Foxit PDF SDK

DEVELOPER GUIDE
FOXIT PDF SDK

For Android

©Foxit Software Incorporated. All Rights Reserved.

TABLE OF CONTENTS

1 Introduction to Foxit PDF SDK 1

1.1 Foxit PDF SDK 1

1.2 Foxit PDF SDK for Android 1

1.2.1 Why Foxit PDF SDK for Android is your choice 1

1.2.2 Main Frame of Foxit PDF SDK for Android 2

1.2.3 UI Extensions Component Overview 4

1.2.4 Key Features of Foxit PDF SDK for Android 6

1.3 Evaluation 8

1.4 License 8

1.5 About this Guide 8

2 Getting Started 10

2.1 System Requirements 10

2.2 What is in the Package 10

2.3 How to run a demo 12

2.3.1 Function demo 13

2.3.2 Viewer control demo 15

2.3.3 Complete PDF viewer demo 18

3 Rapidly building a full-featured PDF Reader 22

3.1 Create a new Android project 22

3.2 Integrate Foxit PDF SDK for Android into your apps 23

3.3 Initialize Foxit PDF SDK for Android 30

3.4 Display a PDF document using PDFViewCtrl 31

3.5 Open a RMS protected document 37

3.6 Build a full-featured PDF Reader with UI Extensions Component 41

3.7 Add the scanning feature based on the full-featured PDF Reader 49
3.8 Handle Scoped Storage 55
4 Customizing User Interface 57
4.1 Customize the UI through a configuration file 57
4.1.1 Introduction to JSON file 57
4.1.2 Configuration Items Description 63
4.1.3 Instantiate a UIExtensionsManager object with the configuration file 71
4.1.4 Examples for customizing UI through a configuration file 72
4.2 Customize UI elements through APIs 74
4.2.1 Customize top/bottom toolbar 74
4.2.2 Customize to add or remove a specific panel 86
4.2.3 Customize to hide the UI elements in the View setting bar 90
4.2.4 Customize to add or hide the UI elements in the More Menu view 93
4.3 Customize UI implementation through source code 98
5 Working with SDK API 104
5.1 Render 104
5.1.1 How to render a specified page to a bitmap 105
5.2 Text Page 106
5.2.1 How to get the text area on a page by selection 106
5.3 Text Search 107
5.3.1 How to search a text pattern in a PDF 108
5.4 Bookmark (Outline) 109
5.4.1 How to travel the bookmarks of a PDF in depth first order 109
5.5 Reading Bookmark 111
5.5.1 How to add a custom reading bookmark and enumerate all the reading bookmarks 111
5.6 Attachment 112

5.6.1 How to embed a specified file to a PDF document 112
5.6.2 How to export the embedded attachment file from a PDF and save it as a single file 113
5.7 Annotation 114
5.7.1 How to add annotations to a PDF page 115
5.7.2 How to delete annotations in a PDF page 117
5.7.3 How to register listeners to receive annotation events 118
5.8 Form 119
5.8.1 How to import and export form data from or to a XML file 119
5.9 Security 120
5.9.1 How to encrypt a PDF file with password 120
5.10 Signature 121
5.10.1 How to sign a PDF document and verify the signature 121
5.10.2 How to set customized time information for signature 123
6 Creating a custom tool 127
7 Implement Foxit PDF SDK for Android using Cordova 140
8 Implement Foxit PDF SDK for Android using React Native 141
9 Implement Foxit PDF SDK for Android using Xamarin 142
10 FAQ 143
10.1 Open a PDF document from a specified PDF file path 143
10.2 Display a specified page when opening a PDF document 144
10.3 License key and serial number cannot work 145
10.4 Add a link annotation to a PDF file 145
10.5 Insert an image into a PDF file 146
10.6 SetDocModified API 148

10.7 Highlight the links in PDF documents and set the highlight color 148
10.8 Highlight the form fields in PDF form files and set the highlight color 149
10.9 Indexed Full Text Search support 149
10.10 Print PDF document 152
10.11 Night mode color settings 153
10.12 Output exception/crash log information 154
10.13 Reduce size of APK 155
10.14 Enable shrink-code (set "minifyEnabled" to "true") 155
10.15 Localization settings 156
10.16 Migrate to AndroidX 157
10.17 Support Chromebook 159
10.18 Use UIExtensions for read-only mode 160
10.19 Compatible with Android Studio 3.2 160
10.20 Update page binding to support Right-to-Left 161
10.21 Issue with opening web PDFs 164
10.22 Improve efficiency in inserting and rendering watermarks 164

11 Technical Support 174

1 Introduction to Foxit PDF SDK

1.1 Foxit PDF SDK

Foxit PDF SDK provides high-performance libraries to help any software developer add robust PDF functionality to their enterprise, mobile and cloud applications across all platforms (includes Windows, Mac, Linux, Web, Android, iOS, and UWP), using the most popular development languages and environments.

Application developers who use Foxit PDF SDK can leverage Foxit's powerful, standard-compliant PDF technology to securely display, create, edit, annotate, format, organize, print, share, secure, search documents as well as to fill PDF forms. Additionally, Foxit PDF SDK includes a built-in, embeddable PDF Viewer, making the development process easier and faster. For more detailed information, please visit the website https://developers.foxitsoftware.com/pdf-sdk/.

