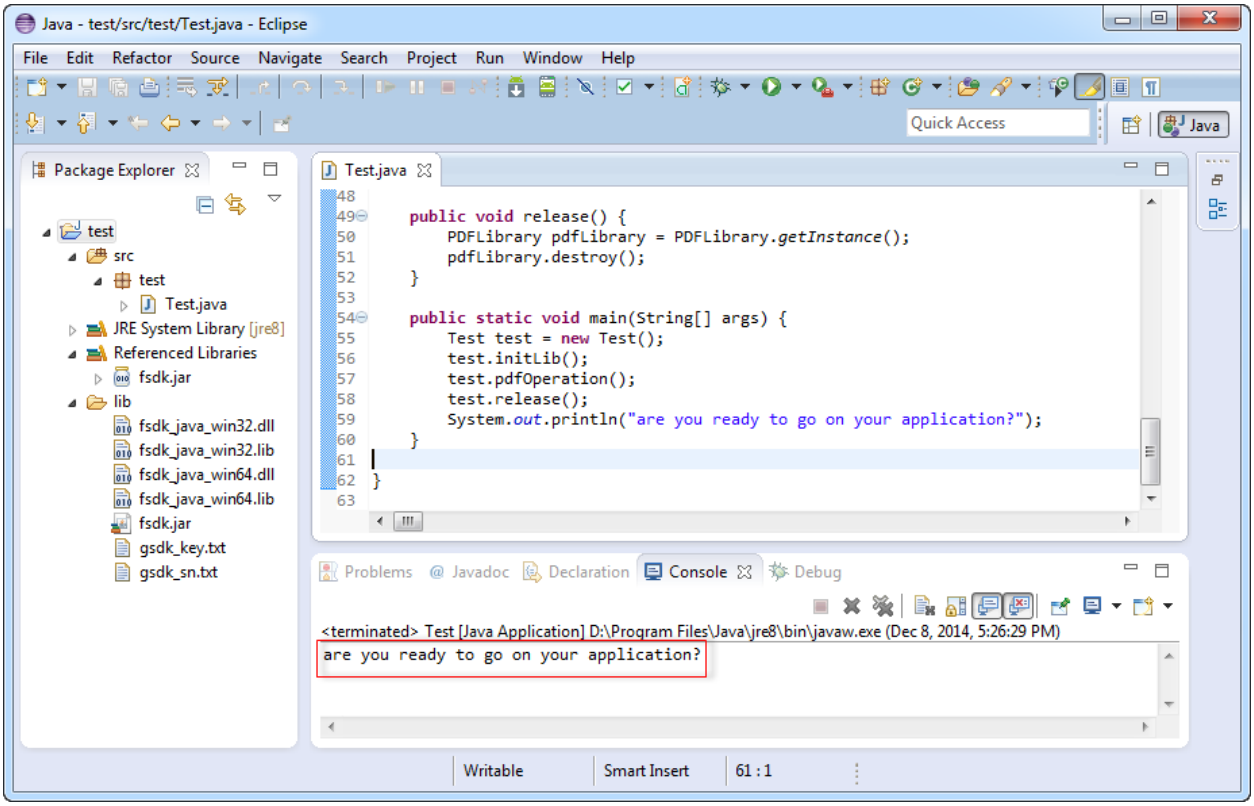

Figure 3-9

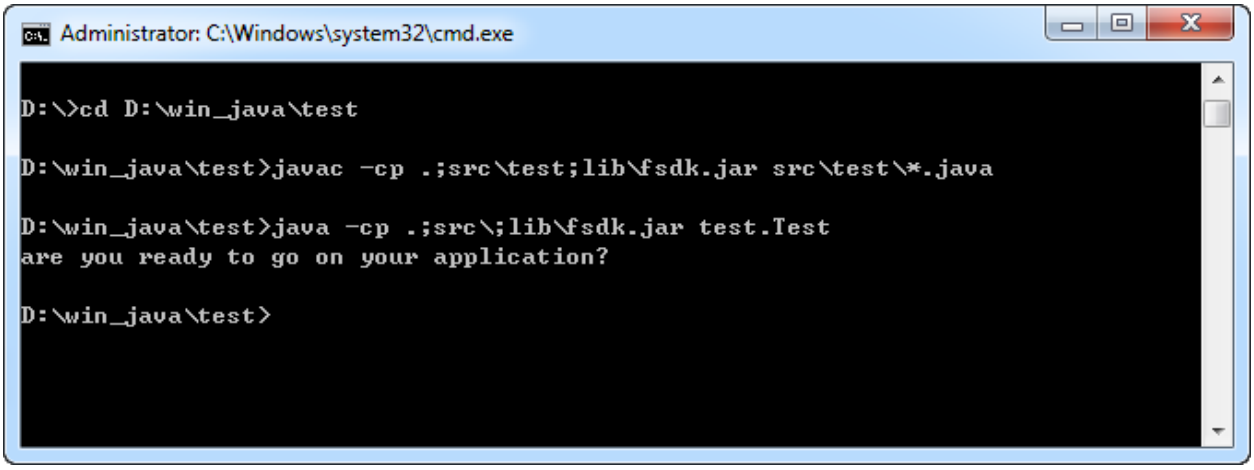

Figure 3-10

3.3 Linux

3.3.1 What is in the Package

Download Foxit PDF SDK for Linux Java package and extract it to a new directory like "foxitpdfsdk_5_3_linux_java". The structure of the release package is shown in Figure 3-11. This package contains the following folders:

docs: API references, developer guide

lib: libraries and license files

samples: sample projects and demos

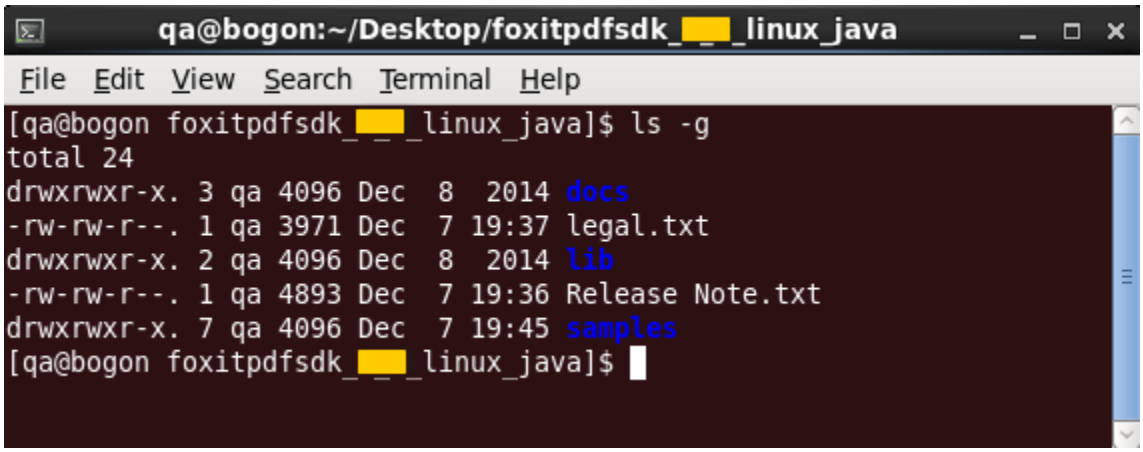

Figure 3-11

Foxit PDF SDK provides "fsdk.jar" file including 10 packages in directory "lib", which can refer to Windows platform in section 3.2.1.

3.3.2 How to apply a license

It is necessary for applications to initialize and unlock Foxit PDF SDK license before calling any APIs. The function ***unlock*** *(sn, key)* is provided in PDFLibrary.java. An example of applying a license with hardcode method is shown below. The parameter "sn_xxx" can be found in the "gsdk_sn.txt" (the string after "SN=") and the "password_xxx" can be found in the "gsdk_key.txt" (the string after "Sign=").

```
//load the PDF SDK library. Here we assume your system is 64-bit.
static{
	System.load(System.getProperty("user.dir") + "/lib/libfsdk_java_linux64.so ");
}

PDFLibrary pdfLibrary = PDFLibrary.getInstance();
try {
	pdfLibrary.initialize(30*1024*1024, true);
	pdfLibrary.unlock("sn_xxx", "password_xxx");

} catch (PDFException e) {
```

```
        e.printStackTrace();
}
```

3.3.3 How to run a demo

1) **Demo Environment**

Foxit PDF SDK provides useful examples for developers to learn how to call SDK APIs. The followings are the components for the development environments:

- lib/libfsdk_java_linux64.so (libfsdk_java_linux32.so)– A dynamic link library using Java Native Interface (JNI) to expose native C/C++ functions to the Java project in a cross compilation environment. The advantage of .so (shared object) is that they are linked during the runtime.
- SDK Library jar file (fsdk.jar) – operates on the Java layer. They provide all the classes and functionalities of our PDF library.

2) **Setting up and running demo project**

Download and install Eclipse IDE (http://www.eclipse.org/) in Linux platform.

In “samples”, there are three demos illustrating how to implement PDF document application with Foxit PDF SDK. The demos are shown in Figure 3-12.

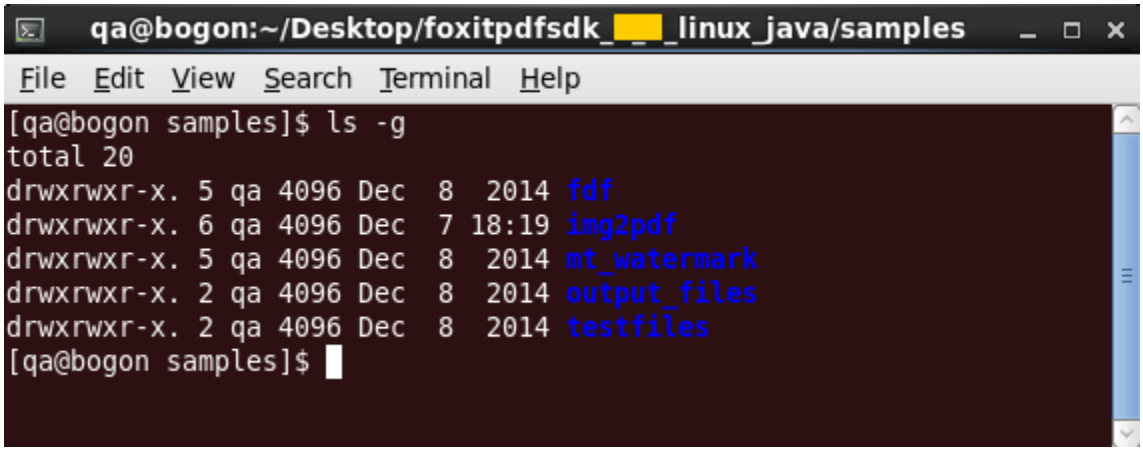

Figure 3-12

To build and run these demos, there are two options: in Eclipse or in terminal.

