



DEVELOPER GUIDE

FOR FOXIT PDF SDK 4.1

FOR ANDROID

2014-8-20

FOXIT SOFTWARE INCORPORATED
42840 CHRISTY STREET, SUITE 201
FREMONT, CA 94538, USA

TABLE OF CONTENTS

1	Introduction to Foxit PDF SDK 4.1	1
1.1	Features.....	1
1.1.1	Evaluation.....	1
1.1.2	License.....	2
1.1.3	Out of Memory (OOM).....	2
1.2	System Requirements on Android	2
1.3	About this guide	2
2	Introduction to PDF	3
2.1	History of PDF.....	3
2.2	PDF Document Structure	3
3	Getting Started	4
3.1	What's in the Package	4
3.2	How to apply a license	8
3.3	How to run a demo	9
3.3.1	Demo Environment	9
3.3.2	Setting up and running demo project	9
4	Working with SDK API	19
4.1	Common data structures and operations	19
4.2	Load Library.....	20
4.3	File	20
4.4	Document.....	21
4.5	Attachment	22
4.6	Page.....	23

4.7	Render	24
4.8	Text Page	26
4.9	Text Link	27
4.10	Form	28
4.11	Annotations.....	29
4.12	Image Conversion.....	31
4.13	Bookmark	32
4.14	Reflow	34
4.15	Pressure Sensitive Ink	35
4.16	PDF Action	36
4.17	Page Object	38
5	FAQ.....	39
	References	40
	Support	41
	Glossary of Terms & Acronyms.....	42

1 INTRODUCTION TO FOXIT PDF SDK 4.1

1.1 Features

Foxit PDF SDK 4.1 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 4.1 has 12 main features. They help application developers focus on functions that they really need and reduce the development cost.

Features

PDF Document	Open and close files, set and get metadata
PDF Page	Parse, render, read and set the properties of a page
Render	Graphics engine created on a bitmap for platform graphics device
PDF Text Page	Text processing in a PDF document
Text Link	Extract URL formatted link
Bookmark	Directly locate and link to point of interest within a document
Image Conversion	Convert between PDF files and images(BMP,TIF,JPX,JPG,PNG), and GIF to PDF
Annotation	Create, edit and remove annotations
Form	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
Out Of Memory	recover from an OOM condition

1.1.1 Evaluation

Foxit PDF SDK allows users to download trial version to evaluate SDK. The trial version has no difference with a standard version except for the 30-day limitation and trail watermarks in the generated PDF pages and images. After evaluation period, customers should contact Foxit sales team to purchase licenses for SDK if they want to continue using it.

1.1.2 License

It is required for customers to explicitly call java method to apply the license. It grants users to release their applications based on SDK 4.1 libraries. However, users are prohibited to distribute any documents, sample code, or source code in the SDK 4.1 released package to any third party without the permission from Foxit Software Incorporated. Users are also prohibited to develop competing applications against Foxit products based on SDK 4.1.

1.1.3 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.

1.2 System Requirements on Android

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

1.3 About this guide

This guide aims at introducing installation package structure on Android platform, basic knowledge on PDF and the usage of SDK. The targeted audience should be those who wants to develop PDF applications while lacks specialized knowledge on PDF.

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.

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 What's in the Package

Download Foxit PDF SDK for Android Java APIs and untar it to a directory "foxitpdfsdk_4_1_Android".The structure of the release package is show in Figure 3-1. This package contains the following folders:

