Foxit

DEVELOPER GUIDE

Foxit® PDF SDK

For Android

Microsoft® Partner
Gold Independent Software Vendor (ISV)

TABLE OF CONTENTS

1 Introduction to Foxit PDF SDK 1

1.1 Why is Foxit your choice 1

1.2 Features 1

1.2.1 Evaluation 2

1.2.2 License 2

1.2.3 Out of Memory (OOM) 2

1.3 About this guide 3

2 Introduction to PDF 4

2.1 History of PDF 4

2.2 PDF Document Structure 4

2.3 PDF Document Features 4

3 Getting Started 5

3.1 System Requirements 5

3.2 What is in the Package 5

3.3 How to apply a license 10

3.4 How to run a demo 10

3.4.1 Demo Environment 10

3.4.2 Setting up and running demo project 10

4 Working with SDK API 40

4.1 Apply a License 40

4.1.1 How to apply a license 40

4.2 File 40

4.2.1 How to create a FileHandler object 40

4.3 Document 41

4.3.1 How to get the first page of a PDF 41

4.3.2 How to save PDF to a file 41

4.4 Attachment 42

4.4.1 How to insert an attachment file into a PDF 42

4.4.2 How to remove a specific attachment of a PDF 42

4.4.3 How to change the properties of an attachment 43

4.5 Page 43

4.5.1 How to create a PDF page 43

4.5.2 How to get page size 43

4.5.3 How to delete a PDF page 44

4.5.4 How to flatten a PDF page 44

4.5.5 How to calculate bounding box of page contents 44

4.6 Render 45

4.6.1 How to parse a PDF page 45

4.6.2 How to render a PDF page by drawing bitmaps 45

4.7 Text Page 46

4.7.1 How to select text in a PDF page 46

4.7.2 How to search text in a PDF page 47

4.7.3 How to extract text from a PDF 47

4.8 Text Link 47

4.8.1 How to get the first URL formatted texts in a PDF page 48

4.9 Form 48

4.9.1 How to load the forms in a PDF 48

4.9.2 How to count form fields and get the properties 48

4.9.3 How to export the form data in a PDF to a FDF file 49

4.9.4 How to import form data from a FDF file 49

4.9.5 How to get and set the properties of form fields 49

4.10 Form Filler 50

4.10.1 How to fill a form with form filler. 50

4.11 Form Design 50

4.11.1 How to add a text form field to a PDF document. 51

4.11.2 How to remove a text form field from a PDF 51

4.12 Annotations 51

4.12.1 How to add a link annotation to another page in the same PDF 53

4.12.2 How to add a highlight annotation to a page and set the related annotation properties 53

4.13 Image Conversion 54

4.13.1 How to convert PDF pages to bitmap files 54

4.13.2 How to convert png file to PDF file 55

4.14 Bookmark 56

4.14.1 How to create a bookmark tree and show all bookmarks 56

4.14.2 How to remove all bookmarks from a PDF 57

4.15 Reflow 57

4.15.1 How to create a reflow page 57

4.16 Pressure Sensitive Ink 58

4.16.1 How to create a PSI and set the related properties for it 58

4.17 PDF Action 59

4.17.1 How to operate link action 59

4.17.2 How to operate embedded goto action 60

4.17.3 How to operate launch action 61

4.17.4 How to operate JavaScript action 61

4.18 Page Object 61

4.18.1 How to create an image object in a PDF page 61

4.18.2 How to create a path object and set its properties 62

4.19 Marked content 62

4.19.1 How to get marked content in a page and get the tag name 62

4.20 Layer 63

4.20.1 How to traverse layer tree and set the opposite visible state of every PDF layer 63

4.21 Watermark 64

4.21.1 How to create a text watermark and insert it into the first page 64

4.21.2 How to create an image watermark and insert it into the first page 65

4.21.3 How to remove a specified watermark from a page 66

4.21.4 How to remove all watermarks from a page 66

4.22 Security 67

4.22.1 How to encrypt a PDF file with user password "123" and owner password "456" 67

4.22.2 How to encrypt a PDF file with Certificate 68

4.22.3 How to encrypt a PDF file with Foxit DRM 68

4.23 RMS 69

4.23.1 How to encrypt a PDF document with Microsoft RMS 69

4.23.2 How to decrypt a PDF document with Microsoft RMS 70

4.24 Signature 71

4.24.1 How to sign the PDF document with default signature handler 72

4.24.2 How to sign the PDF document with custom signature handler 73

5 FAQ 74

References 75

Support 76

Glossary of Terms & Acronyms 77

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 is Foxit 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.
- **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 Android is a Software Development Kit written in Java. It enables users to develop their applications on Android platform. It allows developers to perform operations such like view, text search, adding bookmarks in PDF documents and applying pressure sensitive ink (PSI) by using Foxit PDF technology.

Foxit PDF SDK for Android 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
Marked Content	Get and edit content mark
Form	Form filling with JavaScript support
Reflow	Arrange page content to fit changed page size
Pressure Sensitive Ink	Incorporate pressure sensitive digital ink capabilities into PDF solutions
Layer	Access layer information
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
RMS	Support Microsoft RMS encryption and decryption
Out Of Memory	Recover from an OOM condition

1.2.1 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.

1.2.2 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.

1.2.3 Out of Memory (OOM)

Development of robust PDF applications is challenging on mobile platforms which offer limited memory. When memory allocation fails, applications may crash or exit unexpectedly. To deal with this issue, Foxit PDF SDK introduces an OOM mechanism to support applications.

When OOM occurs, Foxit PDF SDK should be able to detect it, report it to applications and recover the data. To achieve this mechanism, Foxit SDK classifies object handles into **short-term handle** and **long-term handle**. Short-term handles are released when OOM occurs. Both short-term handles used in current operations and used by long-term handles are recovered. Overall, applications have three options when receiving the OOM notification from Foxit SDK.

a) Prompt users of OOM and exit the application.

b) Prompt users of OOM and keep running. If no change is made to the PDF document or no short-term handle is used by applications, the whole document could be fully recovered. Otherwise, some data could be lost due to the release of short-term handles.

c) Keep running and recover all handles (short term and long term).

Foxit PDF SDK recommends developers to use the second option while the third option requires special support.

1.3 About this guide

This guide is intended for the developers who need to integrate Foxit PDF SDK for Android into their own applications. It aims at introducing installation package structure on Android platform, 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 supports all 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 Android platform. The following sections introduce the structure of the installation package, how to apply a license, and how to run a demo on Android platform.

3.1 System Requirements

Runtime requirements: Android version 2.2 (API-Level-8) and later versions

The release package for Android Java APIs includes 2 types of "*.so" and "*.Jar" SDK libraries

1) x86 library for x86 devices
2) armeabi-v7a/arm64-v8a library for arm devices

Android device: At least 10 MB free disk space is required. Memory requirement depends on source document to be used.

Note: The Android device you used should support Neon, or the program may crash when running. If you need to use those devices that do not support Neon, please contact us at support@foxitsoftware.com.

3.2 What is in the Package

Download Foxit PDF SDK zip for Android Java APIs and extract it to a new directory like "foxitpdfsdk_5_3_Android". The structure of the release package is shown in Figure 3-1. One thing to note is 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. The release package contains the following folders:

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

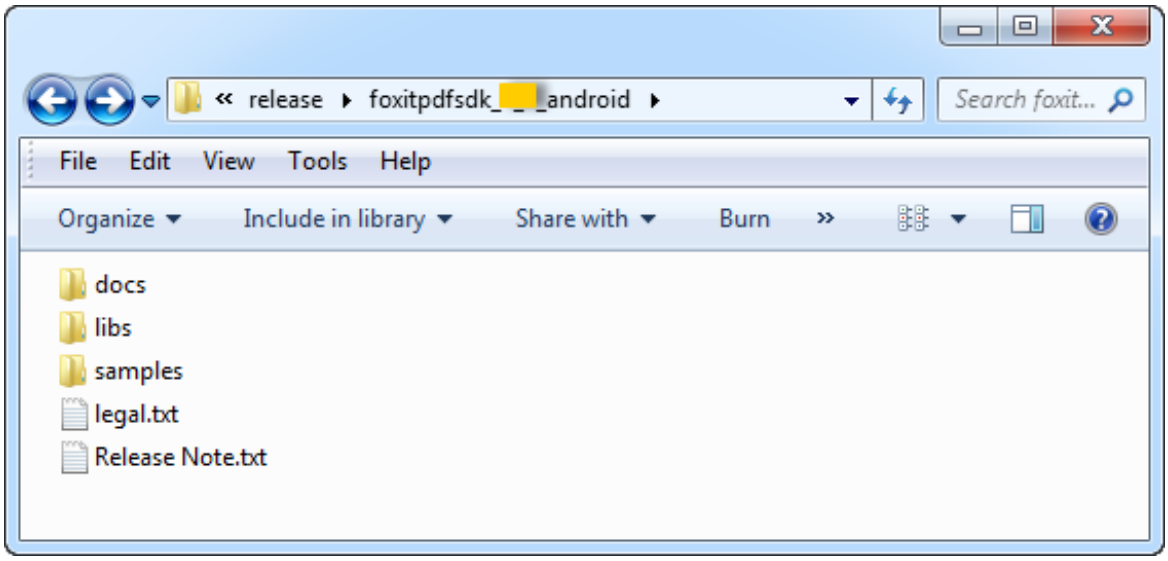

Figure 3-1

Foxit PDF SDK provides “fsdk_android.jar” file in directory “libs”, it contains 15 packages that are shown in Figure 3-2. In each package, there are java classes listed in Table 3-1.