For example, to run the “img2pdf” demo in Eclipse, you can follow the steps below:

a) Launch the Eclipse, import the project into Eclipse following “File->Import-> General/Existing Project into Workspace”, and choose the directory where the demo was extracted by “Browse”. If there is an exclamation mark in the project, please click on “Project->Clean” or right click the project and click on “Refresh”. The directory structure of the demo will be like Figure 3-13.

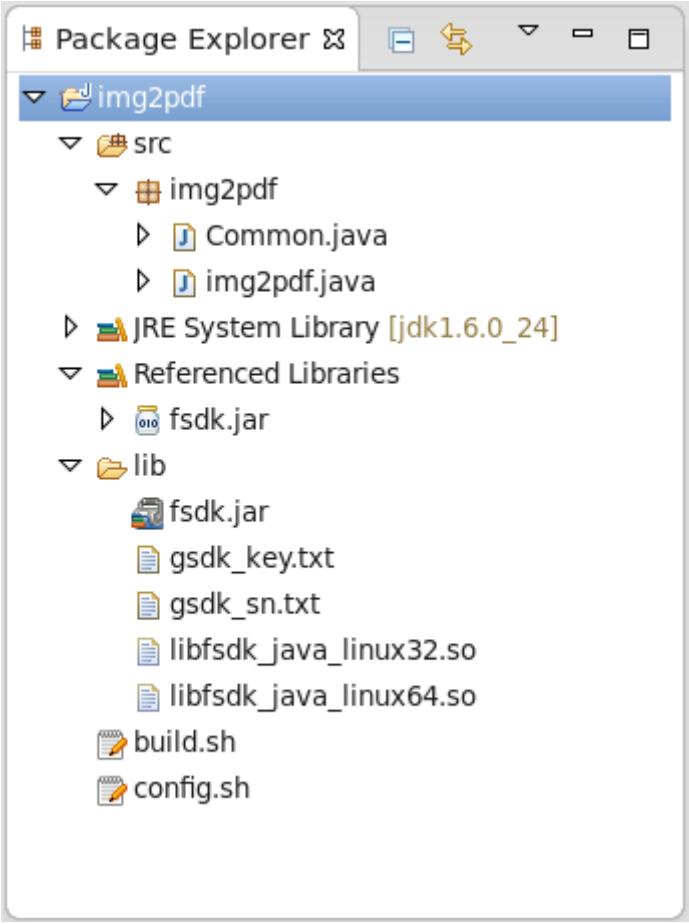

Figure 3-13

Two points are important to note here:

i. If you check the "Copy projects into workspace" when importing the demo project into Eclipse, you should manually copy the "fsdk_linux.jar", "libfsdk_java_linux64.so" and "libfsdk_java_linux32.so" files in directory "foxitpdfsdk_5_3_linux_java/lib" to "img2pdf/lib" in the workspace.

ii. If a permission denied error occurs when importing the demo project into Eclipse, you should modify the demo persission in terminal with command "chmod –R 777 img2pdf" before importing it into Eclipse.

b) Right-click the demo project in package explorer, and then choose "Run As → Java Application" to run the demo. The input files are put in directory "foxitpdfsdk_5_3_linux_java/samples/testfiles", and the output files are generated in "foxitpdfsdk_5_3_linux_java/samples/output_files/img2pdf".

To run the "img2pdf" demo in terminal, navigate to "foxitpdfsdk_5_3_linux_java/samples/img2pdf" directory, open a terminal window, run "sh build.sh" to build and run the demo, which is shown in

Figure 3-14. Then, you can find the output files in "foxitpdfsdk_5_3_linux_java/samples/output_files/img2pdf" directory.

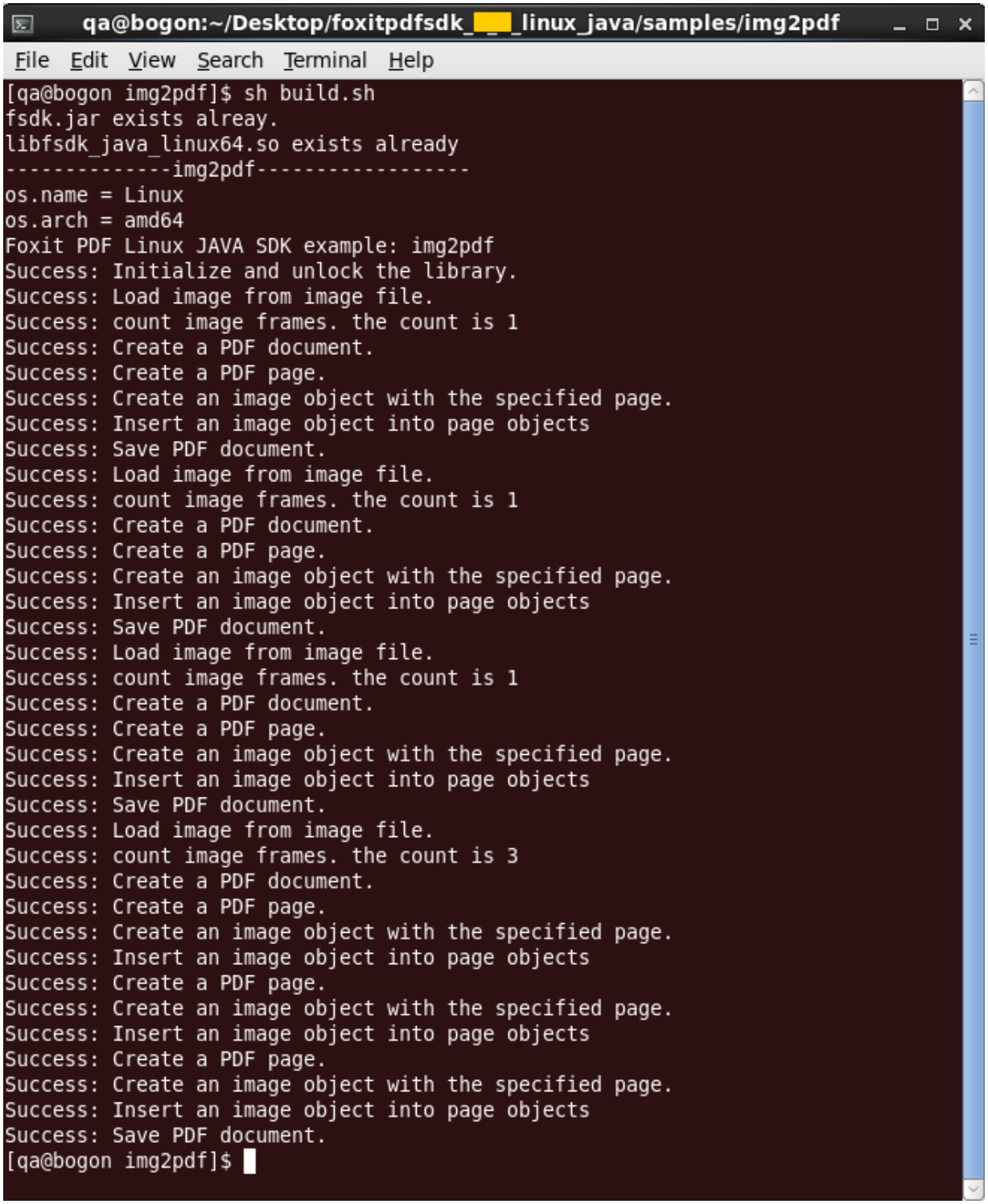

Figure 3-14

3.3.4 How to create your own project

In this section, we will show you how to create your own project by using Foxit PDF SDK APIs. Create a Java project in Eclipse called “test”. Copy “lib” folder from the download package to the project folder, and then refresh the project. The structure of the test project is shown in Figure 3-15.

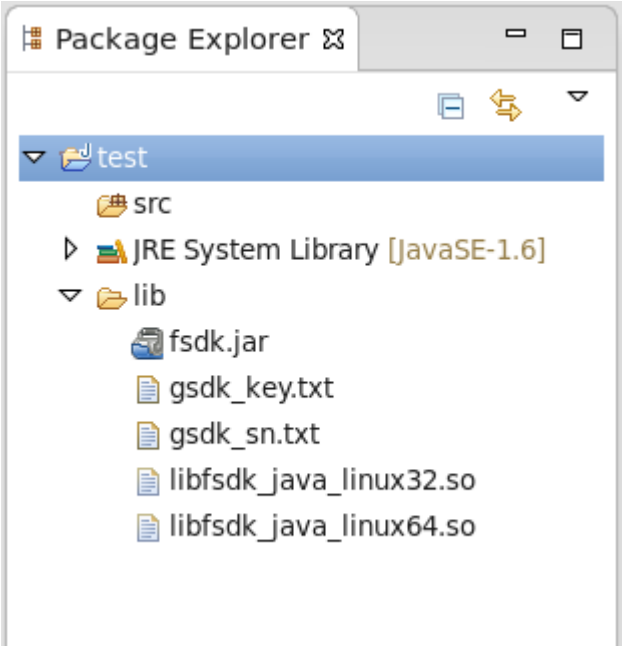

Figure 3-15

To run the test project, follow the steps below:

a) Add the “fsdk.jar” to the project. Right click the test project, select “Build Path→ Configure Build Path→ Libraries→ Add JARs”, and choose the “fsdk.jar” in “test/lib” as shown in Figure 3-16.