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

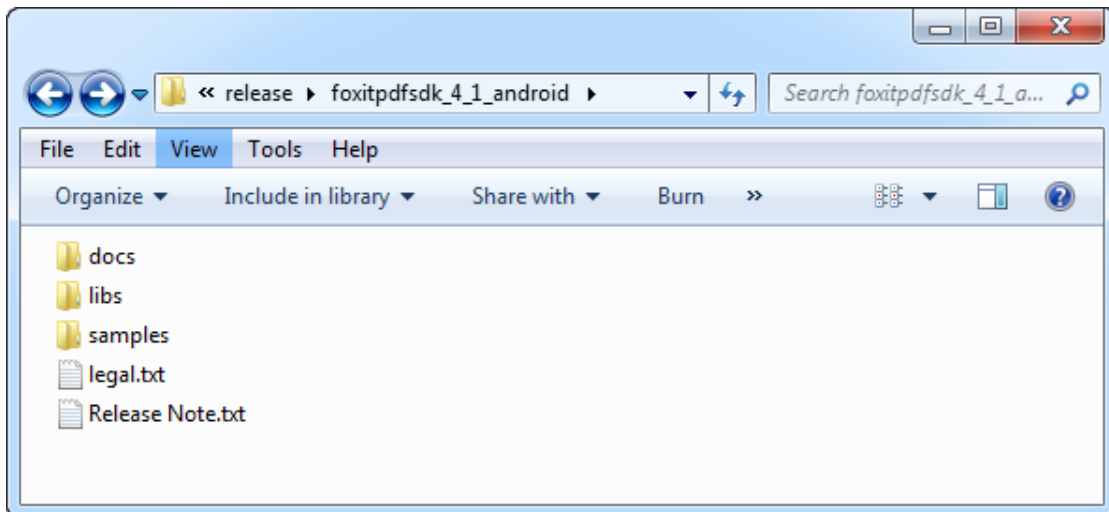


Figure 3-1

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

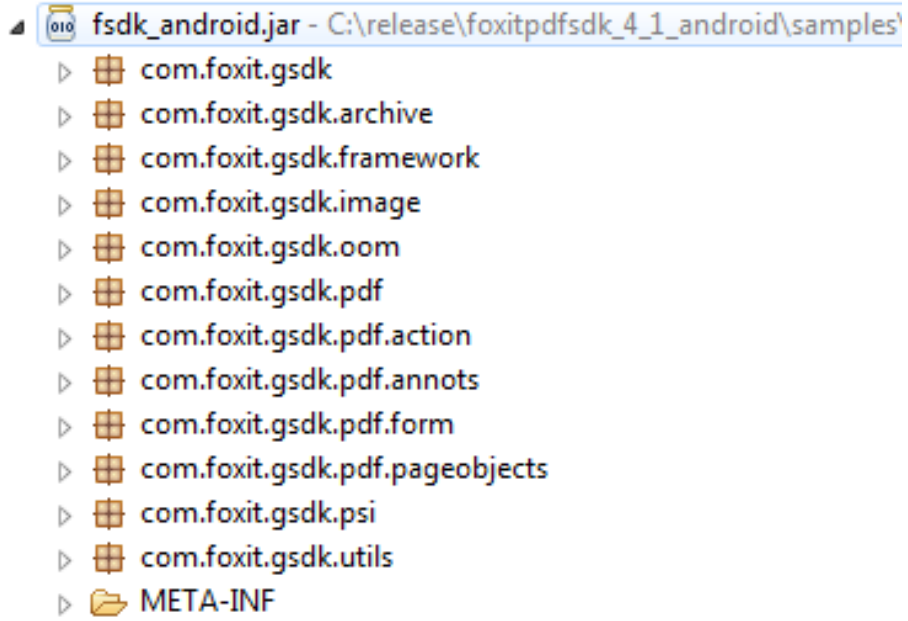


Figure 3-2

Table 3-1

Java Package	Java Class
Com.foxit.gsdk	IApp.class IInvalidate.class PDFException.class PDFLibrary.class
com.foxit.gsdk.archive	Archive.class
com.foxit.gsdk.framework	ParamInterface.class
com.foxit.gsdk.image	Image.class ImageFile.class
com.foxit.gsdk.oom	OOM.class

Java Package	Java 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 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 PDFJavascriptAction.class PDFLaunchAction.class PDFNamedAction.class PDFRemoteGotoAction.class PDFResetFormAction.class PDFSubmitFormAction.class PDFURIAction.class

Java Package	Java Class
com.foxit.gsdk.pdf.form	FormActionHandler.class PDFForm.class PDFFormControl.class PDFFormField.class PDFFormFiller.class
com.foxit.gsdk.pdf.pageobjects	ImageObject.class PageObject.class PageObjects.class
com.foxit.gsdk.psi	PSI.class
com.foxit.gsdk.utils	DateTime.class FileHandler.class Size.class SizeF.class