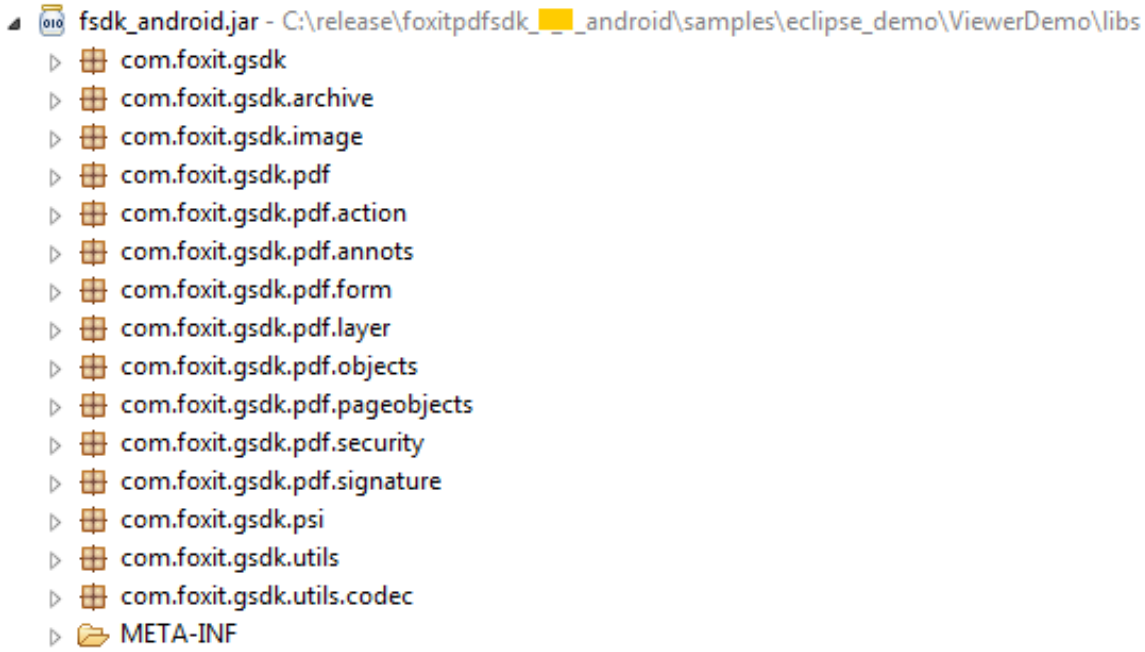

Figure 3-2

Table 3-1

Java Package	Java Class
com.foxit.gsdk	IApp.class IInvalidate.class PDFException.class PDFLibrary.class
com.foxit.gsdk.archive	Archive.class
com.foxit.gsdk.image	Image.class ImageFile.class
com.foxit.gsdk.pdf	BookmarkPos.class DefaultAppearance.class Font.class FontManager.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
com.foxit.gsdk.pdf.action	PDFAction.class PDFDestination.class PDFEmbeddedGotoAction.class PDFEmbeddedGotoActionTarget.class PDFGotoAction.class PDFHideAction.class PDFImportDataAction.class

Java Package	Java 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 Text.class TextMarkup.class TrapNet.class UnderLine.class Watermark.class Widget.class

Java Package	Java Class
com.foxit.gsdk.pdf.form	FormActionHandler.class PDFForm.class PDFFormControl.class PDFFormField.class PDFFormFiller.class
com.foxit.gsdk.pdf.layer	Layer.class LayerContext.class LayerNode.class
com.foxit.gsdk.pdf.objects	Dictionary.class PDFObject.class
com.foxit.gsdk.pdf.pageobjects	ImageObject.class MarkedContent.class PageObject.class PageObjects.class PathObject.class
com.foxit.gsdk.pdf.security	CertificateEncryptionParams.class CertificateHandler.class CustomEncryptionParams.class EncryptionParams.class FoxitDRMEncryptionParams.class FoxitDRMHandler.class PasswordEncryptionParams.class RMSEncryptionParams.class SecurityHandler.class
com.foxit.gsdk.pdf.signature	Signature.class SignatureHandler.class TSAClient.class
com.foxit.gsdk.psi	PSI.class
com.foxit.gsdk.utils	DateTime.class FileHandler.class Size.class SizeF.class
com.foxit.gsdk.utils.codec	Base64.class