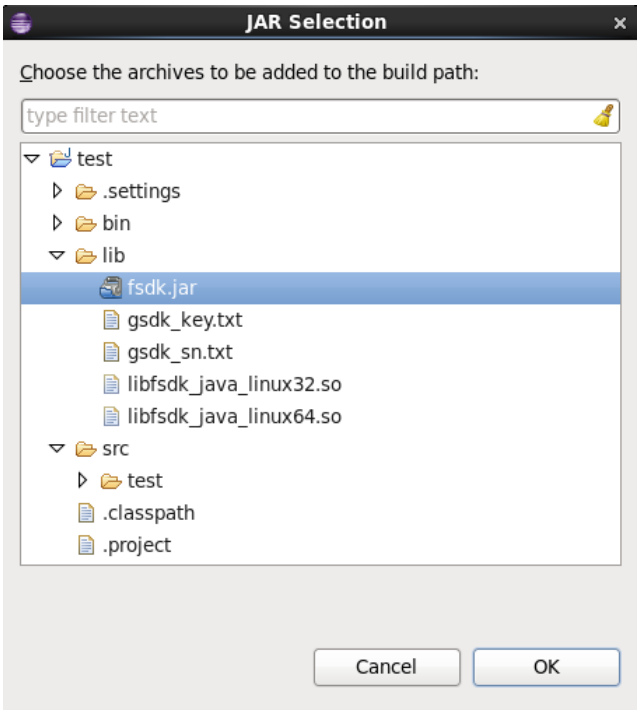

Figure 3-16

b) Create a new class file called Test.java under “test/src/test” directory. The structure of the test project will be like Figure 3-17.

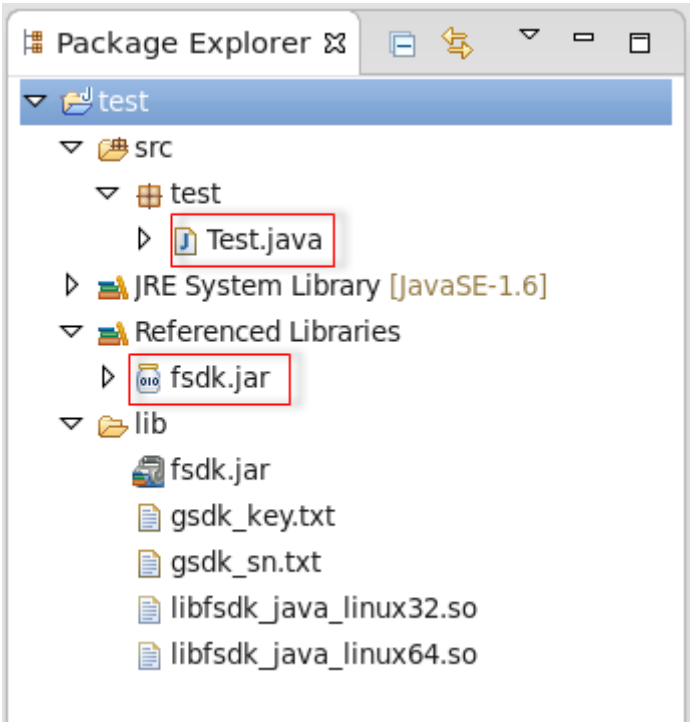

Figure 3-17

c) Open the “Test.java” file, import the classes that you need to use in “fsdk.jar”. Here, we just import the classes as follows:

```
import com.foxit.gsdk.PDFException;
import com.foxit.gsdk.PDFLibrary;
```

d) Load the PDF SDK library. The project can automatically load the proper library for your system with the following code. If your system is 64 bit, the libfsdk_java_linux64.so library will be loaded, or loading the libfsdk_java_linux32.so library.

```
static {
	try {
		String arch = System.getProperty("os.arch");
		if (arch.contains("64"))
			System.load(System.getProperty("user.dir")
					+ "/lib/libfsdk_java_linux64.so");
		else {
			System.load(System.getProperty("user.dir")
					+ "/lib/libfsdk_java_linux32.so");
		}
	} catch (UnsatisfiedLinkError e) {
		System.out.println("Native code library failed to load.\n" + e);
		System.exit(1);
	}
}
```

e) Construct the code to build a PDF application. The necessary functions and the structure of the code are as follows. Here we do not elaborate details on how to apply a license (initLib() function), which can be referred in section 3.2.2.

```
public void initLib() {
		// The implementation of initiating SDK library manager and applying
license goes here
	}

	public void pdfOperation() {
		// The implementation of pdf operation goes here
	}

	public void release() {
		PDFLibrary pdfLibrary = PDFLibrary.getInstance();
		pdfLibrary.destroy();
	}

	public static void main(String[] args) {
		Test test = new Test();
		test.initLib();
		test.pdfOperation();
		test.release();
		System.out.println("are you ready to go on your application?");
	}
```

f) Build and Run the project. Also, there are two ways to build and run the test project: in Eclipse or in terminal.

i. In Eclipse, please right-click the test project in package explorer, and then choose “Run As → Java Application” to run it. The screenshot of the running result is shown in Figure 3-18.

ii. In terminal, navigate to “test” folder, right click and select “Open in Terminal”, input the command “*javac –cp .:src/test:lib/fsdk.jar src/test/*.java*” to generate the Test.class file which is placing under “test/src/test” directory. Then, run it by using “*java –cp .:src/:lib/fsdk.jar test.Test*”. This is shown in Figure 3-19.

Now, you are ready to go on your application!

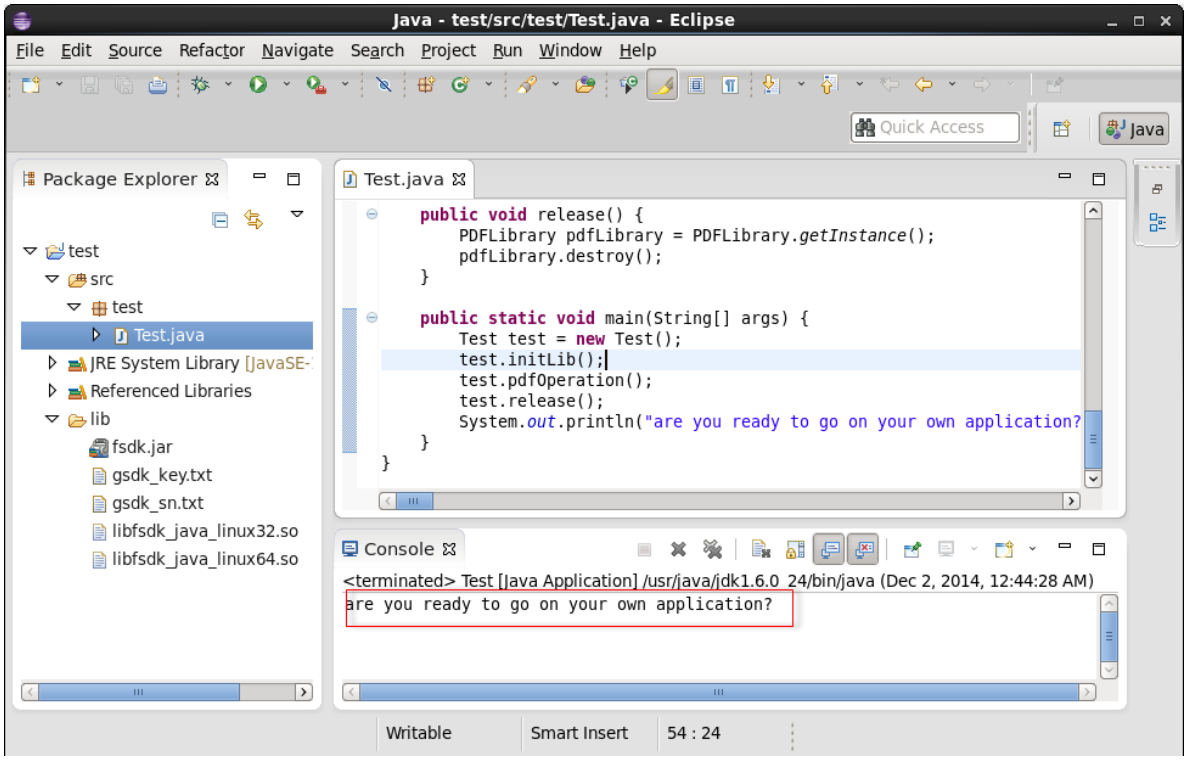

Figure 3-18

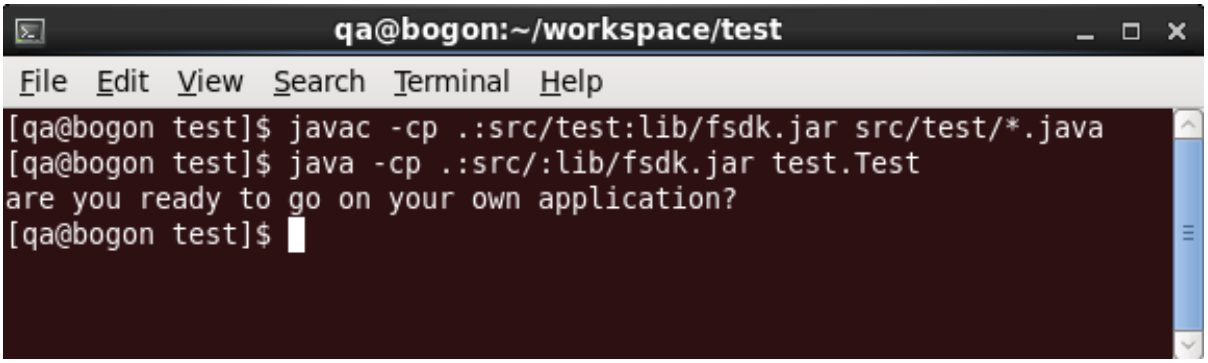

Figure 3-19

4 WORKING WITH SDK API

In this section, we will introduce a set of major features and list some examples for each feature to show you how to integrate powerful PDF capabilities with your applications on Android platform. You can refer to the API reference [2] to get more details about the APIs used in all of the examples.