Java Package	Java 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	

3.2 How to apply a license

It is necessary for applications to 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();  
}
```

3.3 How to run a demo

3.3.1 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/ libfsdk_android.so (or libs/x86/ 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 – operates on the Java layer used by the Virtual Machine. They provide all the functionalities of our PDF library.

3.3.2 Setting up and running demo project

Download and install Eclipse IDE (<http://www.eclipse.org/>) and Android SDK (<http://developer.android.com/sdk/index.html>).

In “samples/view_demo”, there are a viewer demo and an OOM handing demo illustrating how to implement a simple viewer and how to handle OOM with SDK respectively. The demos are shown in Figure 3-3.

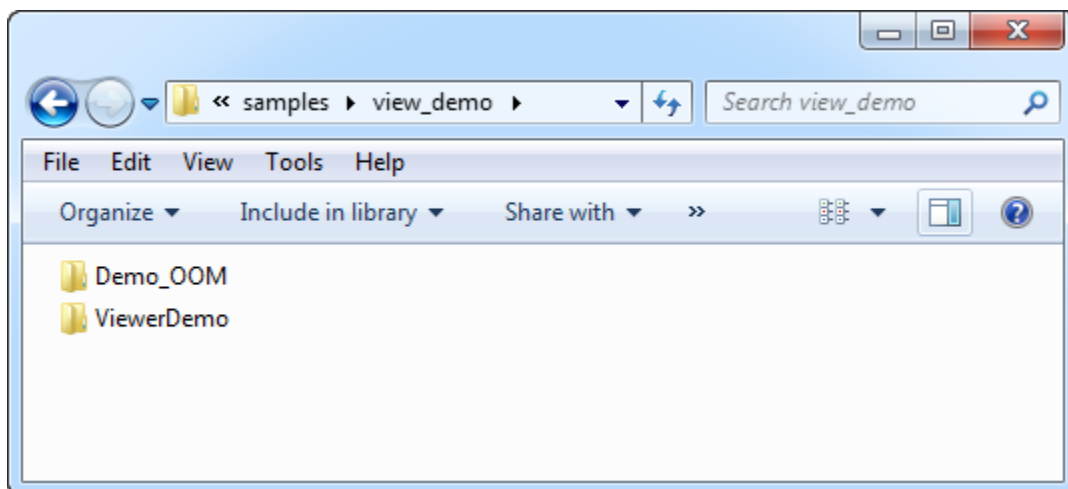


Figure 3-3

OOM Demo

This demo provides an example for handling OOM using PDF SDK APIs. To run it in Eclipse,

- a) Import the project into Eclipse following “File->Import-> Existing Project into workspace”, and choose the directory where the demo was extracted by “Browse”. The directory structure of the demo will be like Figure 3-4.

One thing worth noting is 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, “armeabi” and “x86” folders in directory “foxitpdfsdk_4_1_android/libs” to “Demo_OOM/libs” in the workspace.

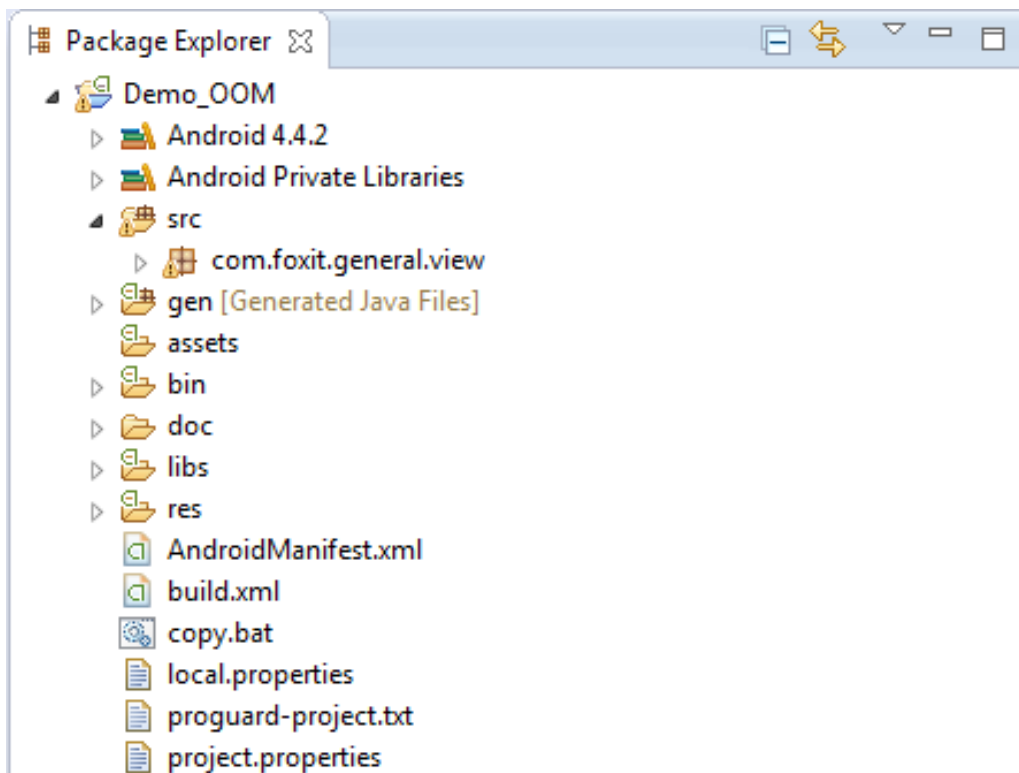


Figure 3-4

- b) Push the PDF files in “samples/testfiles” to the SD card (\mnt\sdcard), or the demo may not be tested. Three PDF files will be displayed in “storage/sdcard” as shown in Figure 3-5.

mnt	2014-08-13	03:50	drwxrwxr-x	
├─ asec	2014-08-13	03:50	drwxr-xr-x	
├─ media_rw	2014-08-13	03:50	drwx-----	
├─ obb	2014-08-13	03:50	drwxr-xr-x	
└─ sdcard	2014-08-13	03:50	lrwxrwxrwx	-> /storag...
├─ secure	2014-08-13	03:50	drwx-----	
├─ shell	2014-08-13	03:50	drwx-----	
├─ proc	1969-12-31	19:00	dr-xr-xr-x	
├─ property_contexts	2161	1969-12-31	19:00	-rw-r--r--
├─ root	2013-07-09	20:46	drwx-----	
├─ sbin	1969-12-31	19:00	drwxr-x---	
├─ sdcard	2014-08-13	03:50	lrwxrwxrwx	-> /storag...
├─ seapp_contexts	656	1969-12-31	19:00	-rw-r--r--