3.3 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=").

```
static{
	System.loadLibrary("fsdk_android");
}

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

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

3.4 How to run a demo

3.4.1 Demo Environment

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

- libs/armeabi-v7a/libfsdk_android.so (or libs/x86/libfsdk_android.so, libs/arm64-v8a/libfsdk_android.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_android.jar) – operates on the Java layer used by the Virtual Machine. They provide all the classes and functionalities of our PDF library.

3.4.2 Setting up and running demo project

Download and install Eclipse IDE (http://www.eclipse.org/), Android Studio IDE (http://developer.android.com/sdk/installing/studio.html), and Xamarin Studio IDE (http://xamarin.com) and Android SDK (http://developer.android.com/sdk/index.html).

Foxit PDF SDK provides three types of demos which are running in Eclipse, Android Studio and Xamarin Studio respectively as shown in Figure 3-3.

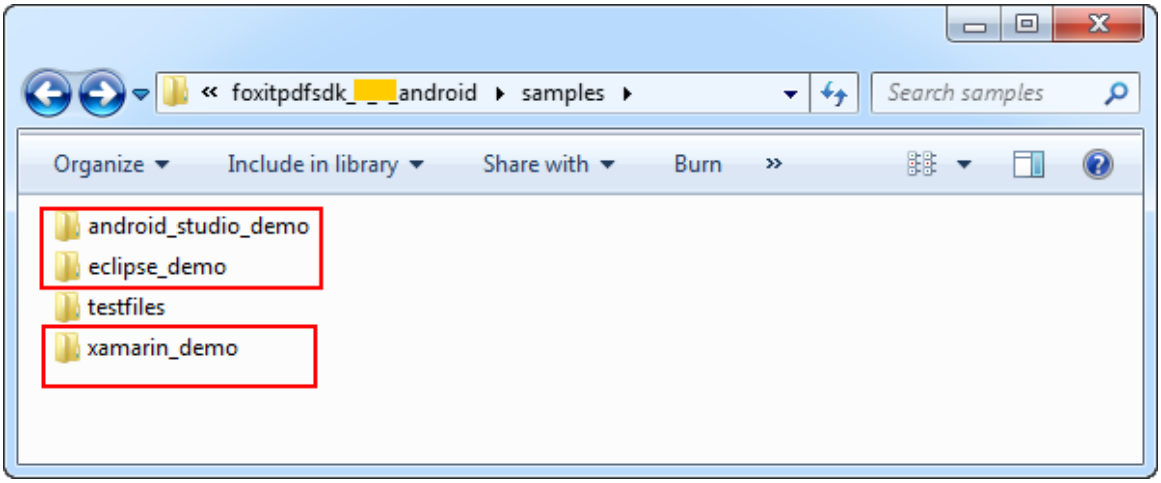

Figure 3-3

3.4.2.1 Eclipse Demo

In "samples/eclipse_demo", there are a viewer demo, an OOM handing demo and a form filling demo illustrating how to implement a simple viewer, how to handle OOM and how to fill a form including importing/exporting FDF file with SDK respectively. The demos are shown in Figure 3-4.

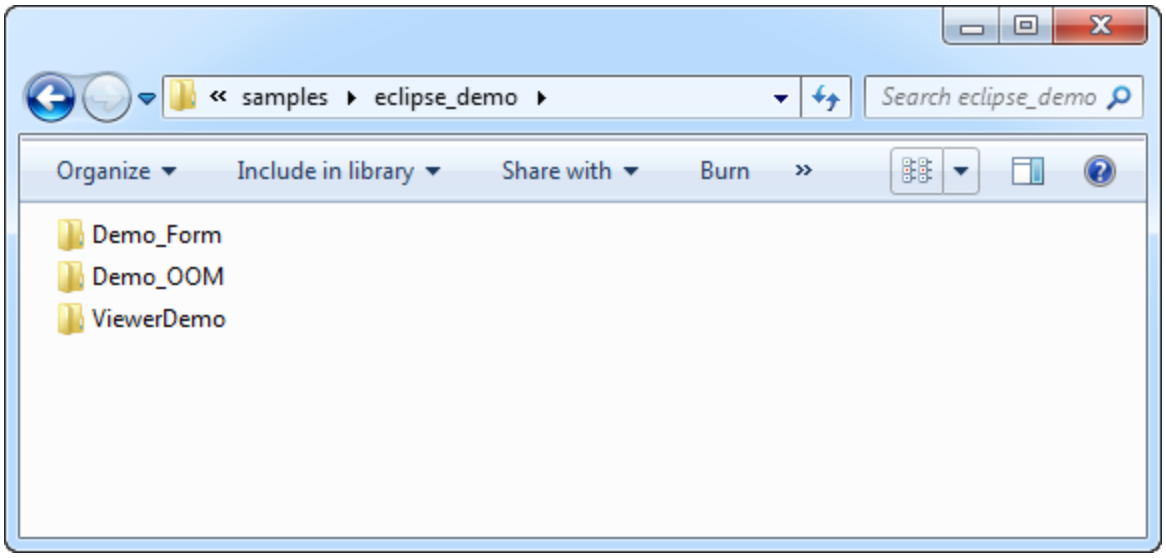

Figure 3-4

OOM Demo

This demo provides an example for handling OOM using PDF SDK APIs. To run it in Eclipse, follow the steps below:

a) Import the project into Eclipse following "File->Import-> 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 3-5.

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_android.jar” file, “arm64-v8a”, “armeabi-v7a” and “x86” folders in directory “foxitpdfsdk_4_5_android/libs” to “Demo_OOM/libs” in the workspace.

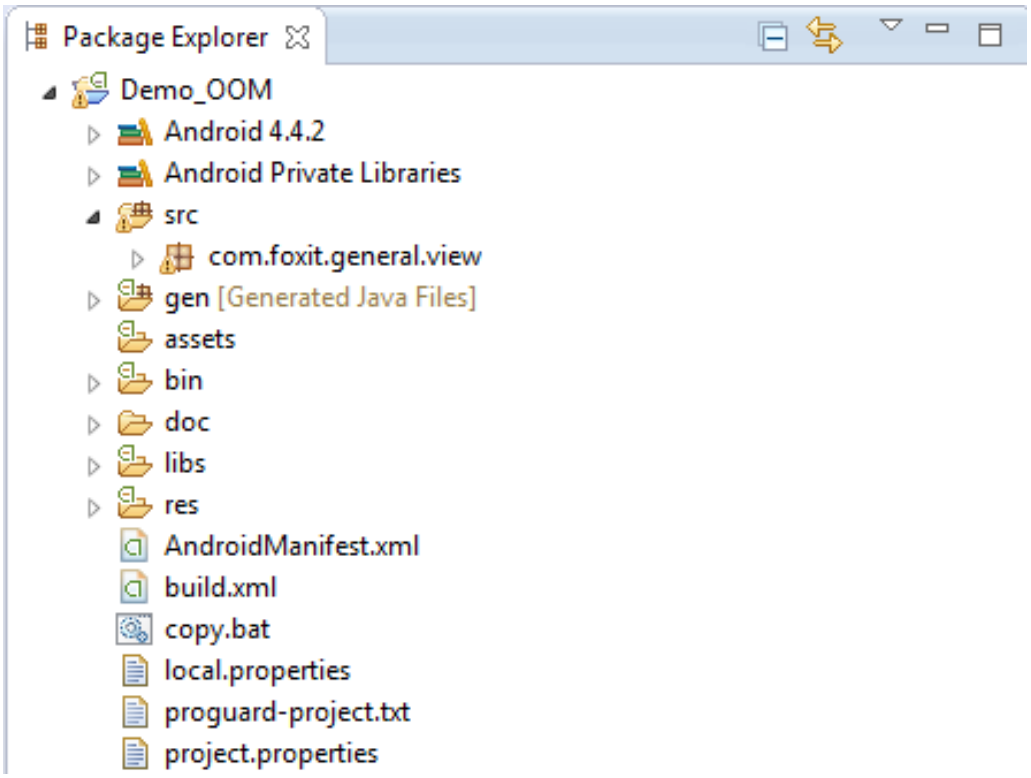

Figure 3-5

b) Push the PDF files under “samples/testfiles” folder to the SD card. Four PDF files will be displayed in “storage/sdcard” as shown in Figure 3-6.

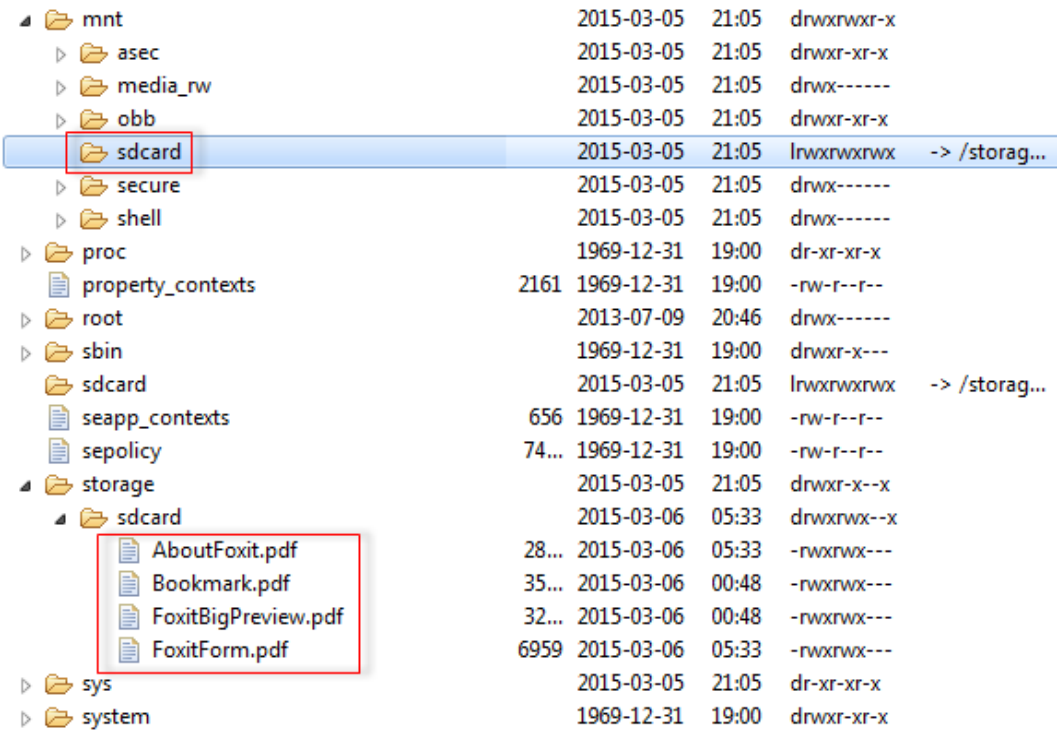

Figure 3-6

c) Select the demo project in package explorer, then choose “Run As → Android Application” to download the demo onto a device or an emulator (AVD), and it will launch automatically.

Figure 3-7 shows the demo running in an AVD targeting Android 4.4.2. Here we set the memory size to 20(MB).

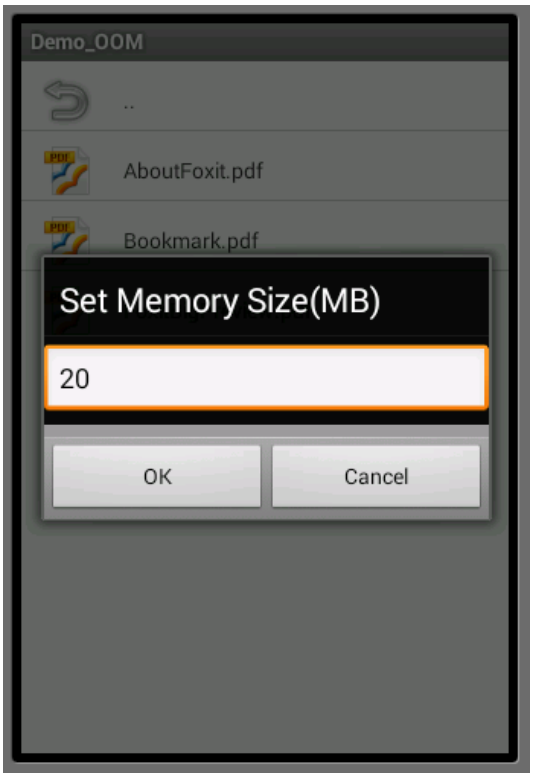

Figure 3-7

d) After setting the memory size to 20, click on “OK”, and then choose the “FoxitBigPreview.pdf” as shown in Figure 3-8.

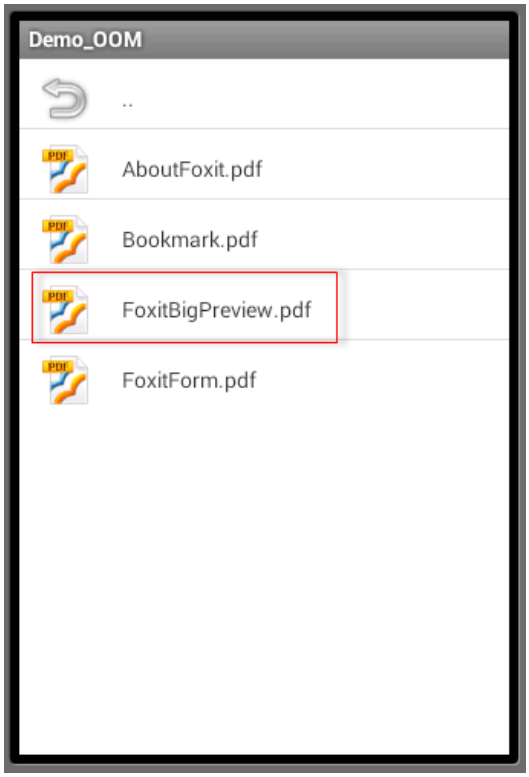

Figure 3-8

e) The “FoxitBigPreview.pdf” is displayed in the screen as shown in Figure 3-9, which means the setting memory size is enough to load the PDF file.

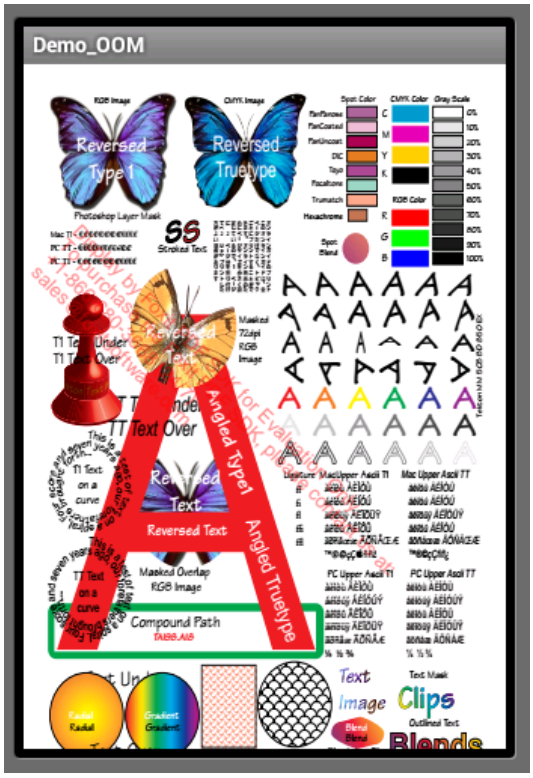

Figure 3-9

f) If we set the memory size to 10(MB) in the start page of the demo, then click on “OK” and choose the “FoxitBigPreview.pdf”, it will show a message box like Figure 3-10.

Figure 3-10

g) The message “OOM…unrecover” means that the setting memory size is not enough to load the PDF file. In this case, click on “OK”, and then press “back” and “menu” keys on the AVD. A menu Item “Set memory size (MB)” will appear as shown in Figure 3-11.