4.1 Apply a License

It is necessary for applications to initialize and unlock Foxit PDF SDK license before calling any APIs. The function ***unlock*** *(sn, key)* is provided in PDFLibrary.java to unlock Foxit PDF SDK. A license should be purchased for the application and pass unlock key and code to get proper supports. Two examples of applying a license with hardcode method on Windows and Linux platforms are shown below.

Note *The parameter "sn_xxx" can be found in the "**gsdk_sn.txt**" (the string after "SN=") and the "password_xxx" can be found in the "**gsdk_key.txt**" (the string after "Sign=").*

Example:

4.1.1 How to apply a license on Windows platform

```
//load the PDF SDK library. Here we assume your system is 64-bit.
static{
	System.load(System.getProperty("user.dir") + "//lib//fsdk_java_win64.dll")
}

PDFLibrary pdfLibrary = PDFLibrary.getInstance();
try {
	pdfLibrary.initialize(30*1024*1024, true);
	pdfLibrary.unlock("sn_xxx", "password_xxx");

} catch (PDFException e) {
	e.printStackTrace();
}
```

4.1.2 How to apply a license on Linux platform

```
//load the PDF SDK library. Here we assume your system is 64-bit.
static{
	System.load(System.getProperty("user.dir") + "/lib/libfsdk_java_linux64.so ");
}

PDFLibrary pdfLibrary = PDFLibrary.getInstance();
try {
	pdfLibrary.initialize(30*1024*1024, true);
	pdfLibrary.unlock("sn_xxx", "password_xxx");

} catch (PDFException e) {
	e.printStackTrace();
}
```

4.2 File

PDF file access (I/O) is managed by file handler **FileHandler**. Developers can determine whether to implement reading actions or writing actions in the **FileHandler** handle based on application intentions, but please note that the reading and writing actions cannot be done at the same time. Foxit PDF SDK provides the capability of reading file path from a file or memory**.**

Example:

4.2.1 How to create a FileHandler object

```
try {
FileHandler fileHandler = FileHandler.create(filename, fileMode);
	PDFDocument pdfDocument = PDFDocument.open(fileHandler, null);
}
catch (PDFException e) {
	// TODO Auto-generated catch block
		e.printStackTrace();
}
```

4.3 Document

PDF document is represented by **PDFDocument** handle object. Document level APIs provide functions to open and close files, get page, metadata and etc. A **PDFDocument** handle should be initialized by calling **PDFDocument.open(FileHandler, byte[])** or **PDFDocument.open(FileHandler, byte[], int)** to allow page or deeper level API to work.

Example:

4.3.1 How to get the first page of a PDF

```
PDFDocument pdfDocument = null;
try {
	//Assuming a FileHandler has been created.
	pdfDocument = PDFDocument.open(fileHanlder, null);

	int count = pdfDocument.countPages();
	PDFPage page = pdfDocument.getPage(0);
	pdfDocument.closePage(page);
	pdfDocument.close();

}
catch (PDFException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

4.3.2 How to save PDF to a file

```
PDFDocument pdfDocument = null;
Progress progress = null;

try {
	//Assuming a FileHandler has been created.
	pdfDocument = PDFDocument.open(fileHandler, null);
	FileHandler saveFile = FileHandler.create("save.pdf", FileHandler.FILEMODE_TRUNCATE);
	progress = pdfDocument.startSaveToFile(saveFile, PDFDocument.SAVEFLAG_INCREMENTAL);
	if (progress != null)
	{
		int ret = Progress.TOBECONTINUED;
		while (ret == Progress.TOBECONTINUED)
		{
			ret = progress.continueProgress(30);
		}
	}
	progress.release();
	pdfDocument.close();
}
catch (PDFException e) {
	e.printStackTrace();
}
```

4.4 Attachment

In Foxit PDF SDK, attachments are only referred to attachments of documents rather than file attachment annotation, which allow whole files to be encapsulated in a document, much like email attachments. PDF SDK provides applications APIs to access attachments such as loading attachments, getting attachments, inserting/removing attachments, and accessing properties of attachments.

Example:

4.4.1 How to insert an attachment file into a PDF

```
//Assuming PDFDocument document/newDoc has been loaded.
//Assuming returning values will be checked in active source code.
...

try {
     PDFAttachments attachs = document.loadAttachments();
     int count = attachs.countAttachment();
     PDFAttachment attach = PDFAttachment.create(newDoc);
     attachments.insertAttachment(index, attach);

     FileHandler handler = FileHandler.create(filename, fileMode);
     Attach.setFile(handler);

}
catch (PDFException e) {
```

```
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4.4.2 How to remove a specific attachment of a PDF

```
//Assuming PDFDocument document/newDoc has been loaded.
//Assuming returning values will be checked in active source code.
...

try {
        PDFAttachments attachs = document.loadAttachments();
        PDFAttachment attachment = attachs.getAttachment(index);
        attachs.removeAttachment(attachment);
        ...
}
catch (PDFException e) {
        e.printStackTrace();
}
```

4.4.3 How to change the properties of an attachment

```
//Assuming PDFDocument document/newDoc has been loaded.
//Assuming returning values will be checked in active source code.
...

try {
        PDFAttachments attachs = document.loadAttachments();
        PDFAttachment attachment = attachs.getAttachment(index);
        attachment.setDescription(description);
        attachment.setModifiedDateTime(date);
        ...
}
catch (PDFException e) {
        e.printStackTrace();
}
```

4.5 Page

PDF page is the basic and important component of PDF Document. It is represented by **PDFPage** handle object. **PDFPage** object is created by a **PDFDocument** object using **PDFDocument.getPage(int)** or **PDFDocument.createPage(int)**. Page level APIs provide functions to parse, render, edit (includes creating, deleting and flattening) a page, and read or set the properties of a page. A PDF page needs to be parsed before it is rendered or processed for text extraction.

Example:

4.5.1 How to create a PDF page

```
PDFDocument pdfDocument = null;
PDFPage page = null;
```

```
try
{
	pdfDocument = PDFDocument.open(fileHandler, null);
	int cnt = pdfDocument.countPages();
	page = pdfDocument.createPage(0);
	Assert.assertEquals(cnt + 1, pdfDocument.countPages());
	pdfDocument.closePage(page);
}
catch (PDFException e)
{
	e.printStackTrace();
}
pdfDocument.close();
```

4.5.2 How to get page size

```
PDFDocument pdfDocument = null;
PDFPage page = null;
try
{
	pdfDocument = PDFDocument.open(fileHandler, null);
	page = pdfDocument.getPage(0);
	SizeF pageSize = page.getSize();
	...
}
catch (PDFException e)
{
	e.printStackTrace();
}
```

4.5.3 How to delete a PDF page

```
PDFDocument pdfDocument = null;
PDFPage page = null;
try
{
	pdfDocument = PDFDocument.open(fileHandler, null);
	page = pdfDocument.getPage(0);
	pdfDocument.deletePage(page);
	pdfDocument.close();
}
catch (PDFException e)
{
	e.printStackTrace();
}
```

4.5.4 How to flatten a PDF page

```
PDFDocument pdfDocument = null;
PDFPage page = null;
try
{
	pdfDocument = PDFDocument.open(fileHandler, null);
```

```
        page = pdfDocument.getPage(0);
        page.flatten(FLATTENFLAG_DISPLAY);
        ...
}
catch (PDFException e)
{
        e.printStackTrace();
}
```

4.5.5 How to calculate bounding box of page contents

```
PDFDocument pdfDocument = null;
PDFPage page = null;
try
{
        pdfDocument = PDFDocument.open(fileHandler, null);
        page = pdfDocument.getPage(0);
        RectF contentBoxRect = page.calcContentBBox(PDFPage.MARGIN_CONTENTSBBOX);
        ...
}
catch (PDFException e)
{
        e.printStackTrace();
}
```

4.6 Render

PDF rendering is realized through the Foxit renderer, a graphic engine that is created on a bitmap. Rendering process requires a renderer and render context. Renderer on bitmap is created by a renderer object using **Renderer.create(Bitmap)**. The rendering settings (or render context) are set in **RenderContext** object.

Example:

4.6.1 How to parse a PDF page

```
PDFDocument pdfDocument = null;
PDFPage page = null;
try {
        pdfDocument = PDFDocument.open(fileHandler, null);
        page = pdfDocument.getPage(0);
        Progress parserProgress = null;

        if(page != null)
        parserProgress = page.startParse(PDFPage.PARSEFLAG_NORMAL);

        int ret_prog = Progress.TOBECONTINUED;
        while (ret_prog == Progress.TOBECONTINUED){
                ret_prog = parserProgress.continueProgress(30);
        }
        parserProgress.release();
```

```
} catch (com.foxit.gsdk.PDFException e) {
	e.printStackTrace();
}
```

4.6.2 How to render a PDF page by drawing bitmaps

```
Matrix matrix = new Matrix();
SizeF pagesize = null;
try {
	pagesize = page.getSize();
	Bitmap.Config conf = Bitmap.Config.ARGB_8888;
	Bitmap bmp = Bitmap.createBitmap((int)pagesize.getWidth(), (int)pagesize.getHeight(),
conf);

	Renderer renderer = null;
	renderer = Renderer.create(bmp);
      matrix = page.getDisplayMatrix(0, 0,(int)pagesize.getWidth(), (int)pagesize.getHeight(),
0);

	//Render PDF pages by drawing bitmaps
	RenderContext renderContext = null;
	renderContext = RenderContext.create();
	renderContext.setMatrix(matrix);
	Progress renderProgress = page.startRender(renderContext, renderer, 0);
	if(renderProgress != null)
	{
		int r = Progress.TOBECONTINUED;
		while (r == Progress.TOBECONTINUED)
		{
			r = renderProgress.continueProgress(30);
		}
	}
	renderProgress.release();
	renderContext.release();
	render.release();

} catch (com.foxit.gsdk.PDFException e) {
	e.printStackTrace();
}
```

4.7 Text Page

Foxit PDF SDK provides APIs to extract, select, search and retrieve text in PDF documents. PDF text contents are stored in **PDFTextPage** objects which are related to a specific page. Prior to text processing, user should first call **PDFTextPage.create(PDFPage)** or **PDFTextPage.create(PDFPage, int)** to get the textpage object.