├─ sepolicy	74768	1969-12-31	19:00	-rw-r--r--
├─ storage	2014-08-13	03:50	drwxr-x--x	
├─┬─ sdcard	2014-08-13	04:08	drwxrwx--x	
├─┬─ AboutFoxit.pdf	280738	2014-08-13	04:09	-rwxrwx---
├─┬─ FoxitBigPreview.pdf	324629	2014-08-10	09:23	-rwxrwx---
├─┬─ FoxitBookmark.pdf	81640	2014-08-10	08:18	-rwxrwx---
├─ sys	2014-08-13	03:50	dr-xr-xr-x	
└─ system	1969-12-31	19:00	drwxr-xr-x	

Figure 3-5

- 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 3-6 shows the demo running in an AVD targeting 4.4.2. Here we set the memory size to 20(MB).

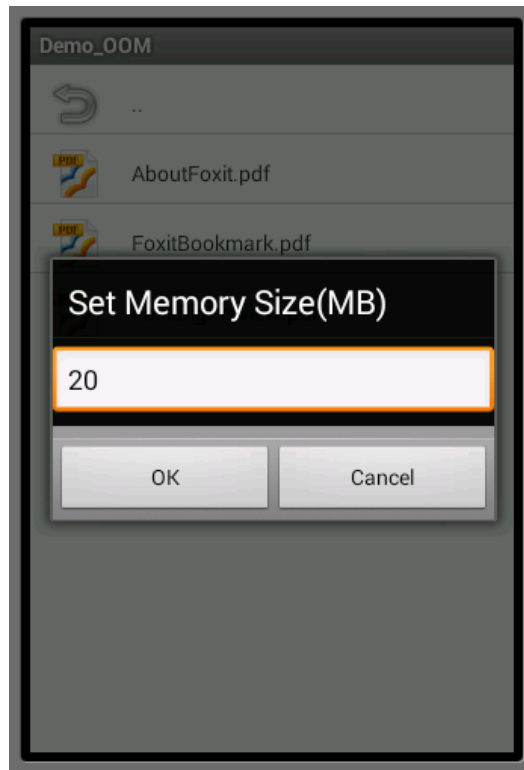


Figure 3-6

- d) After setting the memory size to 20, click on “OK”, then choose the “FoxitBigPreview.pdf” as shown in Figure 3-7.

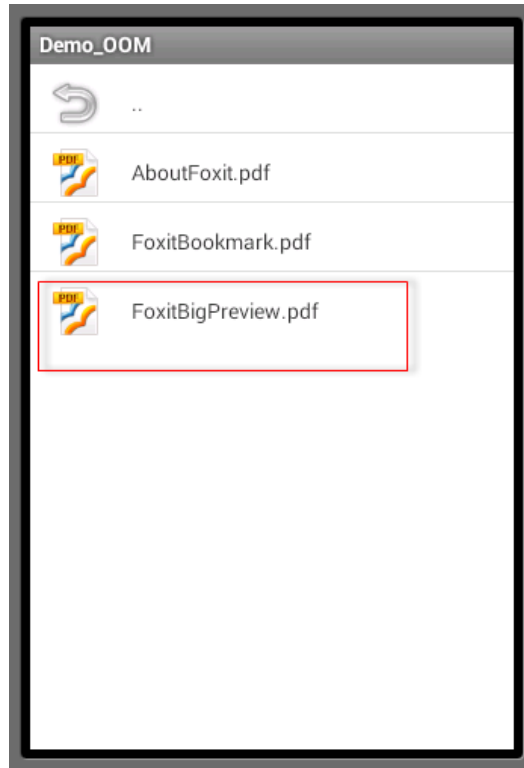


Figure 3-7

- e) The “FoxitBigPreview.pdf” is displayed in the screen as shown in Figure 3-8, which means the setting memory size is enough to load the PDF file.



Figure 3-9

- 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 3-10.
- h) Click the menu item, we can reset the memory size as shown in Figure 3-11.

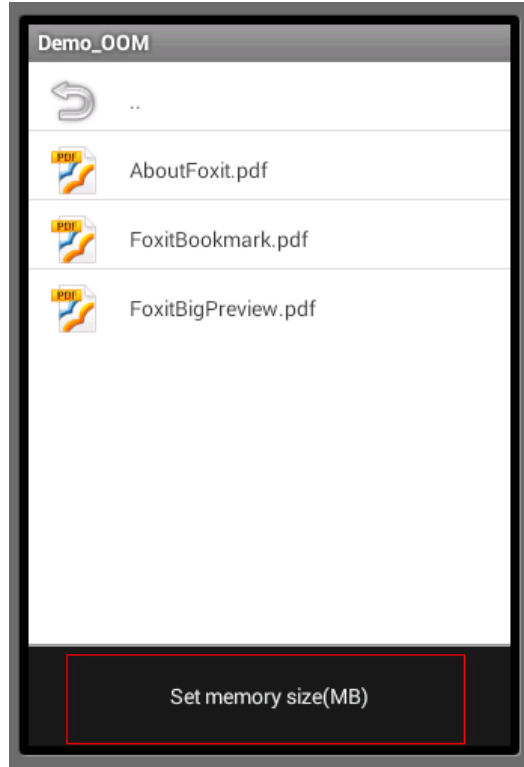


Figure 3-10



Figure 3-11

Viewer Demo

- a) To run this demo in Eclipse, you can refer to the OOM demo. Figure 3-12 shows the viewer demo running in an AVD targeting 4.4.2.

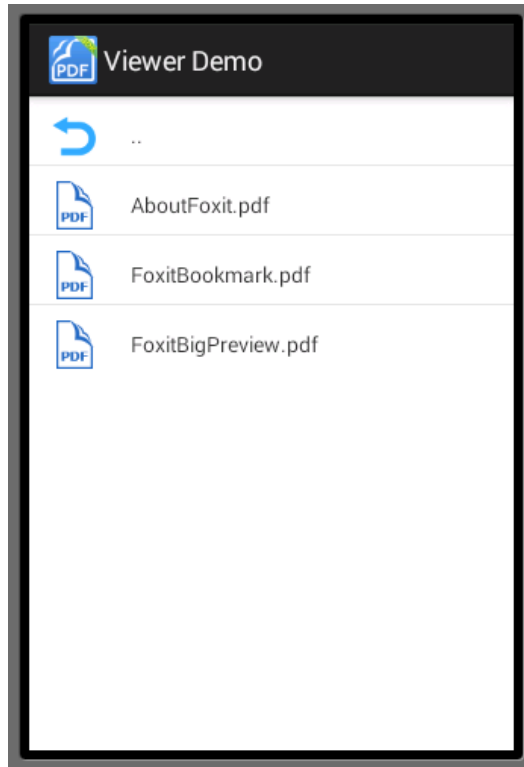


Figure 3-12

- b) Click the "AboutFoxit.pdf", the PDF file will be displayed as shown in Figure 3-13.

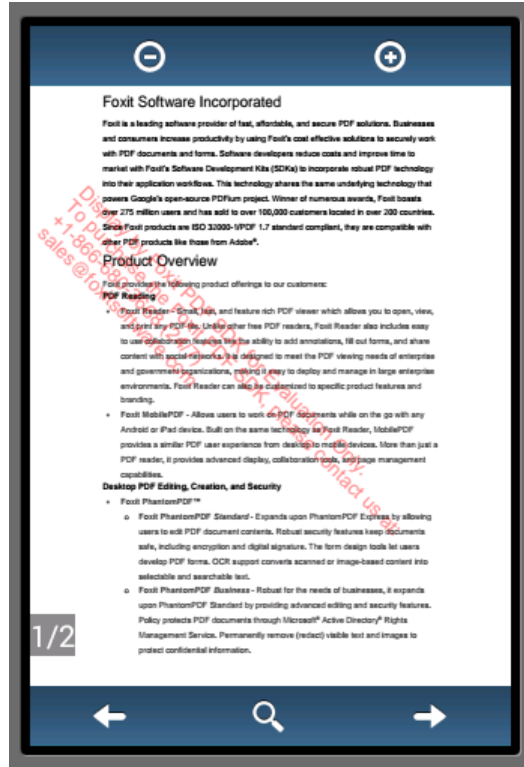


Figure 3-13

- c) The viewer demo provides functionalities like page turning, zooming and text search and extraction. For example, click the search button, type word “overview”, and press “Enter” key, the first search result will be highlighted as shown in Figure 3-14.

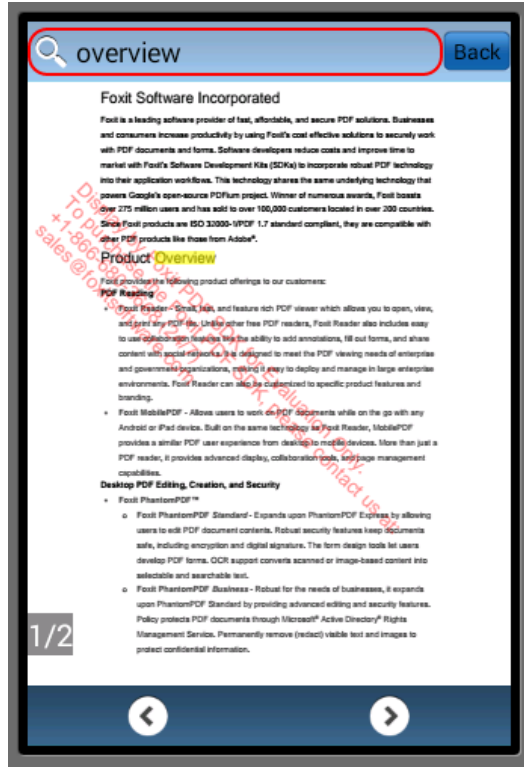


Figure 3-14

4 WORKING WITH SDK API

4.1 Common data structures and operations

In Foxit PDF SDK, resources of some objects (document, page etc.) are accessed using handles. Memory allocation and release need to be performed properly. Common data structures are listed in Table 4-1. For a complete list, please refer to the package “com.foxit.gsdk.pdf” or API reference ^[2].

Table 4-1

Java Class	Usage