h) Click the menu item, we can reset the memory size as shown in Figure 3-12.

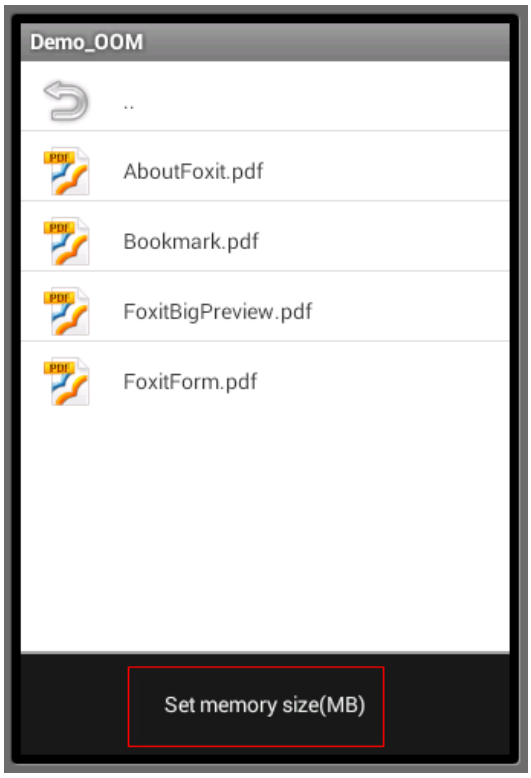

Figure 3-11

Figure 3-12

Viewer Demo

a) To run this demo in Eclipse, you can refer to the OOM demo. Figure 3-13 shows the viewer demo running in an AVD targeting Android 4.4.2.

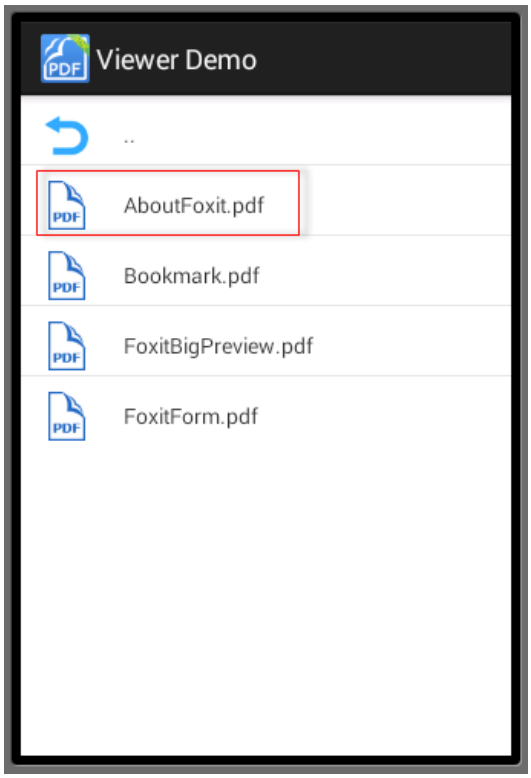

Figure 3-13

b) Click the “AboutFoxit.pdf”, and the PDF file will be displayed as shown in Figure 3-14. The red words are functional specifications.

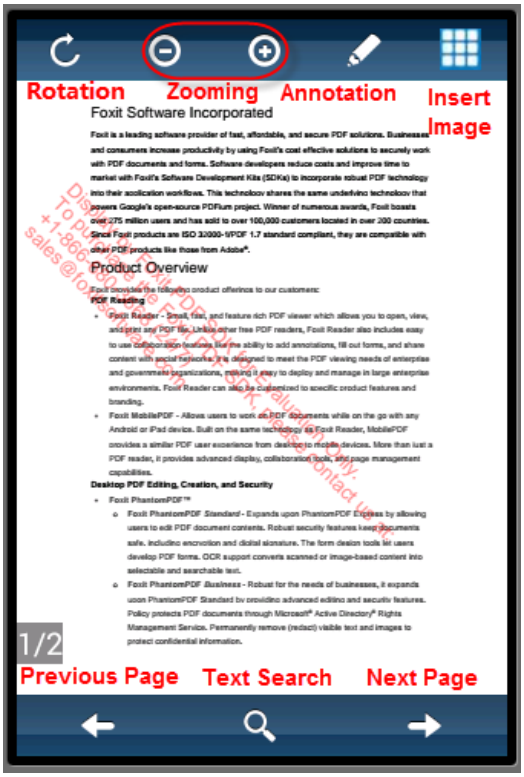

Figure 3-14

c) The viewer demo provides functionalities like rotation, zooming, annotation, page turning, text search and extraction, and inserting an image. Some examples are as follows.

Text Search: click the search button, type word “overview”, and press “Enter” key. The first search result will be highlighted as shown in Figure 3-15.

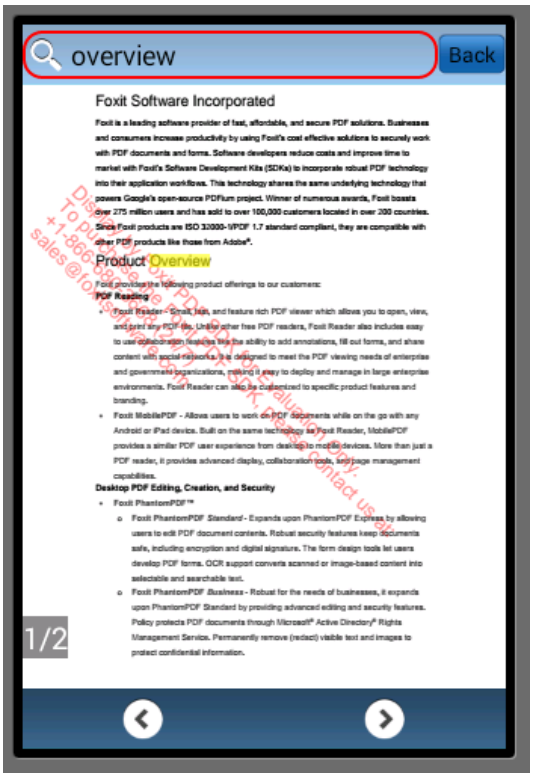

Figure 3-15

Text Extraction: long press the left mouse button, then select a rectangle area. Here, we select the “Foxit Software Incorporated”, and then the texts are extracted as shown in Figure 3-16.

Figure 3-16

Annotation: this demo provides some annotations as shown in Figure 3-17. You can annotate the document by selecting an annotation listed in the menu. In addition, you can export the annotations in the document to external FDF file and import an external FDF file into the document.

For example, select “Link”, click the location that you want to add a link, then input “www.foxitsoftware.com” and click “Save” as shown in Figure 3-18. After that, click the location of the link, it will show a message like Figure 3-19.

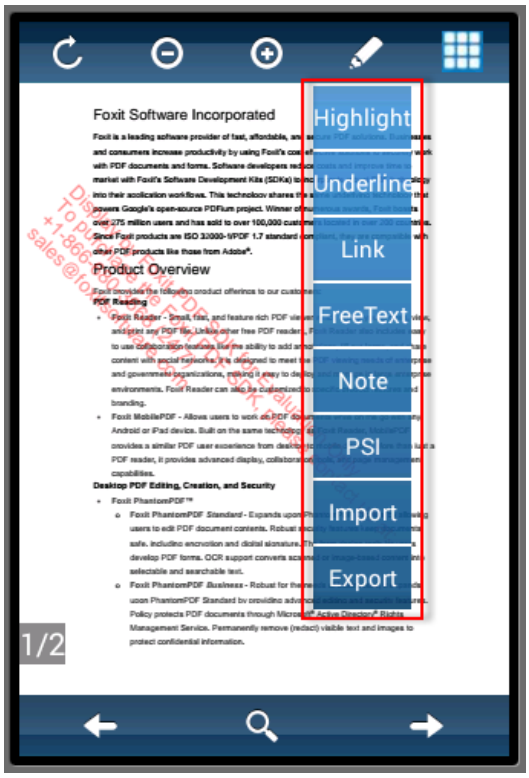

Figure 3-17

Figure 3-18

Figure 3-19

For another example, select “PSI”, then you can draw anything as desired. After drawing, press the “menu” key on the AVD, click the “Confirm” button to exit the PSI mode as shown in Figure 3-20. And then also press the “menu” key on the AVD, click the “Save” button like Figure 3-21 to save the drawing to the document.

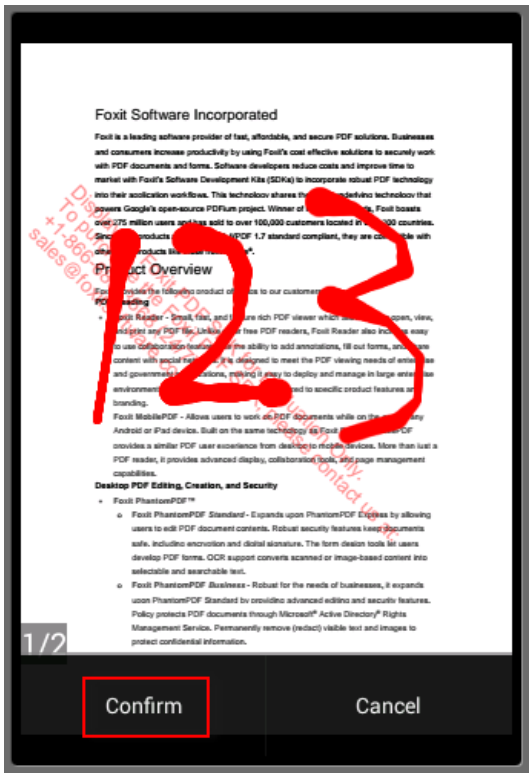

Figure 3-20

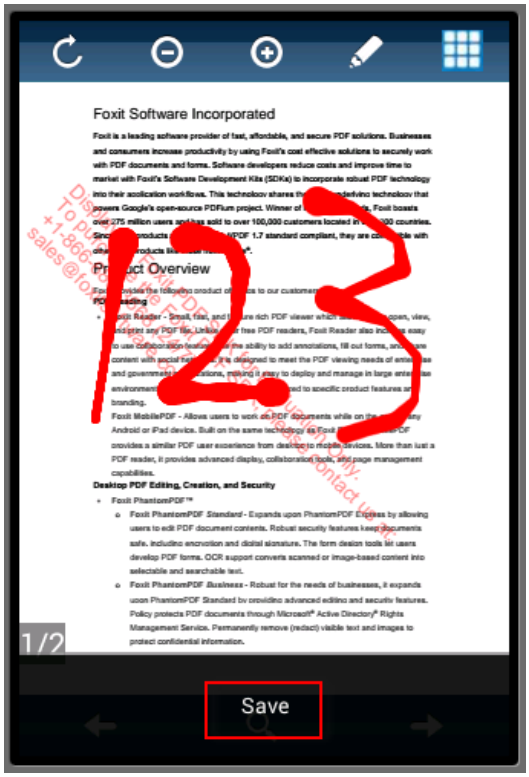

Figure 3-21

Inserting an Image: click the “InsertImage” button, choose the “Foxit-logo.png”, and click the “Confirm” button to exit the selecting window as shown in Figure 3-22. Long press the left mouse button, and then select a rectangle area to display the selected image as shown in Figure 3-23.

Note: *You need to push an image file to the SD card.*

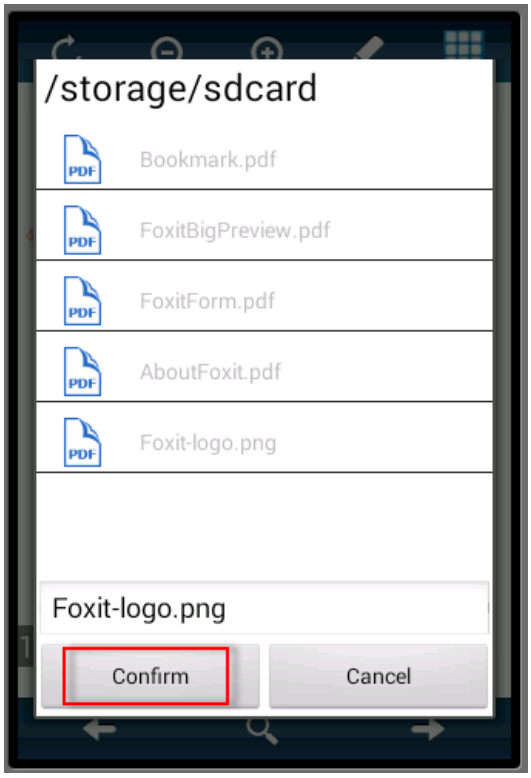

Figure 3-22

Figure 3-23

Form Demo

a) To run this demo in Eclipse, you can refer to the OOM demo. Figure 3-24 shows the form demo running in an AVD targeting 4.4.2.

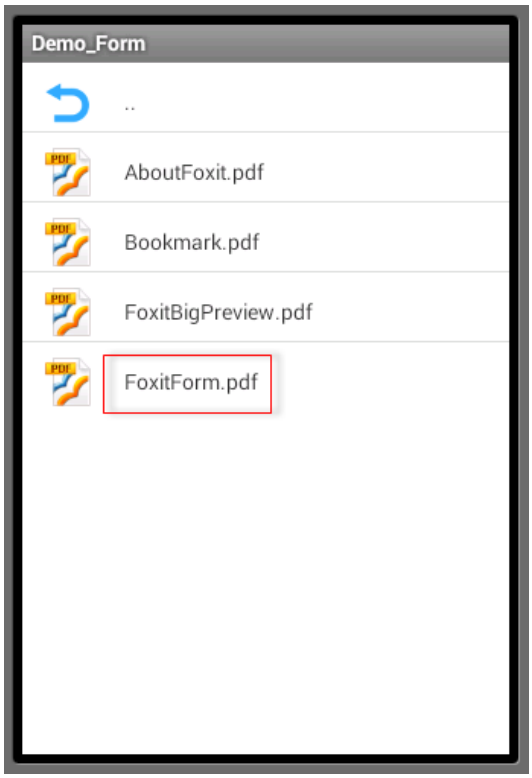

Figure 3-24

b) Click the “FoxitForm.pdf”, and the PDF file will be displayed as shown in Figure 3-25.

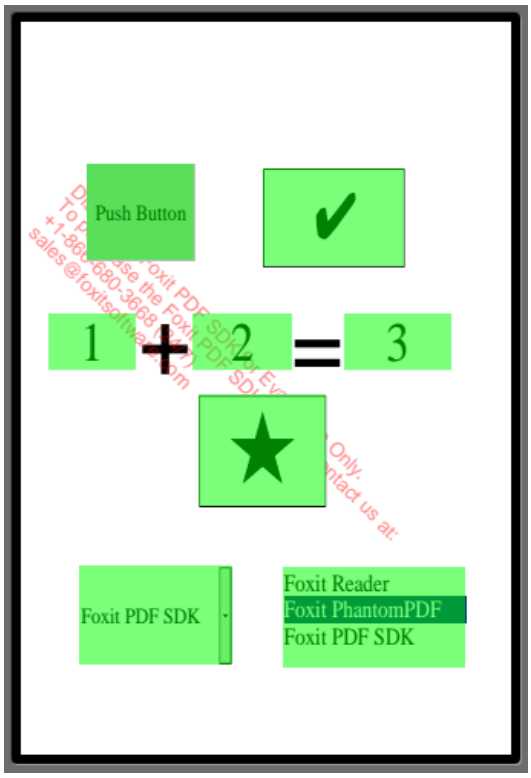

Figure 3-25

c) Fill the form, for example, like Figure 3-26. Press “menu” key on the AVD to show the operation menu as shown in Figure 3-27.

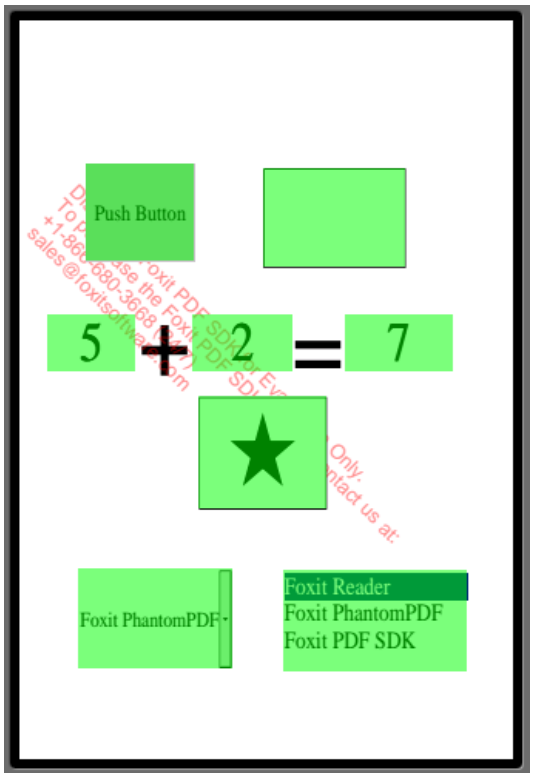

Figure 3-26

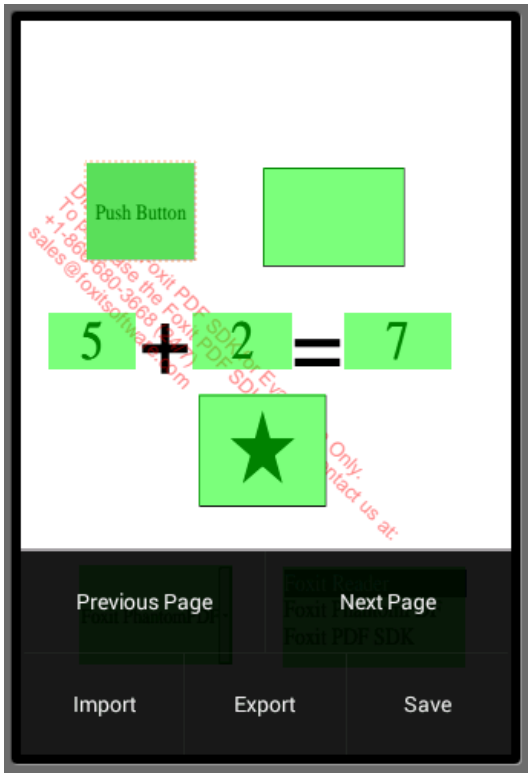

Figure 3-27