Example:

4.7.1 How to select text in a PDF page

```
PDFDocument pdfDocument = null;
PDFPage page = null;
PDFTextPage textPage;
try {
	pdfDocument = PDFDocument.open(fileHandler, null);
	page = pdfDocument.getPage(0);
	textPage = PDFTextPage.create(page);
	PDFTextSelection selection = textPage.selectByRange(0, -1);
	final String s = selection.getChars();
	selection.release();
	textPage.release();
	pdfDocument.closePage(page);
	pdfDocument.close();

}
catch (PDFException e) {
	e.printStackTrace();
}
```

4.7.2 How to search text in a PDF page

```
PDFTextSearch search = null;
PDFTextPage textPage = PDFTextPage.create(pdfpage);
try {
	//whole word is compared with no case sensitive
	search = textPage.startSearch("foxit", PDFTextSearch.FLAG_MATCHWHOLEWORD, 0);
	boolean next = search.findNext();
	//boolean next = mSearch.findPrev();
	if(!next) return true;

	//A match is found here
	PDFTextSelection select = search.getSelection();
	int rectnum = select.countPieces();
}
catch (com.foxit.gsdk.PDFException e) {
	e.printStackTrace();
}
```

4.7.3 How to extract text from a PDF

```
PDFTextSearch search = null;
try {
	PDFTextPage textPage = PDFTextPage.create(pdfpage);
	int count = textPage.countChars();
	String text = textPage.getChars(0, count);
	...
}
catch (com.foxit.gsdk.PDFException e) {
	e.printStackTrace();
}
```

4.8 Text Link

Foxit PDF SDK provides APIs to retrieve, extract and enumerate text hyperlinks in a PDF document in which the hyperlinks are the same with common texts, and then get the extracted results as text selections. Prior to text link processing, user should first call **PDFTextPage.extractLinks()** to get the textlink object.

Example:

4.8.1 How to get the first URL formatted texts in a PDF page

```
PDFDocument pdfDocument = null;
PDFPage page = null;
PDFTextPage textPage;
try {
     pdfDocument = PDFDocument.open(fileHandler, null);
     page = pdfDocument.getPage(0);
       textPage = PDFTextPage.create(page);

       PDFTextLink testlink = textPage.extractLinks();
       int count = testlink.countLinks();
       if(count>0)
       {
              String linkURL = testlink.getLink(0);
              ...
       }
       testlink.release();
}
catch (PDFException e) {
       e.printStackTrace();
}
```

4.9 Form

Foxit PDF SDK provides APIs to view and edit form field programmatically. Form fields are commonly used in PDF documents to gather data. **PDFForm.exportToFDF(FileHandler)** can export data in a PDF document to an FDF (Forms Data Format) document, from where data can be extracted for further use. **PDFForm.importFromFDF(FileHandler)** allows user to import FDF documents to the original PDF to display the form data.

Example:

4.9.1 How to load the forms in a PDF

```
try
{
       //Assuming PDFDocument pdfDoc has been loaded.
    PDFForm pdfForm = pdfDoc.loadForm();
```

```
    ...
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.9.2 How to count form fields and get the properties

```
try
{
       //Assuming PDFDocument pdfDoc has been loaded.
    PDFForm pdfForm = pdfDoc.loadForm();
    int count = pdfForm.countFields(null);
    int nAliment = 0;
    for (int i = 0; i < count; i++)
    {
       PDFFormField formField = pdfForm.getField(null, i);
              if (PDFFormField.TYPE_CHECKBOX == formField.getType())
              {
                            ...
                     }
                     nAliment = formField.getAlignment();
                     ...
    }
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.9.3 How to export the form data in a PDF to a FDF file

```
try
{
       //Assuming PDFDocument pdfDoc has been loaded.
    PDFForm pdfForm = pdfDoc.loadForm();
    FileHandler  handler = FileHandler.create("export.fdf", FileHandler.FILEMODE_TRUNCATE);
    pdfForm.exportToFDF(handler);
    handler.release();
    ...
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.9.4 How to import form data from a FDF file

```
try
{
       //Assuming PDFDocument pdfDoc has been loaded.
    PDFForm pdfForm = pdfDoc.loadForm();
```

```
    FileHandler  handler = FileHandler.create("import.fdf", FileHandler.FILEMODE_READONLY);
    pdfForm.importFromFDF(handler);
    handler.release();
    ...
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.9.5 How to get and set the properties of form fileds

```
try
{
       //Assuming PDFDocument pdfDoc has been loaded.
    PDFForm pdfForm = pdfDoc.loadForm();
    PDFFormField field = pdfForm.getField(null, 0);
    field.setValue("prop")
    String value = field.getValue()
    ...
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.10 Annotations

An annotation associates an object such as note, line, and highlight with a location on a page of a PDF document. It provides a way to interact with users by means of the mouse and keyboard. PDF includes a wide variety of standard annotation types as listed in Table 4-1. Among these annotation types, many of them are defined as markup annotations for they are used primarily to mark up PDF documents. These annotations have text that appears as part of the annotation and may be displayed in other ways by a conforming reader, such as in a Comments pane. The 'Markup' column in Table 4-1 shows whether an annotation is a markup annotation.

Foxit PDF SDK supports most annotation types defined in PDF reference [1]. PDF SDK provides APIs of annotation creation, properties access and modification, appearance setting and drawing.

Table 4-1

Annotation type	Description	Markup	Supported by SDK
Text(Note)	Text annotation	Yes	Yes
Link	Link Annotations	No	Yes
FreeText(TypeWritter)	Free text annotation	Yes	Yes
Line	Line annotation	Yes	Yes
Square	Square annotation	Yes	Yes

Circle	Circle annotation	Yes	Yes
Polygon	Polygon annotation	Yes	Yes
PolyLine	PolyLine annotation	Yes	Yes
Highlight	Highlight annotation	Yes	Yes
Underline	Underline annotation	Yes	Yes
Squiggly	Squiggly annotation	Yes	Yes
StrikeOut	StrikeOut annotation	Yes	Yes
Stamp	Stamp annotation	Yes	Yes
Caret	Caret annotation	Yes	Yes
Ink(pencil)	Ink annotation	Yes	Yes
Popup	Popup annotation	Yes	Yes
File Attachment	FileAttachment annotation	Yes	Yes
Sound	Sound annotation	Yes	No
Movie	Movie annotation	No	No
Widget*	Widget annotation	No	Yes
Screen	Screen annotation	Yes	No
PrinterMark	PrinterMark annotation	No	No
TrapNet	Trap network annotation	No	No
Watermark*	Watermark annotation	Yes	No
3D	3D annotation	Yes	No

Note:

1. The annotation types of widget and watermark are special. They aren't supported in the module of 'Annotation'. The type of widget is only used in the module of 'form filler' and the type of watermark only in the module of 'watermark'.

2. Foxit SDK supports a customized annotation type called PSI (pressure sensitive ink) annotation that is not described in PDF reference [1]. Usually, PSI is for handwriting features and Foxit SDK treats it as PSI annotation so that it can be handled by other PDF products.

Example:

4.10.1 How to add a link annotation to another page in the same PDF

```
try {
	//The function of load Annots shall be called before any operations on annotations
	pdfPage.loadAnnot();

	//Prepare the rectangle object of annotation bounding box, in PDF page coordination.
	RectF rect = {0, 100, 100, 0};
	//Prepare the string object of the annotation filter.
	String filter = "link";
	//Add an annotation to a specific index with specific filter.
```

```
        Annot annot = pdfPage.addAnnot(rect, Annot.TYPE_LINK, filter, 0);

        Link link = (Link)annot;
        //Set the stroke color and opacity of annotation.
        link.setBorderColor(0x0000FF00);
        link.setOpacity(0.55);
        //Add a url action to link annotation
        PDFURIAction action = (PDFURIAction) PDFAction.createURIAction(url, false);
        link.insertAction(Link.TRIGGER_ANNOT_MU, 0, action);
        ...
}
catch (PDFException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4.10.2 How to add a highlight annotation to a page and set the related annotation properties

```
try {
//The function of load Annots shall be called before any operations on annotations
pdfPage.loadAnnot();

//Prepare the rectangle object of annotation bounding box, in PDF page coordination.
RectF rect = {0, 100, 100, 0};
//Prepare the string object of the annotation filter.
String annotType = “Highlight”;
//Add an annotation to a specific index with specific filter.
Annot annot = pdfPage.addAnnot(rect, annotType, annotType, 1);
//Set the quadrilaterals points of annotation.
QuadpointsF quadPoints = new QuadpointsF();
quadPoints.x1 = 0;
quadPoints.y1 = 0;
quadPoints.x2 = 100;
quadPoints.y2 = 0;
quadPoints.x3 = 0;
quadPoints.y3 = 50;
quadPoints.x4 = 100;
quadPoints.y4 = 50;

Highlight highlight = (Highlight)annot;
highlight.setQuadPoints(quadPoints);
//Set the stroke color and opacity of annotation.
Highlight.setBorderColor(0x0000FF00);
highlight.setOpacity(0.55);

}
catch (PDFException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4.11 Image Conversion

Foxit PDF SDK provides APIs for conversion between PDF files and images. Applications could easily fulfill functionalities like image creation and image conversion which supports the following image formats: BMP, TIFF, PNG, JPX, JPEG, and GIF. Foxit PDF SDK can make the conversion between PDF files and the supported image formats except for GIF. It only supports converting GIF images to PDF files.

Example:

4.11.1 How to convert PDF pages to bitmap files

```
//if file and password are ready for use
PDFDocument document = PDFDocument.open(fileHandler, password);
...
int count = document.countPages();
...
PDFPage page = null;
for (int i=0; i< count; i++)
{
	page = document.getPage(i);
	Progress progress = page.startParse(PDFPage.RENDERFLAG_NORMAL);
	if(progress != null)
	{
		int ret = Progress.TOBECONTINUED;
		while (ret == Progress.TOBECONTINUED)
		{
			ret = progress.continueProgress(30);
		}
	}
	progress.release();

	SizeF pageSize = page.getSize();
	Matrix matrix = new Matrix();
	int width = (int)pageSize.getWidth();
	int height = (int)pageSize.getHeight();
	matrix = page.getDisplayMatrix(0, 0, width, height, 0);
	Size size = new Size();
	size.setWidth(width);
	size.setHeight(height);
	Bitmap bmp = Bitmap.create(size,Bitmap.FORMAT_24BPP_BGR, null,0);