PDFDocument	Handle to a pdf document
PDFPage	Handle to a page
Font	Handle to a font
PDFAction	Handle to base action
PDFReflowPage	Handle to a reflow page of a PDF page
PDFTextPage	Handle to a PDFTextPage represents an object which contains all PDF texts
PDFTextSearch	Handle to a PDFTextSearch represents a search object to search a PDF page text
RenderContext	Handle to a RenderContext represents the context of rendering
Renderer	Handle to render device

After the operation with the class, the handles no longer referenced need to be freed from resources. The APIs that are called for memory management are listed in Table 4-2.

Table 4-2

Java Class	Create/Initialize	Release/Close
PDFDocument	create() open()	close()
PDFPage	PDFDocument.getPage()	PDFDocument.closePage()
Font	FontManager.create() FontManager.createFromFile() FontManager.createStandard()	FontManager.release()
PDFAction	createGotoAction() createHideAction() createImportDataAction() createJavascriptAction() createLaunchAction() createNamedAction() createRemoteGotoAction()	release()

	createResetFormAction() createSubmitFormAction() createUriAction()	
PDFReflowPage	create(PDFPage)	release()
PDFTextPage	create(PDFPage)	release()
PDFTextSearch	PDFTextSearch(PDFTextPage)	release()
RenderContext	create()	release()
Renderer	create(Bitmap)	release()

4.2 Load Library

The PDFLibrary class offers methods to initialize and unlock the SDK. Foxit PDF SDK manages a license control mechanism to determine how to run for the application purpose. A license should be purchased for the application and pass unlock key and code to get proper supports. It can be constructed by the ways listed in Table 4-3. An example shows how to apply a license with hardcode method.

Table 4-3

Java Class	Description
getInstance()	Get the PDFLibrary object
getLicenseType()	Get the current license type
initialize()	Initialize Foxit PDF SDK library
unlock()	Unlock Foxit PDF SDK library using license key and code. This function should be called after Foxit PDF SDK library is initialized successfully
destroy()	Finalize PDF module
addFontFile()	Add an additional font

Example: 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();
}
```