For example, select “Export”, input the exported name “formfill.fdf” as shown in Figure 3-28, and then click “OK” to export the form data to “formfill.fdf” file.

Figure 3-28

3.4.2.2 Android Studio Demo

In “samples/android_studio_demo”, there is a viewer demo illustrating how to implement a simple viewer in Android Studio with SDK.

Viewer Demo

To run it in Android Studio, follow the steps below:

a) Import the project into Android Studio following “File -> Import Project…”, and choose the directory where the demo was extracted. The directory structure of the demo will be like Figure 3-29.

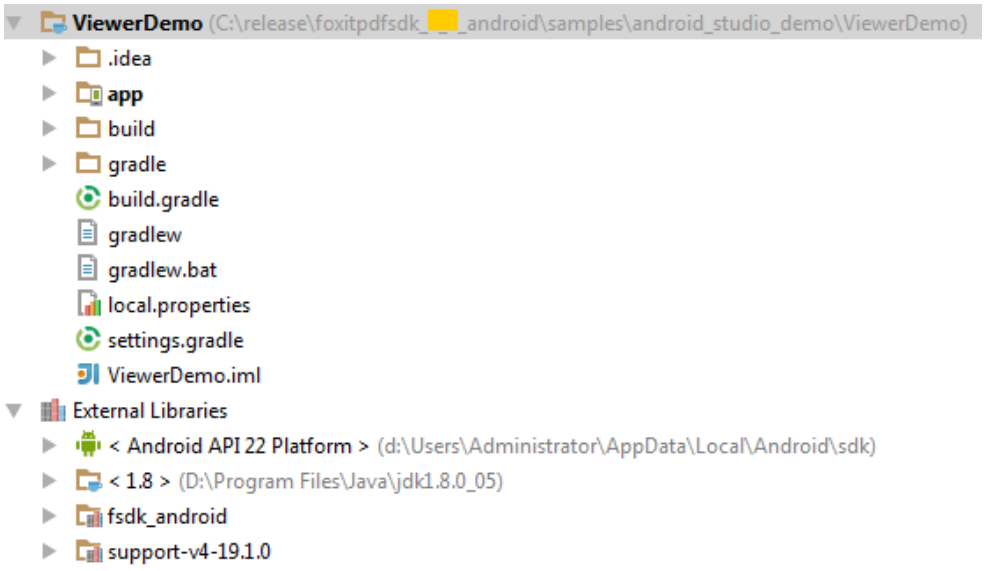

Figure 3-29

b) Launch an Android device or an emulator (AVD). In this guide, we take an AVD targeting Android 4.4.2 as an example. Push the "AboutFoxit.pdf" file under "samples/testfiles" folder to the SD card. The added PDF file will be displayed in "storage/sdcard" as shown in Figure 3-30.

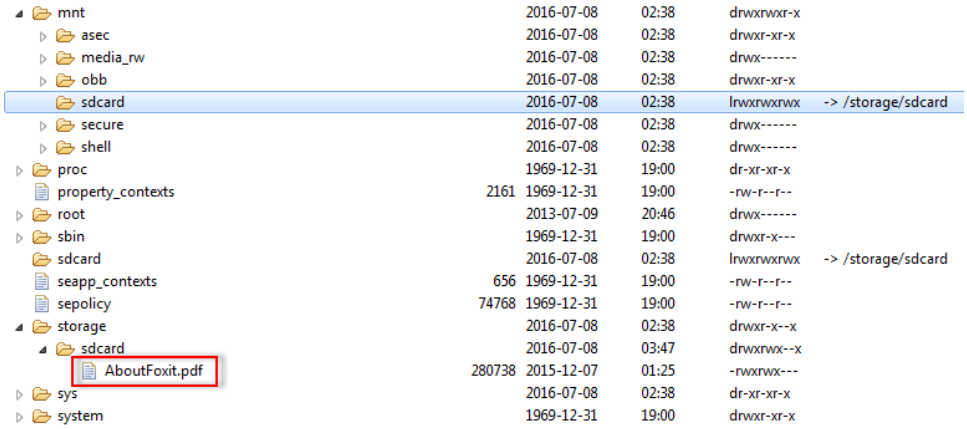

Figure 3-30

c) Click on "Run -> Run 'app'" to run the demo. Figure 3-31 shows the viewer demo running in an AVD targeting Android 4.4.2.

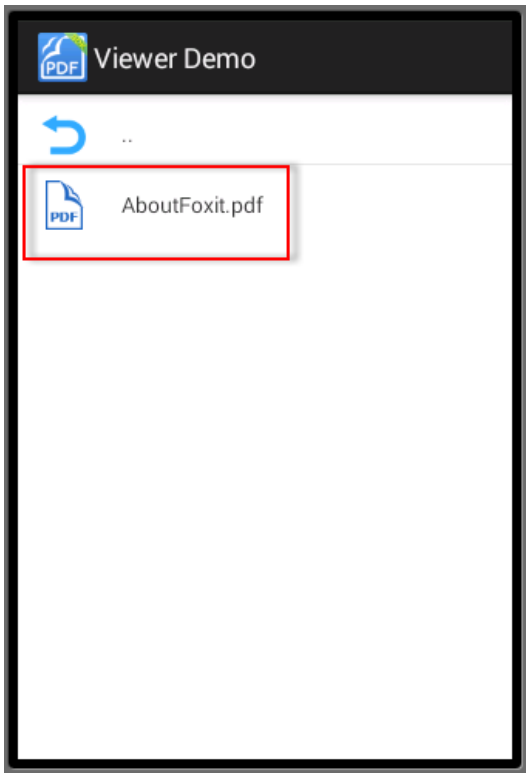

Figure 3-31

d) Click the “AboutFoxit.pdf”, and the PDF file will be displayed as shown in Figure 3-32. The viewer demo provides functionalities like rotation, zooming, annotation, page turning, text search and extraction, and inserting an image. For more details about those functionalities, please refer to the introduction of the viewer demo running in Eclipse.

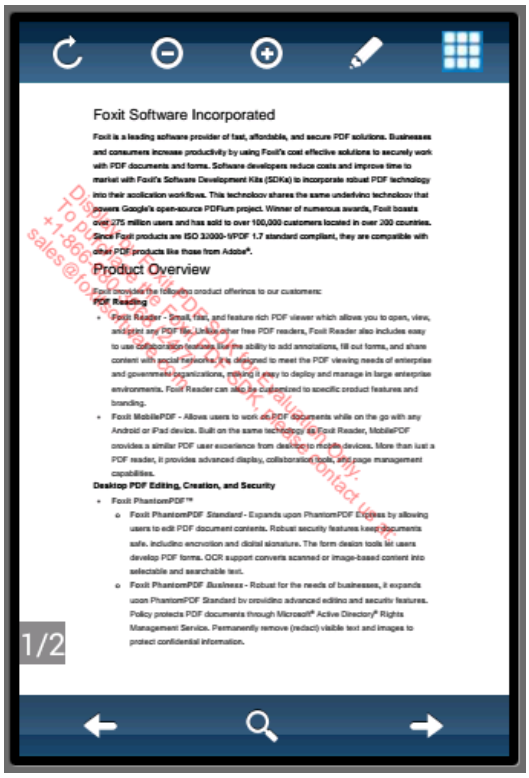
Foxit Software Incorporated
Foxit is a leading software provider of fast, affordable, and secure PDF solutions. Businesses and consumers increase productivity by using Foxit's cost effective solutions to securely work with PDF documents and forms. Software developers reduce costs and improve time to market with Foxit's Software Development Kits (SDKs) to incorporate robust PDF technology into their application workflows. This technology shares the same underlying technology that powers Google's open-source PDFium project. Winner of numerous awards, Foxit boasts over 275 million users and has sold to over 100,000 customers located in over 200 countries. Since Foxit products are ISO 32000-1/PDF 1.7 standard compliant, they are compatible with other PDF products like those from Adobe®.
Product Overview
Foxit provides the following product offerings to our customers:
PDF Reading
Foxit Reader - Small, fast, and feature rich PDF viewer which allows you to open, view, and print any PDF file. Unlike other free PDF readers, Foxit Reader also includes easy to use collaboration features like the ability to add annotations, fill out forms, and share content with social networks. It is designed to meet the PDF viewing needs of enterprise and government organizations, making it easy to deploy and manage in large enterprise environments. Foxit Reader can also be customized to specific product features and branding.
Foxit MobilePDF - Allows users to work on PDF documents while on the go with any Android or iPad device. Built on the same technology as Foxit Reader, MobilePDF provides a similar PDF user experience from desktop to mobile devices. More than just a PDF reader, it provides advanced display, collaboration tools, and page management capabilities.