	Renderer render = Renderer.create(bmp);
	RenderContext renderContext = RenderContext.create();
	renderContext.setMatrix(matrix);
	renderContext.setFlags(RenderContext.RENDERCONTEXTFLAG_ANNOT);
	Progress renderProgress = page.startRender(renderContext, render, 0);
	if(renderContext !=null){
		int ret = Progress.TOBECONTINUED;
		while(ret == Progress.TOBECONTINUED ){
			ret = renderProgress.continueProgress(30);
		}
```

```
        }
        renderProgress.release();
        ...
}
```

4.11.2 How to convert png file to PDF file

```
try {

        PDFDocument document = PDFDocument.create();
        PDFPage page = null;
        page = pdfDocument.createPage(0);
        page.setSize(width, height);
        ImageObject imgObj = ImageObject.create(page);
        FileHandler  pngFile = FileHandler.create("1.png", FileHandler.FILEMODE_READONLY);
        Image img = Image.load(pngFile)
        imgObj.setImage(img);
        Matrix matrix = new Matrix();
        matrix.setScale(100, 100);
        matrix.postTranslate(100, 0);
        imgObj.setMatrix(matrix);

        PageObjects pageobjs = page.getPageObjects();
        pageobjs.insertObject(PageObject.TYPE_IMAGE, 0, imgObj);
        pageobjs.generateContents();
        FileHandler docFile = FileHandler.create("png2pdf.pdf", FileHandler.FILEMODE_TRUNCATE);
        Progress progress = doc.startSaveToFile(docFile, PDFDocument.SAVEFLAG_NOORIGINAL);
        progress.continueProgress(0);
        progress.release();
        docFile.release();
        pngFile.release();

}
catch (PDFException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4.12 Bookmark

Foxit PDF SDK provides navigational tools called Bookmarks to allow users to quickly locate and link their point of interest within a PDF document. PDF bookmark is also called outline, and each bookmark contains a destination or actions to describe where it links to. PDFBookmarkIterator object is created by calling **PDFDocument.createBookmarkIterator()**, and **PDFBookmarkIterator.getBookmarkData()** can be used to get the data of the current bookmark item.

Example:

4.12.1 How to create a bookmark tree and show all bookmarks

```
PDFDocument pdfDocument = null;
```

```
try {
	pdfDocument = PDFDocument.open(fileHandler, null);
	ArrayList<String> bookmarkArray = new ArrayList<String>();
	PDFBookmarkIterator i = pdfDocument.createBookmarkIterator();
	ArrayList<Integer> pageIndexArray = new ArrayList<Integer>();

	//only iterate upmost level

	i.moveToFirstChild();

	BookmarkData bookmarkData_first = i.getBookmarkData();
	bookmarkArray.add(bookmarkData_first.mTitle);

	int i_actions_count = i.countActions();

	PDFGotoAction pdfAction = (PDFGotoAction) i.getAction(0);
	pageIndexArray.add(pdfAction.getDestination().getPageIndex());

	while(!i.isLastChild())
	{
		i.moveToNextSibling();
		BookmarkData bookmarkData = i.getBookmarkData();
		bookmarkArray.add(bookmarkData.mTitle);

		PDFGotoAction pdfAction_internal = (PDFGotoAction) i.getAction(0);
		pageIndexArray.add(pdfAction_internal.getDestination().getPageIndex());
	}

	String displayString= new String();

	for(int j = 0; j<bookmarkArray.size(); j++)
	{
		displayString += bookmarkArray.get(j);
		displayString += " @ page: ";
		displayString += pageIndexArray.get(j);
		displayString += "\n";
	}

	final String threadDisplayString = displayString;

	parent.runOnUiThread(new Runnable() {
		public void run() {
			Toast.makeText(parent.getBaseContext(), threadDisplayString, 3).show();
		}
	});
}
catch (PDFException e) {
	e.printStackTrace();
}
```

4.12.2 How to remove all bookmarks from a PDF

```
try {

	pdfBookmark = pdfDocument.createBookmarkIterator();
	pdfBookmark.moveToRoot();

	haschild = true;
	while(pdfBookmark.hasChild())
```

```
        {
                pdfBookmark.moveToFirstChild();
                pdfBookmark.remove();
        }
        pdfBookmark.release();
        ...
}
catch (PDFException e) {
        e.printStackTrace();
}
```

4.13 Reflow

Reflow is a function that rearranges page content when the page size changes. It is useful for applications that have output devices with difference sizes. Reflow frees the applications from considering layout for different devices. This function provides APIs to create, render, release and access properties of 'reflow' pages.

Example:

4.13.1 How to create a reflow page

```
PDFDocument document = PDFDocument.open(fileHandler, null);
PDFPage page = document.getPage(0);
Progress parseProgress = page.startParse(PDFPage.PARSEFLAG_NORMAL);
if (parseProgress != null)
{
        int ret = Progress.TobeContinued;
        while (Progress.TobeContinued == ret)
        {
                ret = parseProgress.continueProgress(30);
        }
}
parseProgress.release();
if (page.isParsed() == false) return;

SizeF pageSize = page.getSize();
PDFReflowPage reflowPage = PDFReflowPage.create(page);
reflowPage.setSize(pageSize.mWidth, pageSize.mHeight);
Progress reflowpProgress = reflowPage.startParse(PDFReflowPage.REFLOWFLAG_NORMAL);
if (reflowpProgress != null)
{
        int ret = Progress.TOBECONTINUED;
        while (ret == Progress.TOBECONTINUED)
        {
                ret = reflowpProgress.continueProgress(30);
        }
}
reflowpProgress.release();
reflowPage.release();
document.closePage(page);
```

```
document.close();
```

4.14 Pressure Sensitive Ink

Pressure **S**ensitive **I**nk (**PSI**) is a technique to obtain varying electrical outputs in response to varying pressure or force applied across a layer of pressure sensitive devices. In PDF, PSI is usually used for hand writing signatures. PSI data are collected by touching screens or handwriting on boards. PSI data contains coordinates and canvas of the operating area which can be used to generate appearance of PSI. Foxit PDF SDK allows applications to create PSI, access properties, operate on ink and canvas, and release PSI.

Example:

4.14.1 How to create a PSI and set the related properties for it

```
PSI psi = null;
RectF psiRect = new RectF(100F, 100F, 200F, 200F);
RectF pdfRect = new RectF(100F, 100F, 200F, 200F);
PDFDocument document;
PDFPage page;

try {
	pdfDocument = PDFDocument.open(fileHandler, null);
	page = loadPDFPage(document);

	Progress parserProgress = null;
	parserProgress = page.startParse(PDFPage.PARSEFLAG_NORMAL);
	assertNotNull(parserProgress);
	int ret = parserProgress.continueProgress(0);
	assertEquals(ret, Progress.FINISHED); //
	parserProgress.release();

	psi = PSI.create(true);
	psi.initCanvas(200, 200);
	psi.setInkColor(0xff0000);
	psi.setInkDiameter(1);
	psi.addPoint(new PointF(300, 300), 0.5F, PSI.PT_MOVETO);
	psi.addPoint(new PointF(100, 100), 0.5F, PSI.PT_LINETO | PSI.PT_ENDPATH);
	psi.convertToPDFAnnot(psiRect, page, pdfRect);
	psi.release();

	document.closePage(page);
	document.close();

}
catch (PDFException e) {
	e.printStackTrace();
}
```

4.15 PDF Action

PDFAction is represented as the base PDF action class. Foxit PDF SDK provides APIs to create a series of actions and get the action handlers, such as embedded goto action, JavaScript action, named action and launch action, etc.

Example:

4.15.1 How to operate link action

```
try{
	PDFPage page = pdfDocument.getPage(nPageIndex);
	Matrix matrix = page.getDisplayMatrix(0, 0, pageWidth, pageHeight, PDFPage.ROTATION_0);

	//load all annotations first.
	page.loadAnnots();
	Point pt = new Point();
	pt.x = 100;
	pt.y = 100;
	Annot annot = page.getAnnotAtDevicePos(null, matrix, pt, 1.0f);

	//Only deal link annotation
	if (annot.getType().contentEquals(Annot.TYPE_LINK))
	{
		Link link = (Link)annot;
		PDFAction action = link.getAction(Annot.TRIGGER_ANNOT_MU, 0);

		//Only deal goto action
		if (action.getType() == PDFAction.ACTION_GOTO)
		{
			PDFGotoAction gotoAction = (PDFGotoAction)action;
			PDFDestination destination = gotoAction.getDestination();
			int newIndex = destination.getPageIndex();
			...
		}
		else if (action.getType() == PDFAction.ACTION_URI)
		{
			PDFURIAction uriAction = (PDFURIAction)action;
			String uri = uriAction.getURL();
			Toast.makeText(MainActivity.this, uri, Toast.LENGTH_LONG).show();
		}
	}
	else {
			Toast.makeText(MainActivity.this, "It is not a link annotation!",
Toast.LENGTH_LONG).show();

		}
}
catch (PDFException e1){
// TODO Auto-generated catch block
	if (e1.getLastError() == PDFException.ERRCODE_NOTFOUND){
	Toast.makeText(MainActivity.this, "It is not a annotation!", Toast.LENGTH_LONG).show();
```

```
        }
}
```

4.15.2 How to operate embedded goto action

```
try{

        PDFAttachments attachments = mDocument.loadAttachments();
        PDFAttachment attachment = PDFAttachment.create(mDocument);
        attachments.insertAttachment(0, attachment);
        String filePath = "1.txt";
        attachment.setFileName(filePath,false);
        FileHandler fhandler = FileHandler.create(inputFileDir+"1.txt",
                        FileHandler.FILEMODE_READONLY);
        attachment.setFile(fhandler);
        float[] destParams3 = {0.000001f,842,1};
        PDFDestination embeddedGotoDestination =
PDFDestination.create(3,PDFDestination.ZOOM_FACTOR,destParams3);

        PDFEmbeddedGotoActionTarget target = PDFEmbeddedGotoActionTarget.create("relationship",
"filename", "destname", "annotname", 0, 0);
      action = PDFAction.createEmbeddedGotoAction(target, attachment, embeddedGotoDestination,
true);

        PDFEmbeddedGotoAction embeddedGotoAction = (PDFEmbeddedGotoAction)action;
        embeddedGotoAction.setAttachment(attachment);
        ...
}
catch (PDFException e){
        e.printStackTrace();
}
```