4.3 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 actions 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. Some common APIs for file processing are listed in Table 4-4. For a complete list, please refer to “com.foxit.gsdk.utils.FileHandler.class” or API reference [2]. An example shows how to create a File Handler object.

Table 4-4

Java Class	Description
create(java.lang.string filename, int fileModes)	Create a FileHandler object from the specific file path
create(byte[] buffer, int fileModes)	Create a memory-based FileHandler object
getSize()	Get the actual size of a FileHandler object
Release()	Release a FileHandler object

Example: 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();
}
```

4.4 Document

PDF document is represented by PDFDocument handle object. Document level APIs provide functions to open and close files, get page, metadata and etc., which can be found in “com.foxit.gsdk.pdf.PDFDocument.class”. An PDFDocument handle should be initialized by calling open() to allow page or deeper level API to work. Some common APIs at document level are listed in Table 4-5. For a complete list, please refer to “com.foxit.gsdk.pdf.PDFDocument.class” or API reference [2]. An example shows how to get PDF page and save it to a file.

Table 4-5

Java Class	Description
open()	Open an existing PDF document.
close()	Close a PDF document
getEncryptionType()	Get encryption type
getAction()	Get PDF document trigger action
getMetadata()	Get PDF metadata corresponding to the document
createBookmarkIterator()	Refer to Bookmark section
getUIVisibility()	Get UI visibility status from viewer preferences

loadAttachments()	Get a specific attachment from PDF document
create()	Create a new PDF document object
createPage()	Create a new page
startSaveToFile()	Start saving a PDF document to a file in a progressive manner
setAction()	Set document trigger action

Example1: get PDF page

```
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();
}
```

Example2: 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();
}
```

4.5 Attachment

In Foxit PDF SDK, attachments are only referred to attachments of documents rather than file attachment annotation. PDF SDK provides applications APIs to access attachments such as loading attachments, getting attachments, inserting attachments and accessing properties of attachments. Some common APIs are listed in Table 4-6. For a complete list, please refer to

“com.foxit.gsdk.pdf.PDFAttachment.class”, “com.foxit.gsdk.pdf.PDFAttachments.class” or API reference [2]. An example shows how to insert an attachment file into a PDF.

Table 4-6

API Name	Description
PDFDocument.loadAttachments()	Load all attachments of PDF document
PDFAttachments.release()	Release a attachments object
PDFAttachments.countAttachment()	Get the count of attachments
PDFAttachments.getAttachment()	Get a specific attachment
PDFAttachments.insertAttachment()	Insert an attachment
PDFAttachment.getFileName()	Get file name of an attachment
PDFAttachment.setFile()	Set the file of an attachment

Example: 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();
}
```

4.6 Page

PDF page is represented by PDFPage handle object. Page level APIs provide functions to parse, render, read and set the properties of a page. PDFPage object is created by a PDFDocument object using **getPage(int)** or **createPage(int)**. A PDF page needs to be parsed before it is rendered or processed for text extraction. Some common APIs at page level are listed in Table 4-7. For a complete list, please refer to “com.foxit.gsdk.pdf.PDFPage.class” or API reference [2]. Two examples show how to work with PDF page.

Table 4-7

Java Class	Description
startParse()	Start parsing a PDF page
getIndex()	Get page index

getSize()	Get page size
getDisplayMatrix()	Get page transformation matrix
startRender()	Start rendering a PDF page in a renderer with a PDF rendering context
setIndex()	Change page index of a PDF page
setAction()	Set a page trigger action
startRenderAnnots	Render annotations on render context
startRenderPageAnnots	Render all annotations of a page on render context
startRenderFormControls	Render a PDF form control
startRenserPageFormControls	Render all form controls of a page on render context

Example1: create 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();
```

Example2: delete 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();
}
```

4.7 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 **create()**. The rendering settings (or render context) are set in RenderContext object. Some common APIs for rendering are listed in Table 4-8. For a complete list, please refer to

“com.foxit.gsdk.pdf.RenderContext.class”, “com.foxit.gsdk.pdf.Renderer.class”, “com.foxit.gsdk.pdf.RenderColorOption.class” or API reference [2]. Two examples show how to use rendering APIs in PDF SDK.

Table 4-8

Java Class	Description
create()	Create a renderer on a bitmap object
release()	Release a given renderer object
setFlags()	Set flags of a renderer
drawBitmap()	Render a bitmap object
RenderConetxt.create()	Create a PDF rendering context
RenderConetxt.release()	Release a PDF rendering context
RenderConetxt.setMatrix ()	Set a transformation matrix to a rendering context

Example1: parse 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();
}
```