Desktop PDF Editing, Creation, and Security
Foxit PhantomPDF™
Foxit PhantomPDF Standard - Expands upon PhantomPDF Express by allowing users to edit PDF document contents. Robust security features keep documents safe, including encryption and digital signature. The form design tools let users develop PDF forms. OCR support converts scanned or image-based content into selectable and searchable text.
Foxit PhantomPDF Business - Robust for the needs of businesses, it expands upon PhantomPDF Standard by providing advanced editing and security features. Policy protects PDF documents through Microsoft® Active Directory® Rights Management Service. Permanently remove (redact) visible text and images to protect confidential information.
1/2

Figure 3-32

3.4.2.3 Xamarin Demo

In “samples/xamarin_demo”, there is a simple viewer demo illustrating how to run a simple PDF viewer on Xamarin Studio.

Viewer Demo

To run it on Xamarin Studio, follow the steps below: (in this guide, we run the demo on Mac platform)

a) Import the project into Xamarin Studio following “File -> Open…”, and choose the “sdk_demo.sln” under “samples\xamarin_demo” folder. The directory structure of the demo will be like Figure 3-33.

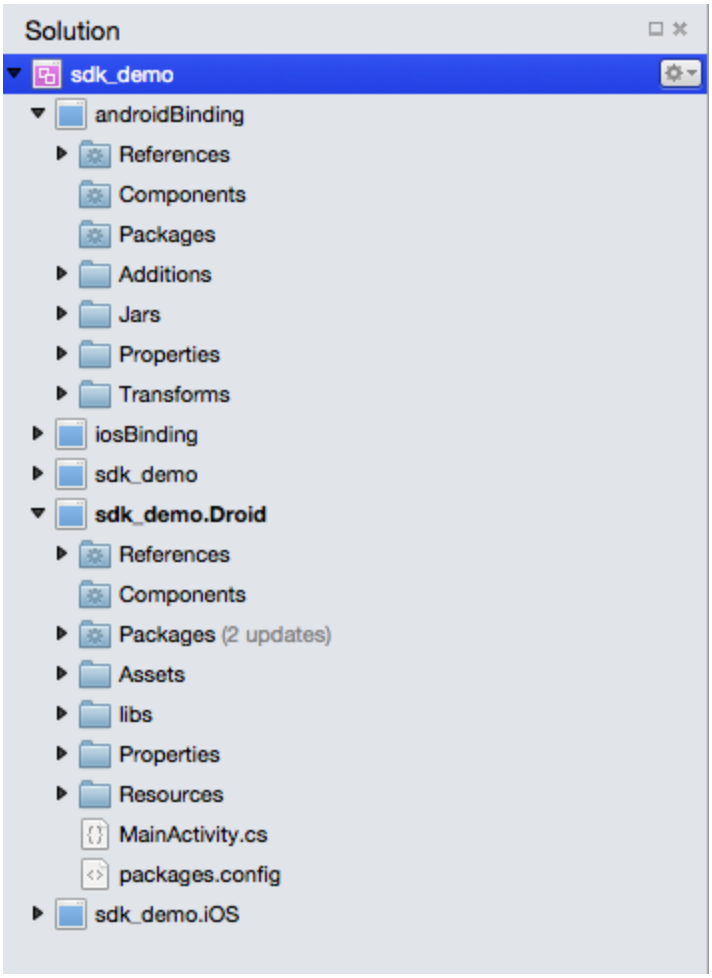

Figure 3-33

Note: *This demo includes the project codes of both Android platform and iOS platform, but the downloaded Foxit PDF SDK for Android Java APIs package only contains the SDK library for Android to run the Xamarin Android project “sdk_demo.Droid” on Xamarin Studio. If you want to*

run the Xamarin iOS project "sdk_demo.iOS", please put the SDK library for iOS to the corresponding directory.

b) Launch an Android device or an emulator (AVD). In this guide, we take an AVD targeting Android 4.4.2 as an example. Push the "AboutFoxit.pdf" file under "samples/testfiles" folder to the SD card. The added PDF file will be displayed in "storage/sdcard" as shown in Figure 3-34.

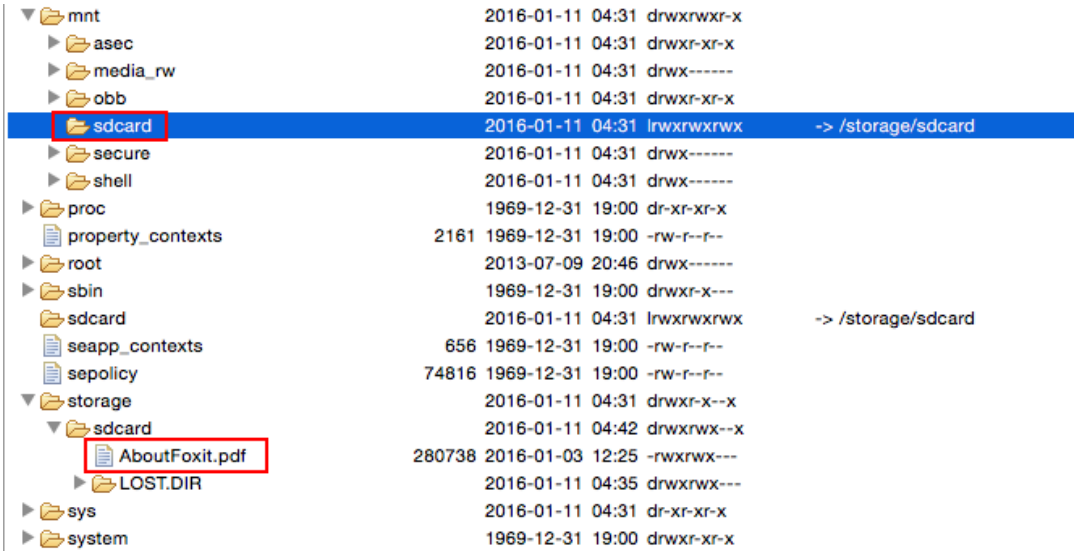

Figure 3-34

c) Right click the project "sdk_demo.Droid" and select "Set As Startup Project" as shown in Figure 3-35.

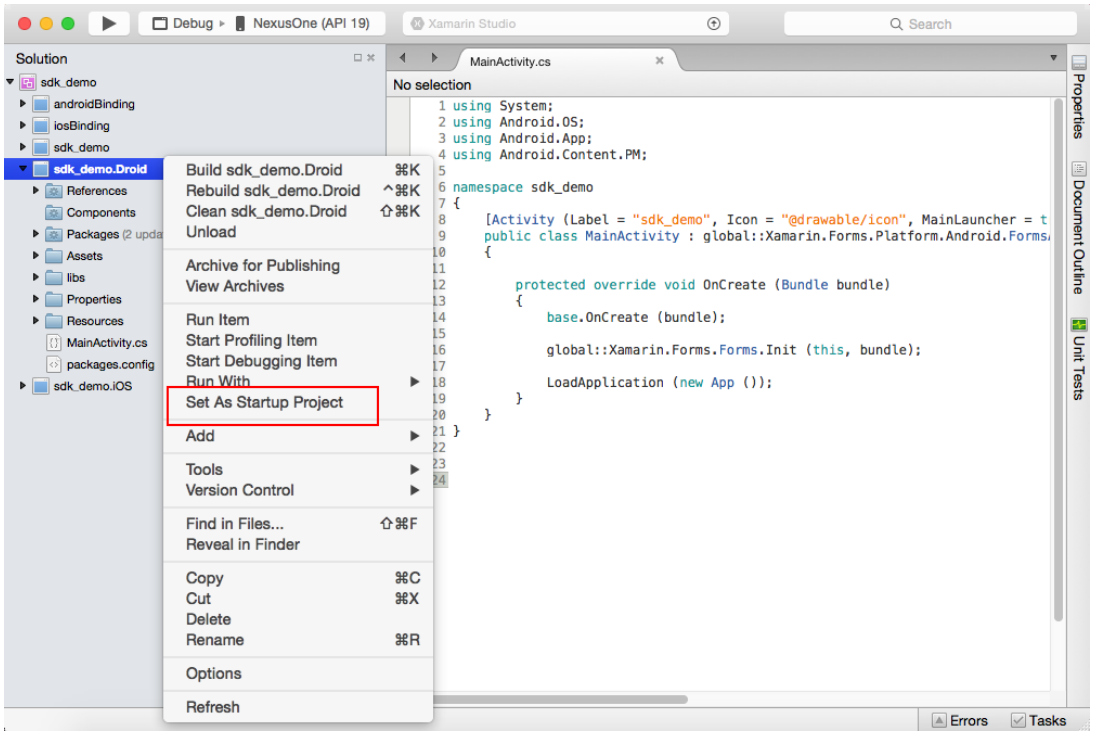

Figure 3-35

d) Click on ▶ button or "Run -> Start Debugging" to run the demo. The "AboutFoxit.pdf" file will be displayed as shown in Figure 3-36.

Note: *The "androidBinding" project used for binding "fsdk_android.jar" package will be compiled automatically first when running the "sdk_demo.Droid" project.*

Please notice that the "fsdk_anroid.jar" package and "libfsdk_android.so" library under "libs" folder required for the demo are loaded to "androidBinding\Jars" and "sdk_demo.Droid\libs\armeabi-v7a" folders respectively with relative path. So, if you change their location, please add them to the corresponding folders manually.

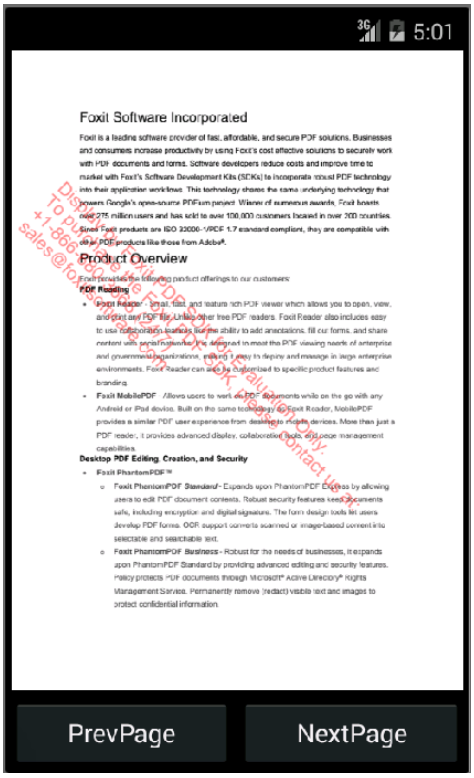

Figure 3-36

e) The simple viewer demo only provides page turning feature. Click on “NextPage” button to turn to the next page as shown in Figure 3-37.

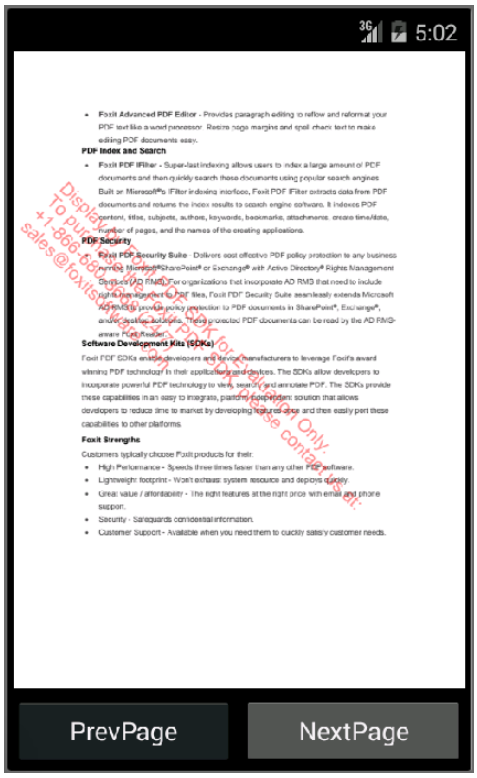

Figure 3-37

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. An example of applying a license with hardcode method is 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