4.15.3 How to operate launch action

```
try{

        PDFAction action = PDFAction.createLaunchAction("lanuch", "defaultPath", "open", "",
true);
        PDFLaunchAction launchAction = (PDFLaunchAction)action;
        launchAction.setFileName("launch2");
        launchAction.setDefaultPath("defaultPath2");
        PDFAttachment attachment = PDFAttachment.create(mDocument);
        launchAction.setAttachment(attachment);
        ...
}
catch (PDFException e){
        e.printStackTrace();
}
```

4.15.4 How to operate JavaScript action

```
try{
```

```
        PDFAction  action = PDFAction.createJavascriptAction("alert('javascript');");
        PDFJavascriptAction javascriptAction = (PDFJavascriptAction)action;
        javascriptAction.setJavascript("alert('javascript2');");
        …
}
catch (PDFException e){
        e.printStackTrace();
}
```

4.16 Page Object

Page object is a feature that allows novice users having limited knowledge of PDF objects to be able to work with PDF objects. Foxit PDF SDK provides APIs to add and delete PDF objects in a page and set specific attributes. Using page object, users can create PDF page from object contents. Other possible usages of page object include adding headers and footer to PDF documents, adding an image logo to each page, and generating a template PDF on demand.

Example:

4.16.1 How to create an image object in a PDF page

```
//Assuming PDFPage page and Bitmap bitmap has been created.
try {
      ImageObject imageObject = ImageObject.create(page);
      imageObject.setBitmap(bitmap, null);
      PageObjects pageObjects = page.getPageObjects();
      pageObjects.insertObject(PageObject.TYPE_IMAGE, 0, iamgeObject);
      pageObjects.generateContents();
}
catch (PDFException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4.17 Watermark

Watermark is a type of PDF annotation and is widely used in PDF document. Watermark is a visible embedded overlay on a document consisting of text, a logo, or a copyright notice. The purpose of a watermark is to identify the work and discourage its unauthorized use. Foxit PDF SDK provides APIs to work with watermark, allowing applications to create, insert, and remove watermarks.

Example:

4.17.1 How to create a text watermark and insert it into the first page

```
WatermarkTextProperty properties = new PDFWatermark.WatermarkTextProperty();
properties.alignment = PDFWatermark.TEXTALIGNMENT_CENTER;
properties.color = 0;
```

```
properties.font = Font.createStandard(Font.STDFONT_COURIER);
properties.fontSize = 16;
properties.fontStyle = PDFWatermark.FONTSTYLE_NORMAL;
properties.lineSpace = 1.5f;

WatermarkSetting settings = new WatermarkSetting();
settings.flags = PDFWatermark.FLAG_NOPRINT;
settings.offsetX = 21.3f;
settings.offsetY = 23.1f;
settings.opacity = 99;
settings.position = PDFWatermark.POS_CENTER;
settings.rotation = 80.0f;
settings.scaleX = 0.3f;
settings.scaleY = 0.3f;

FileHandler fileHInput = FileHandler.create((inputFile), FileHandler.FILEMODE_READONLY);
PDFDocument doc = PDFDocument.open(fileHInput, null);

PDFWatermark watermark = PDFWatermark.create(doc, text, properties, settings);
int pageCount=doc.countPages();
for(int i=0; i<pageCount; i++){
	PDFPage page=doc.getPage(i);
	if(page.isParsed() == false)
	{
		Progress progress = page.startParse(PDFPage.PARSEFLAG_NORMAL);
			if(progress!= null)
			{
				int ret = Progress.TOBECONTINUED;
				while (ret == Progress.TOBECONTINUED)
				{
					ret = progress.continueProgress(30);
				}
			}
	}
	watermark.insertToPage(page);
}
```

4.17.2 How to create an image watermark and insert it into the first page

```
try {
	FileHandler inputImage=FileHandler.create(inputFilePath+"imageBMP.bmp",
FileHandler.FILEMODE_READONLY);
	Image image=Image.load(inputImage);
	image.loadFrame(0);
	WatermarkSetting  settings = new PDFWatermark.WatermarkSetting();
	settings.flags = PDFWatermark.FLAG_ONTOP;
	settings.offsetX = 1.0f;
	settings.offsetY = 1.0f;
	settings.opacity = 100;
	settings.position = PDFWatermark.POS_CENTER;
	settings.rotation = 0.0f;
	settings.scaleX = 1.0f;
	settings.scaleY = 1.0f;
```

```
        watermark=PDFWatermark.create(doc, image, settings);
        page = doc.getPage(0);
        Progress progress = page.startParse(PDFPage.PARSEFLAG_NORMAL);
        progress.continueProgress(0);
        progress.release();
        watermark.insertToPage(page);
        FileHandler docFile = FileHandler.create("watermark.pdf",
FileHandler.FILEMODE_TRUNCATE);
        Progress progress = doc.startSaveToFile(docFile, PDFDocument.SAVEFLAG_NOORIGINAL);
        progress.continueProgress(0);
        progress.release();
        docFile.release();

        doc.closePage(page);
        watermark.release();
        image.release();
        inputImage.release();
        …
}
catch (PDFException e) {
        e.printStackTrace();
        return;
}
```

4.17.3 How to remove a specified watermark from a page

```
try {
        FileHandler handler = FileHandler.create(fileName,   FileHandler.FILEMODE_READONLY);
        pdfDocument = PDFDocument.open(handler, password.getBytes());
        page = doc.getPage(0);
        if(page.isParsed() == false)
        {
                Progress progress = page.startParse(PDFPage.PARSEFLAG_NORMAL);
                if (progress != null)
                {
                        int ret = Progress.TOBECONTINUED;
                        while (ret == Progress.TOBECONTINUED)
                        {
                                ret = progress.continueProgress(30);
                        }
                        progress.release();
                }
        }

        int count = page.countWatermarks();
        Boolean result = page.removeWatermark(index);
        ...
}
catch (PDFException e) {
        e.printStackTrace();
        return;
}
```

4.17.4 How to remove all watermarks from a page

```
try {
	FileHandler handler = FileHandler.create(fileName,   FileHandler.FILEMODE_READONLY);
	pdfDocument = PDFDocument.open(handler, password.getBytes());
	page = doc.getPage(0);
	if(page.isParsed() == false)
	{
		Progress progress = page.startParse(PDFPage.PARSEFLAG_NORMAL);
		if (progress != null)
		{
			int ret = Progress.TOBECONTINUED;
			while (ret == Progress.TOBECONTINUED)
			{
				ret = progress.continueProgress(30);
			}
			progress.release();
		}
	}

	int count = page.countWatermarks();
	page.removeWatermarks();
	...
}
catch (PDFException e) {
	e.printStackTrace();
	return;
}
```

4.18 Security

Foxit PDF SDK provides a range of encryption and decryption functions to meet different level of document security protection. Users can use regular password encryption and certificate-driven encryption, or using their own security handler for custom security implementation.

Example:

4.18.1 How to encrypt a PDF file with user password "123" and owner password "456"

```
FileHandler fileHInput = FileHandler.create((inputFile), FileHandler.FILEMODE_READONLY);
PDFDocument pdfDoc = PDFDocument.open(fileHInput, null);
PasswordEncryptionParams pwdenparams = new PasswordEncryptionParams();
try {
	pwdenparams.setCipher(EncryptionParams.CIPHER_RC4, 16);
	} catch (PDFException e){
		e.printStackTrace();
		pdfDoc.close();
		fileHInput.release();
		return;
		}
	pwdenparams.setEncryptMetadata(true);
```

```
	pwdenparams.setUserPermissions(PDFDocument.PERMISSION_PRINT);
	pwdenparams.setUserPassword((new String("123")).getBytes());
	pwdenparams.setOwnerPassword((new String("456")).getBytes());
	FileHandler fileHEncrypt = FileHandler.create((outputFile),
FileHandler.FILEMODE_TRUNCATE);
	Progress encryptProgress = null;
	try {
		encryptProgress = pdfDoc.startEncryption(pwdenparams, fileHEncrypt,
PDFDocument.SAVEFLAG_INCREMENTAL);
		int status = Progress.TOBECONTINUED;
		while (Progress.TOBECONTINUED == status)
			status = encryptProgress.continueProgress(0);
		encryptProgress.release();
	} catch (PDFException e){
		e.printStackTrace();
		fileHEncrypt.release();
		pdfDoc.close();
		fileHInput.release();
		return;
		}
	fileHEncrypt.release();
	fileHEncrypt = null;
	pdfDoc.close();
	pdfDoc = null;
	fileHInput.release();
	fileHInput = null;
```

4.18.2 How to encrypt a PDF file with Certificate

```
try {
	FileHandler  encryptedFile = FileHandler.create(mParam.encryptedFile,
FileHandler.FILEMODE_TRUNCATE);
	CertificateEncryptionParams params = new CertificateEncryptionParams();
	params.setCipher(EncryptionParams.CIPHER_RC4);
	params.setEncryptMetadata(true);
	params.setFilePath(inputFilePath + "foxit.cer");
	Progress progress = document.startEncryption(params, encryptedFile, flag);
	if(progress != null)
	{
		int ret = Progress.TOBECONTINUED;
		while (ret == Progress.TOBECONTINUED)
		{
			ret = progress.continueProgress(30);
		}
		progress.release();
	}
	encryptedFile.release();
	...
}
catch (PDFException e) {
	e.printStackTrace();
	return;
```