Example2: render 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();
}

```

4.8 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() to get the textPage object. Some common APIs for text processing are listed in Table 4-9. For a complete list, please refer to “com.foxit.gsdk.pdf.PDFTextPage.class” or API reference [2]. Two examples show how to use text APIs in PDF SDK.

Table 4-9

Java Class	Description
create()	Prepare information about all characters in a page
release()	Release all resources allocated for a PDF text page handle
getChars()	Get text content in a page, within a specific character range
exportToFile()	Export text content in a page to a specific file handle
selectByRange()	Get a text selection handle by specific character range
getChars()	Extract the whole text from a PDF text selected area
PDFTextSearch.startSearch()	Start a PDF text search process
PDFTextSearch.findNext()	Search in the direction from page start to end
PDFTextSearch.getSelection()	Get a PDFTextSelection from a text search when a match is found

Example1: text selection

```

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();
}

```

Example2: text search

```

public PDFTextSearch search = null;
try {
    //whole word is compared with no case sensitive
    search.startSearch("foxit", PDFTextSearch.SEARCHFLAG_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();
}

```

4.9 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. Some common APIs for text link processing are listed in Table 4-10. For a complete list, please refer to “com.foxit.gsdk.pdf.PDFTextLink.class” or API reference [2]. An example shows how to get the first URL formatted texts in a page.

Table 4-10

Java Class	Description
getLink()	Get the linked URL associated with a specific hyperlink
countLinks()	Get count of URL formatted texts inside a page
getSelection()	Get a PDFTextSelection from a specific hyperlink
release()	Release all resources allocated for a PDFTextLink
PDFTextPage: extractLinks()	Process a PDF page text object for getting URL formatted texts

Example: get the first URL formatted texts in a 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();
}

```

4.10 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()** can export data in a PDF document to an FDF (Forms Data Format) document, from where data can be extracted for further use. Some common APIs for form processing are listed in Table 4-11. For a complete list, please refer to the classes in package “com.foxit.gsdk.pdf.form” or API reference [2]. An example shows how to count form fields and get the properties.

Table 4-11

API Name	Description
PDFDocument.loadForm()	Retrieve a form handle for specific document
PDFDocument.releaseForm()	Release the resources of a form handle
PDFForm.getField()	Search and retrieve the name and type of a field satisfying a name filter in a form
PDFFormField.getAction()	Retrieve action associated with a field and a trigger type at a specified index in a form
PDFForm.exportToFDF()	Export data in a form to a FDF document
PDFForm.setDefaultAppearance()	Set default appearance of a form
PDFFormFiller.setHighlightColor()	Set the highlight color for the form field
PDFFormField.insertAction()	Insert an action associated with a field and a trigger type at a specified index in a form
PDFForm.beginFormFiller	Begin the form filling.
PDFFormFiller.setHighlightColor	Set the highlight color for the form field
PDFFormFiller.ShowHighlight	Whether to show the highlight of form field or not.
PDFForm.endFormFiller	Finish the form filling.

Example: 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();
}
```

4.11 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 4-12. 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 4-12 shows whether an annotation is a markup annotation.

Foxit PDF SDK supports most annotation types defined in PDF ISO standard^d. PDF SDK provides APIs of annotation creation, properties access and modification, appearance setting and drawing. Some common APIs are listed in Table 4-13. For a complete list, please refer to the classes in package “com.foxit.gsdk.pdf.annots” or API reference^[2].

Table 4-12

Annotation type	Description	Markup	Supported by SDK
Text(Note)	Text annotation	Yes	Yes
Link	Link Annotations	No	Yes
FreeText(TypeWriter)	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 ISO standard [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.

Table 4-13

API Name	Description
PDFPage.loadAnnots	Load annotations from a PDF page
PDFPage.countAnnots	Get count of annotations by a specific filter
PDFPage.getAnnot	Get a specified annotation by a specific filter
Annot.getIndex	Get index of an annotation filtered by a string
Annot.getName	Get an annotation's name.
PDFPage.addAnnot	Add an annotation by a given index and a string filter
PDFPage.removeAnnot	Remove an annotation from a given page
Annot.setFlags	Set flags of an annotation
Annot.setName	Set name of an annotation

Example: 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();
}
```

4.12 Image Conversion

Foxit PDF SDK provides APIs for conversion between PDF files and images. Applications could easily fulfill functionality like image creation, conversion, input and output operations. Some common APIs are listed in Table 4-14. For a complete list, please refer to the classes in package “com.foxit.gsdk.image” or API reference^[2]. An example shows how to convert PDF pages to bitmap files.

Table 4-14

API Name	Description
Image.load	Load image from image file.
Image.countFrames	Count frames of an image.
Image.loadFrame	Load image frame by index.
Image.getCurrentFrameBitmap	Retrieve bitmap of current frame.
ImageFile.create	Create an image file.
ImageFile.addFrame	Add a frame to image file.