```
static{
	System.loadLibrary("fsdk_android");
}

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 fields

```
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 Form Filler

Form filler is the most commonly used feature for users. Form filler allows applications to fill forms dynamically. The key point for applications to fill forms is to construct some callback functions for PDF SDK to call. Those functions are defined in **FormActionHandler** class.

Example:

4.10.1 How to fill a form with form filler.

```
try {
        hasForm = pdfDocument.hasForm();
        if(hasForm)
        {
                form = pdfDocument.loadForm();
                PDFFormFiller.setActionHandler(pdfDocument.getHandle(), formAction);
                formFiller = form.beginFormFiller();
                formFiller.setHighlightColor(PDFFormField.TYPE_PUSHBUTTON, 0x8000ff00);
                formFiller.setHighlightColor(PDFFormField.TYPE_CHECKBOX, 0x8000ff00);
                formFiller.setHighlightColor(PDFFormField.TYPE_RADIOBUTTON, 0x8000ff00);
                formFiller.setHighlightColor(PDFFormField.TYPE_COMBOBOX, 0x8000ff00);
                formFiller.setHighlightColor(PDFFormField.TYPE_LISTBOX, 0x8000ff00);
                formFiller.setHighlightColor(PDFFormField.TYPE_TEXTFIELD, 0x8000ff00);
                formFiller.showHighlight(true);
                pdfDocument.initiateJavaScript();
                form.endFormFiller();
        }
} catch (PDFException e) {
                // TODO Auto-generated catch block
        e.printStackTrace();
}
```

4.11 Form Design

Fillable PDF forms (AcroForm) are especially convenient for preparation of various applications, such as taxes and other government forms. Form design provides APIs to add or remove form fields (Acroform)

to or from a PDF file. Designing a form from scratch allows developers to create the exact content and layout of the form they want.

Example:

4.11.1 How to add a text form field to a PDF document.

```
try{
       RectF rect = new RectF();
       rect.right = 50;
       rect.top = 50;
       PDFForm form = document.createForm();
       PDFFormField field = new PDFFormField("Text Field", PDFFormField.TYPE_TEXTFIELD);
       PDFFormControl control = form.addField(page, field, rect);
       control.setBorderColor(0xff00ff00);
       control.setName("Text Field ");
       control.setContents("Text Field contents ");
       control.resetAppearance();
} catch (PDFException e) {
              // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.11.2 How to remove a text form field from a PDF

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

4.12 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:

4.12.1 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();
}
```

4.12.2 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();
}
```

4.13 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:

4.13.1 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);
        Bitmap bmp = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
        bmp.eraseColor(Color.WHITE);

        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();
        ...
}
```

4.13.2 How to convert png file to PDF file

```
try {

        PDFDocument document = PDFDocument.create();
        PDFPage page = null;
        page = pdfDocument.createPage(0);
        page.setSize(500, 600);
        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();
}
```

4.14 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:

4.14.1 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();
}
```

4.14.2 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();
}
```

4.15 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:

4.15.1 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();
```

4.16 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:

4.16.1 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();
}
```

4.17 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:

4.17.1 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();
        }
}
```

4.17.2 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();
}
```

4.17.3 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();
}
```

4.17.4 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();
}
```

4.18 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, generating a template PDF on demand, creating path object and set its properties to achieve the feature like Foxit PSI.

Example:

4.18.1 How to create an image object in a PDF page

```
//Assuming PDFPage page and Bitmap bitmap have 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();
}
```

4.18.2 How to create a path object and set its properties

```
//Assuming PDFPage page and PDFPath path have been created.
try {
     PathObject pathObj = PathObject.create(page);
     pathObj.setPathData(path);
     pathObj.setFillMode(PathObject.FILLMODE_ALTERNATE);
     pathObj.setStrokeState(false);
     pathObj.setFillColor(Color.GREEN);
     pathObj.setStrokeColor(Color.RED);
     pageObjects.insertObject(PageObject.TYPE_PATH, 0, pathObj);
     pageObjects.generateContents();
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.19 Marked content

In PDF document, a portion of content can be marked as marked content element. Marked content helps to organize the logical structure information in a PDF document and enables stylized tagged PDF. Tagged PDF has a standard structure types and attributes that allow page content to be extracted and reused for other purposes. More details about marked content could be found in chapter 10.5 of PDF reference 1.7 [1].