```
}
```

4.18.3 How to encrypt a PDF file with Foxit DRM

```
try {

	MyFoxitDRMHandler drmHandler = new MyFoxitDRMHandler();
	FoxitDRMEncryptionParams drmParams = new FoxitDRMEncryptionParams();
	drmParams.setEncryptMetadata(true);
	CryptionParams cryptParams = null;
	cryptParams = drmHandler.new CryptionParams();
	cryptParams.isOwner = true;
	cryptParams.userPermission = 0xFFFFFFFC;
	cryptParams.cipher = EncryptionParams.CIPHER_AES;
	cryptParams.keyLen = 16;
	cryptParams.fileID = "FileID";
	cryptParams.initialKey = "123";
	drmParams.setCryptionParams(cryptParams);
	drmParams.setSubFilter("drm");
	String[] strKey =
{"Issuer","Creator","FileID","FlowCode","Order","User","ServiceURL","Vender"};
	String[] strValue = {"IS","CR","FI","FL","OR","US","SE","VE"};
	for (int i = 0; i<strKey.length; i++)
		drmParams.setDRMParam(strKey[i],  strValue[i]);
	FileHandler fileHEncrypt = FileHandler.create(outputFilePath + "drm_encrypt.pdf",
FileHandler.FILEMODE_TRUNCATE);
	Progress encryptProgress = null;
	encryptProgress = pdfDoc.startEncryption(drmParams, fileHEncrypt,
PDFDocument.SAVEFLAG_INCREMENTAL);
	assertTrue(null != encryptProgress);
	int status = Progress.TOBECONTINUED;
	while (Progress.TOBECONTINUED == status)
		status = encryptProgress.continueProgress(0);
	encryptProgress.release();
	fileHEncrypt.release();
	fileHEncrypt = null;
	pdfDoc.close();
	pdfDoc = null;
	fileHInput.release();
	fileHInput = null;
	...
}
catch (PDFException e) {
	e.printStackTrace();
	return;
}
```

4.19 Signature

PDF Signature module can be used to create and sign digital signatures for PDF documents, which protects the security of documents' contents and avoids it to be tampered maliciously. It can let the receiver make sure that the document is released by the signer and the contents of the document are

complete and unchanged. Foxit PDF SDK provides APIs to create digital signature, verify the validity of signature, delete existing digital signature, get and set properties of digital signature, display signature and customize the appearance of the signature form fields. It provides default signature signing and custom signature signing to sign or verify a PDF document. The custom signature signing only provides the third-party signature interface and requires the customers have their own signature implementation.

Note

- *For Signature module, if you want to purchase Foxit PDF SDK license and use any functions of this module, you need contact Foxit to enable this module explicitly.*
- *Starting from SDK 5.2, if you have already integrated the third-party signature feature into your project using SDK version before 5.2, the signature fields may not be able to verify successfully when using your original callback implementation. In that case, you need to do a conversion in the callback function verify(Object clientData, Object context, Signature sigField, final String digest, final String signedData) that converts the input signedData to the value which is the same as the return value from the callback function sign(Object clientData, Object context, Signature sigField, final String digest). It is because the signedData provided by Foxit (used to verify the signature fields) is in all uppercase from SDK 5.2, you need to convert it in your verify callback function and make sure it is the same as the return value from the sign callback function. If you didn't integrate the third-party signature feature into your project using SDK version before 5.2, just ignore this notice.*

Example:

4.19.1 How to sign a PDF document with custom signature signing

```
MySignatureHandler handler = new MySignatureHandler();
Signature.registerSignatureHandler("Adobe.PPKLite", null, handler);
FileHandler fileHInput = FileHandler.create((inputFile), FileHandler.FILEMODE_READONLY);
PDFDocument document = PDFDocument.open(fileHInput, null);

signature = document.getSignature(0);
signature.initValue();
signature.setFilter("Adobe.PPKLite");
signature.setAppearanceFlags(Signature.APPEARANCE_FOXITFLAG);
signature.resetAppearance();
FileHandler signedFile = FileHandler.create(outputFilePath+"startSign_case1_"+"0"+".pdf",
FileHandler.FILEMODE_TRUNCATE);
progress = signature.startSign(signedFile);
if(progress != null)
{
	int ret = Progress.TOBECONTINUED;
	while (ret == Progress.TOBECONTINUED)
	{
		ret = progress.continueProgress(0);
```

```
        }
        progress.release();
}
        signedFile.release();
        document.close();
        document = null;
        fileHInput.release();
        fileHInput = null;
```

Implement signature callback function of signing on MySignatureHandler class

```
@Override
public String sign(Object clientData, Object context, Signature sigField,
                        String digest) throws PDFException {
// TODO Auto-generated method stub
        String tobeSigned = null;
        String encryptStr = null;
        byte[] arrall = null;
        try {
                try {
                        FileHandler filehandler =  pContext.m_file;
                        int size = filehandler.getSize();
                        byte[] arr1 =
filehandler.read((int)pContext.m_pByteRangeArray[0],(int)pContext.m_pByteRangeArray[1]);
                        byte[] arr2 =
filehandler.read((int)pContext.m_pByteRangeArray[2],(int)pContext.m_pByteRangeArray[3]);
                        size = 0;
                        arrall = new
byte[(int)pContext.m_pByteRangeArray[1]+(int)pContext.m_pByteRangeArray[3]];
                        System.arraycopy(arr1,0,arrall,0,arr1.length);
                        System.arraycopy(arr2,0,arrall,arr1.length,arr2.length);
                    } catch (Exception e) {
                      // TODO Auto-generated catch block
                      e.printStackTrace();
                      }
                //third party sign interface
                encryptStr = CertUtil.SignMsg(arrall, "com/verify/cert/foxit_all.pfx", "123");
                return encryptStr;
        } catch (Exception e) {
         // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return null;
}
```

5 SAMPLE APPLICATION

The sample applications (demos) were provided to help users to develop applications based on Foxit PDF SDK. Developers can quickly get started on embedding PDF technology in their applications with those demos.

5.1 mt_watermark

The mt_watermark demo illustrates how to implement multi-thread applications based on Foxit PDF SDK to achieve higher performance. This demo performs to add watermark in multiple documents with multi-thread support.

5.2 fdf

The fdf demo illustrates how to use Foxit PDF SDK to export annotations in PDF files to external FDF files and how to import an external FDF file into a PDF file.

5.3 img2pdf

The img2pdf demo illustrates how to use Foxit PDF SDK to insert image files into a newly-created PDF file and how to convert a multi-page tif file to a PDF file.

5.4 pdf2img

The pdf2img demo illustrates how to use Foxit PDF SDK to convert PDF files to image files that are supported by Foxit PDF SDK. This demo renders PDF pages to bitmaps, and then save the bitmaps to image files.

5.5 signature

The signature demo illustrates how to use Foxit PDF SDK to add signatures to PDF files. This demo uses the default signature signing.

6 FAQ

1. **What's the price of Foxit PDF SDK for Java?**
   To receive a price quotation, please send a request to sales@foxitsoftware.com or call Foxit sales at 1-866-680-3668.

2. **How can I activate it after purchasing Foxit PDF SDK for Java?**
   There are detailed descriptions on how to apply a license in the section 3.2.2 or 3.3.2. You can refer to the descriptions to activate a license.

3. **How can I look for technical support when I try Foxit PDF SDK for Java?**
   You can send email to support@foxitsoftware.com for any questions or comments or call our support at 1-866-693-6948.

REFERENCES

[1] PDF reference 1.7
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[2] Foxit PDF SDK API reference
sdk_folder/docs/ Foxit Java SDK API Manual.chm

Note: sdk_folder is the directory of unzipped package.

SUPPORT

Foxit support home link:

http://www.foxitsoftware.com/support/

Sales contact phone number:

Phone: 1-866-680-3668

Email: sales@foxitsoftware.com

Support & General contact:

Phone: 1-866-MYFOXIT or 1-866-693-6948

Email: support@foxitsoftware.com

GLOSSARY OF TERMS & ACRONYMS

catalog	The primary dictionary object containing references directly or indirectly to all other objects in the document, with the exception that there may be objects in the trailer that are not referred to by the catalog
character	Numeric code representing an abstract symbol according to some defined character encoding rule
developer	Any entity, including individuals, companies, non-profits, standards bodies, open source groups, etc., who are developing standards or software to use and extend ISO 32000-1
dictionary object	An associative table containing pairs of objects, the first object being a name object serving as the key and the second object serving as the value and may be any kind of object including another dictionary
direct object	Any object that has not been made into an indirect object
FDF file	File conforming to the Forms Data Format containing form data or annotations that may be imported into a PDF file
filter	An optional part of the specification of a stream object, indicating how the data in the stream should be decoded before it is used
font	Identified collection of graphics that may be glyphs or other graphic elements
function	A special type of object that represents parameterized classes, including mathematical formulas and sampled representations with arbitrary resolution
glyph	Recognizable abstract graphic symbol that is independent of any specific design
indirect object	An object that is labelled with a positive integer object number followed by a non-negative integer generation number followed by object and having end object after it
integer object	Mathematical integers with an implementation specified interval centred at 0 and written as one or more decimal digits optionally preceded by a sign

name object	An atomic symbol uniquely defined by a sequence of characters introduced by a SOLIDUS (/), (2Fh) but the SOLIDUS is not considered to be part of the name
null object	A single object of type null, denoted by the keyword null, and having a type and value that are unequal to those of any other object
numeric object	An integer object representing mathematical integers or a real object representing mathematic real numbers
object	Basic data structure from which PDF files are constructed. Types of objects in PDF include: boolean, numerical, string, name, array, dictionary, stream and null
object reference	An object value used to allow one object to refer to another; that has the form “<n> <m> R” where <n> is an indirect object number, <m> is its version number and R is the uppercase letter R
PDF	Portable Document Format file format defined by this specification [ISO 32000-1]
real object	This object used to approximate mathematical real numbers, but with limited range and precision and written as one or more decimal digits with an optional sign and a leading, trailing, or embedded PERIOD (2Eh) (decimal point)
rectangle	A specific array object used to describe locations on a page and bounding boxes for a variety of objects and written as an array of four numbers giving the coordinates of a pair of diagonally opposite corners, typically in the form [llx lly urx ury] specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates of the rectangle, in that order
stream object	This object consists of a dictionary followed by zero or more bytes bracketed between the keywords stream and endstream
string object	This object consists of a series of bytes (unsigned integer values in the range 0 to 255). String objects are not integer objects, but are stored in a more compact format