Example: 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 = pPage.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 = pPage.startRender(renderContext, render, 0);
    if(renderContext !=null){
        int ret = Progress.TOBECONTINUED;
        while(ret == Progress.TOBECONTINUED ){
            ret = renderProgress.continueProgress(30);
        }
    }
    renderProgress.release();
}
...
}
```

4.13 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. PDFBookmarkIterator object is created by calling PDFDocument.createBookmarkIterator(), and getBookmarkData() can be used to get the data of the current bookmark item. Some common APIs for bookmark processing are listed in Table 4-15. For a complete list, please refer to “com.foxit.gsdk.pdf.PDFBookmarkIterator.class” or API reference^[2]. An example shows how to create a bookmark tree and show all bookmarks.

Table 4-15

Java Class	Description
isFirstChild()	Determine whether a bookmark iterator is the first child of its parent
isLastChild()	Determine whether a bookmark iterator is the last child of its parent
insert()	Insert a new bookmark at a position and set the data
insertAction()	Insert a bookmark action
getActions()	Get the specified bookmark action
getpos()	Get the bookmark position handle from PDFBookmarkIterator
moveToParent()	Move a bookmark iterator to its parent if it exists
moveToFirstChild()	Move a bookmark iterator to its first child if it exists
clonebookmark()	Clone an iterator to access bookmark in a document
getBookmarkData	Get current bookmark item's data
PDFDocument.createBookmarkIterator()	Create a new PDFBookmarkIterator in the current document
release()	Release a bookmark iterator

Example: 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.getContext(), threadDisplayString, 3).show();
    }
});

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

```

4.14 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 . Some common APIs are listed in Table 4-16. For a complete list, please refer to “com.foxit.gsdk.pdf.PDFReflowPage.class” or API reference ^[2]. An example shows how to create a reflow page.

Table 4-16

Java Class	Description
create()	Create a reflow page from a given PDF page
release()	Release resources allocated for a reflow page before
setSize()	Set screen size. It shall be called before startParse()
setLineSpace()	Set line space. It shall be called before startParse()
startParse()	Start parsing progress for a reflow page.
getContentSize()	Get width and height of a reflow page. It takes effective after startParse()
getMatrix()	Get the transformation matrix of a given reflow page.
startRender()	Start a rendering progress to render a reflow page.
getFocusData()	Get focus data by a given position in device coordinate system.
getFocusPos()	Get a position of focus data by given focus data in device coordinate system.

Example: 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();

```

4.15 Pressure Sensitive Ink

Pressure Sensitive Ink (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. Some common API functions are listed in Table 4-17. For a complete list, please refer to “com.foxit.gsdk.psi.PSI.class” or API reference [2]. An example shows how to create a PSI and set the related properties for it.

Table 4-17

Java Class	Description
create()	Create a pressure sensitive ink object
release()	Destroy a pressure sensitive ink object
initCanvas()	Initialize canvas for pressure sensitive ink
setInkColor()	Set color of ink for a pressure sensitive ink object
setInkDiameter()	Set diameter of ink for a pressure sensitive ink object

setOpacity()	Set opacity of ink for a pressure sensitive ink object.
getContentsRect()	Get content rectangle of a pressure sensitive ink object
addPoint()	Add a point to a pressure sensitive ink object
render()	Render a pressure sensitive ink object
convertToPDFAnnot()	Convert A pressure sensitive ink object to a PDF annotation

Example: 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();
}
```

4.16 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. Some common APIs are listed in Table 4-18. For a complete list, please refer to the classes in package “com.foxit.gsdk.pdf.action” or API reference^[2]. An example shows how to operate link action.

Table 4-18

API name	Description
PDFDocument.getAction	Get document trigger action
PDFBookmarkIterator.getAction	Get the specified bookmark action
PDFPage.getAction	Get a trigger action of a page
Link.getAction	Get action data of specific index associated with an annotation
PDFFormField.getAction	Retrieve action associated with a field and a trigger type at a specified index in a form

Example: 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.getLastErrorMessage() == PDFException.ERRCODE_NOTFOUND){
        Toast.makeText(MainActivity.this, "It is not a annotation!", Toast.LENGTH_LONG).show();
    }
}
```


4.17 Page Object

Page object is a feature that allows novice users having limited knowledge of PDF objects to be able to work with text, path, image, and canvas objects. Foxit PDF SDK provides APIs to add and delete PDF objects in a page and set specific attributes. Using page object, users can create PDF page from object contents. Other possible usages of page object include adding headers and footer to PDF documents, adding an image logo to each page, and generating a template PDF on demand. Some common APIs are listed in Table 4-19. For a complete list, please refer to the classes in package “com.foxit.gsdk.pdf.pageobjects” or API reference [2]. An example shows how to create an image object in a page.

Table 4-19

API name	Description
PDFPage.getPageObjects	Get page objects in a page
PageObjects.insertObject	Insert a page object and it will be automatically freed
PageObjects.countObjects	Get the count of the page objects
PageObjects.generateContents	Generatate PDF Page contents
ImageObject.create	Create an image object

Example: create an image object in a page

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

5 FAQ

1. What's 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 after purchasing Foxit PDF SDK for Android?

There are detailed descriptions on how to apply a license in the section 3.2. You can refer to the descriptions to activate a license.

3. How can I look for technical support when I try Foxit PDF SDK for Android?

You can send email to support@foxitsoftware.com for any questions or comments or call our support at 1-866-693-6948.

REFERENCES

[1] PDF reference 1.7

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[2] Foxit PDF SDK API reference

sdk_folder/docs/ Foxit PDF SDK 4.1 API Reference.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