Example:

4.19.1 How to get marked content in a page and get the tag name

```
//Assuming PDFPage page has been created.
try {
       PageObjects pageobjects = page.getPageObjects();
       int count = pageobjects.countObjects(PageObject.TYPE_ALL);
       PageObject pageobject = pageobjects.getObject(PageObject.TYPE_ALL, 1);
       MarkedContent mc = pageobject.getMarkedContent();
       String name = mc.getTagName(0);
       …
}
catch (PDFException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
```

4.20 Layer

PDF Layers, in other words, Optional Content Groups (OCG), are supported in Foxit PDF SDK. Users can selectively view or hide the contents in different layers of a multi-layer PDF document. Multi-layers are widely used in many application domains such as CAD drawings, maps, layered artwork and multi-language document, etc. **PDFDocument.createLayerContext(int)** can create a view layer context with a given type. **PDFDocument.enumLayers()** could enumerate all PDF layers of a PDF document. Foxit PDF SDK provides APIs to view or hide the contents in different layers, and set layers' name

Example:

4.20.1 How to traverse layer tree and set the opposite visible state of every PDF layer

```
Layer layer = null;
LayerNode layerNode = null;
layerNode = document.enumLayers();
LayerContext context = document.createLayerContext(LayerContext.USAGETYPE_VIEW);

Public void revertlayertree(LayerNode layerNode, LayerContext context)
{
    LayerNode nextNode = null;
    try {
       int childCount = layerNode.countChildren();
       nextNode = layerNode.getChildren(0);
       if(childCount == 0 && nextNode == null)
       {
              layer = layerNode.getLayer();
              if(layer != null)
              {
                     Boolean visible = layerContext.isVisible(layer);
                     context.setVisible(layer, !visible);
              }
       }
       else
              for(int j = 0; j < childCount; j++)
              {
                     nextNode = layerNode.getChildren(j);
                     layerName = nextNode.getName();
                     revertlayertree(nextNode, layerContext);
              }
    }
    catch (PDFException e) {
       e.printStackTrace();
       }
}
```

4.21 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:

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

```
WatermarkSetting settings = new WatermarkSetting();
settings.flags = PDFWatermark.FLAG_ONTOP;
settings.offsetX = 1.0f;
settings.offsetY = 1.0f;
settings.opacity = 100;
settings.position = PDFWatermark.POS_BOTTOMCENTER;
settings.rotation = 0.0f;
settings.scaleX = 1.0f;
settings.scaleY = 1.0f;
try {
	FileHandler handler = FileHandler.create(fileName,   FileHandler.FILEMODE_READONLY);
	pdfDocument = PDFDocument.open(handler, password.getBytes());
	Bitmap bitmap = Bitmap.createBitmap(400, 400, Bitmap.Config.ARGB_8888);
	watermark=PDFWatermark.create(doc,bitmap,settings);
	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(0);
			}
				progress.release();
		}
	}
	watermark.insertToPage(page);
	doc.closePage(page);
	watermark.release();
	FileHandler outputFile=FileHandler.create(savePath, FileHandler.FILEMODE_TRUNCATE);
	Progress progress = doc.startSaveToFile(outputFile, PDFDocument.SAVEFLAG_NOORIGINAL);
	if (progress != null)
	{
		int ret = Progress.TOBECONTINUED;
		while (ret == Progress.TOBECONTINUED)
		{
			ret = progress.continueProgress(30);
		}
```

```
        }
        progress.release();
        outputFile.release();
        doc.close();
        handler.release();
}
catch (PDFException e) {
        e.printStackTrace();
        return;
}
```

4.21.2 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;
}
```

4.21.3 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;
}
```

4.21.4 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;
}
```

4.22 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:

4.22.1 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;
```

4.22.2 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;
}
```

4.22.3 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;
}
```

4.23 RMS

RMS (Right Management Service) module can be used to encrypt and decrypt PDF documents with Microsoft RMS Encryption/Decryption. Foxit PDF SDK provides APIs to integrate with the third-party security mechanism (Microsoft RMS). These APIs allow developers to work with the Microsoft RMS SDK to both encrypt (protect) and decrypt (unprotect) PDF documents.

Example:

4.23.1 How to encrypt a PDF document with Microsoft RMS

```
FileHandler fileHInput = FileHandler.create((inputFile), FileHandler.FILEMODE_READONLY);
PDFDocument pdfDoc = PDFDocument.open(fileHInput, null);
RMSEncryptionParams rmsEncryptParam = new RMSEncryptionParams();
try {
        rmsEncryptParam.setEncryptMetadata(encryptMetadata);
        rmsEncryptParam.setIrmVersion(irmVersion);
//"pubLicInfo" and "serverEulList" should be compressed by Flate compression algorithm
//at first, and then encoded by Base-64 Encryption algorithm.
        rmsEncryptParam.setPublishLicense(pubLicInfo);
        rmsEncryptParam.setServerEulList(serverEulList);
} catch (PDFException e) {
        e.printStackTrace();
        pdfDoc.close();
        fileHInput.release();
        return;
}
//Implement a class extending from "SecurityHandler", and in this class, user should finish
abstract functions in class SecurityHandler to do the encryption and decryption of Microsoft
RMS.
MyRMSSecurityHandler securityHandler = new MyRMSSecurityHandler();
```

```
PDFDocument.registerSecurityHandler("MicrosoftIRMServices", securityHandler);
FileHandler fileHEncrypt = FileHandler.create((outputFile), FileHandler.FILEMODE_TRUNCATE);
Progress encryptProgress = null;
try {
	encryptProgress = pdfDoc.startEncryption(rmsEncryptParam, 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;
```

4.23.2 How to decrypt a PDF document with Microsoft RMS

```
try {
	MySecurityHandler securityHandler = new MySecurityHandler();
	PDFDocument.registerSecurityHandler("MicrosoftIRMServices", securityHandler);
	RMSEncryptionParams rmsEncryptParam = new RMSEncryptionParams();
	boolean encryptMetadata = true;
	String orgPubLic = new String("RMS PublicLicense Info");
	String []orgServerEulList = new String[4];
	orgServerEulList[0] = new String("TESTID-001");
	orgServerEulList[1] = new String("For testing RMS java API");
	orgServerEulList[2] = new String("TESTID-002");
	orgServerEulList[3] = new String("For testing RMS java API-2");
	int irmVersion = 0;
	String pub = FlateAndEncodeData(orgPubLic);
	rmsEncryptParam.setPublishLicense(pub);
	rmsEncryptParam.setServerEulList(orgServerEulList);
	rmsEncryptParam.setEncryptMetadata(true);

	rmsEncryptParam.setIrmVersion(irmVersion);
	FileHandler handler = FileHandler.create(inputFilePath + " FoxitText.pdf",
FileHandler.FILEMODE_READONLY);
	PDFDocument pdfDoc = PDFDocument.open(handler, null);
	FileHandler encryptedFile = FileHandler.create(inputFilePath + "rms_encrypt.pdf",
FileHandler.FILEMODE_TRUNCATE);
	Progress progress = null;

	progress = pdfDoc.startEncryption(rmsEncryptParam, encryptedFile,
```

```
PDFDocument.SAVEFLAG_NOORIGINAL);
        progress.continueProgress(0);
        progress.release();
        encryptedFile.release();
        pdfDoc.close();
        handler.release();
        FileHandler handler = FileHandler.create(inputFilePath + " rms_encrypt.pdf",
FileHandler.FILEMODE_READONLY);
        pdfDoc = PDFDocument.open(handler, null);

        FileHandler dehandler = FileHandler.create(inputFilePath + "rms_deEnc.pdf",
FileHandler.FILEMODE_TRUNCATE);

        progress = pdfDoc.startSaveToFile(dehandler, PDFDocument.SAVEFLAG_REMOVESECURITY);
        progress.continueProgress(0);
        progress.release();
        dehandler.release();
        ...
}
catch (PDFException e){
        e.printStackTrace();
}
```

4.24 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 default signature supports signing with time stamp through digital time stamp service (DTS). The custom signature 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:

4.24.1 How to sign the PDF document with default signature handler

```
FileHandler fileHInput = FileHandler.create((inputFile), FileHandler.FILEMODE_READONLY);
PDFDocument document = PDFDocument.open(fileHInput, null);
Signature.registerDefaultHandler();
PDFPage page = document.getPage(0);
RectF rect = new RectF(100,100,200,200);
signature = page.addSignature(rect);

TSAClient tsa = new TSAClient(TSA_SERVER, userword, password);
KeyStoreInfo ksInfo = new KeyStoreInfo(KeyStoreInfo.TYPE_PKCS12, inputFilePath +
"foxit_all.pfx", "123".getBytes());

signature.setKeyStoreInfo(ksInfo);
signature.setTSAClient(tsa);
signature.setDefaultContentsLength(20000);
signature.initValue();
signature.setFilter("Adobe.PPKLite");
signature.setSubFilter("adbe.pkcs7.detached");
signature.setSigner("default_sign");
signature.setAppearanceFlags(Signature.APPEARANCE_NAME);
signature.resetAppearance();

FileHandler signedFile = FileHandler.create(outputFilePath + "Signer_default.pdf",
FileHandler.FILEMODE_TRUNCATE);
Progress progress = signature.startSign(signedFile);
if(progress != null)
{
       int ret = Progress.TOBECONTINUED;
       while (ret == Progress.TOBECONTINUED)
       {
        ret = progress.continueProgress(0);
       }
       progress.release();
}
signedFile.release();
document.closePage(page);
document.close();
document = null;
fileHInput.release();
fileHInput = null;
```

4.24.2 How to sign the PDF document with custom signature handler

```
MySignatureHandler handler = new MySignatureHandler();
Signature.registerSignatureHandler("Third-part", ”Custom”, handler);
FileHandler fileHInput = FileHandler.create((inputFile), FileHandler.FILEMODE_READONLY);
PDFDocument document = PDFDocument.open(fileHInput, null);

PDFPage page = document.getPage(0);

RectF rect = new RectF(100,100,200,200);
signature = page.addSignature(rect);signature.initValue();
signature.setFilter("Third-part");
signature.setSubFilter("Custom");
signature.setAppearanceFlags(Signature.APPEARANCE_FOXITFLAG);
signature.resetAppearance();
FileHandler signedFile = FileHandler.create(outputFilePath+"Sign_custom.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.closePage(page);
document.close();
document = null;
fileHInput.release();
fileHInput = null;
```

Note: *Users need to implement signature callback class of signing like* ***MySignatureHandler*** *class as mentioned above.*

5 FAQ

1. **What is the price of Foxit PDF SDK for Android?**
   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 Android?**
   There are detailed descriptions on how to apply a license in the section 3.3. 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 Android?**
   You can send email to support@foxitsoftware.com for any questions or comments or call our support at 1-866-693-6948.

4. **How can I do if I want to develop applications on the devices that do not support Neon?**
   Foxit provides another Android SDK library that allows developers to use it on the devices that do not support Neon, you can contact us at support@foxitsoftware.com if needed.

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 Android 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