In this guide, we focus on the introduction of Foxit PDF SDK for Android platform.

1.2 Foxit PDF SDK for Android

Have you ever worried about the complexity of the PDF specification? Or have you ever felt lost when asked to build a full-featured PDF app within a limited time-frame? If your answer is "Yes", then congratulations! You have just found the best solution in the industry for rapidly integrating PDF functionality into your apps.

Foxit PDF SDK for Android focuses on helping developers easily integrate powerful Foxit PDF technology into their own mobile apps. With this SDK, even developers with a limited knowledge of PDF can quickly build a professional PDF viewer with just a few lines of code on Android platform.

1.2.1 Why Foxit PDF SDK for Android is your choice

Foxit is a leading software provider of solutions for reading, editing, creating, organizing, and securing PDF documents. Foxit PDF SDK libraries have been used in many of today's leading apps, and they are proven, robust, and battle-tested to provide the quality, performance, and features that the industry's largest apps demand.

Foxit PDF SDK for Android provides quick PDF viewing and manipulation support for Android Devices. Customers choose it for the following reasons:

- **Easy to integrate**

Developers can seamlessly integrate the SDK into their own apps with just a few lines of code.

- **Perfectly designed**

Foxit PDF SDK for Android is designed with a simple, clean, and friendly style, which provide the best user experience.

- **Flexible customization**

Foxit PDF SDK for Android provides the source code for the user interface which lets the developers have full control of the functionality and appearance of their apps.

- **Robust performance on mobile platforms**

Foxit PDF SDK for Android provides an OOM (out-of-memory) recovery mechanism to ensure the app has high robust performance when running the app on a mobile device which offers limited memory.

- **Powered by Foxit's high fidelity rendering PDF engine**

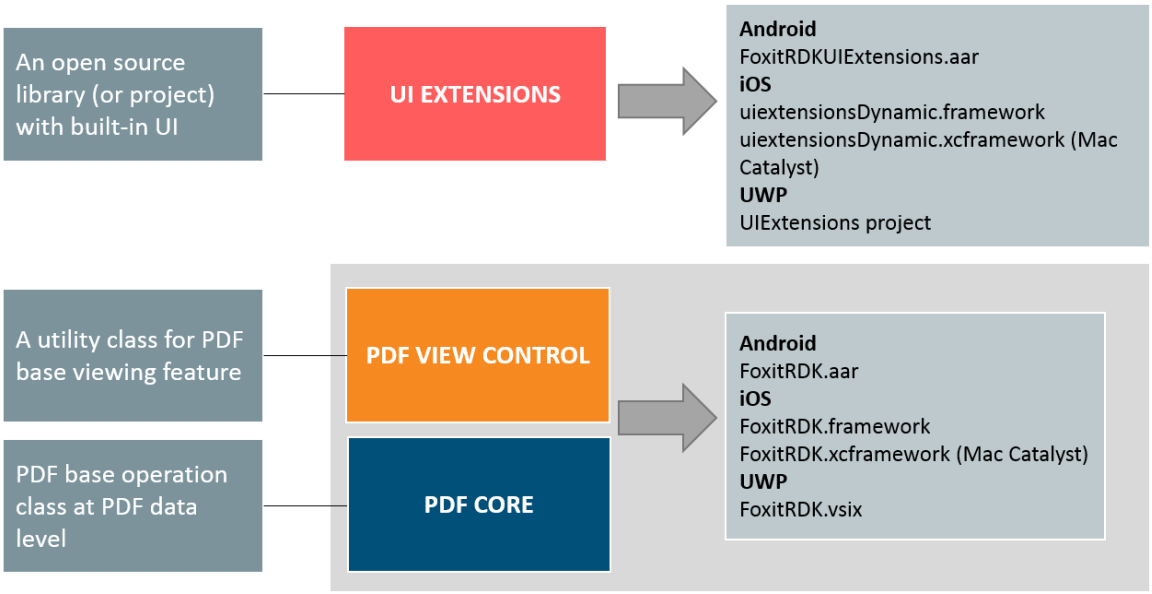
The core technology of the SDK is based on Foxit's PDF engine, which is trusted by a large number of the world's largest and well-known companies. Foxit's powerful engine makes the app fast on parsing, rendering, and makes document viewing consistent on a variety of devices.

- **Premium World-side Support**

Foxit offers premium support for its developer products because when you are developing mission critical products you need the best support. Foxit has one of the PDF industry's largest team of support engineers. Updates are released on a regular basis to improve user experience by adding new features and enhancements.

1.2.2 Main Frame of Foxit PDF SDK for Android

Foxit PDF SDK for Android consists of three elements as shown in the following picture. This structure is shared between all mobile platform versions of Foxit PDF SDK, which makes it easier to integrate and support multiple mobile operating systems and frameworks in your apps.



The three elements of Foxit PDF SDK for Android, iOS and UWP

- **PDF Core API**

The PDF Core API is the heart of this SDK and is built on Foxit's powerful underlying technology. It provides the functionality for basic PDF operation features, and is utilized by the PDF View Control and UI Extensions Component, which ensures the apps can achieve high performance and efficiency. The Core API can be used independently for document rendering, analysis, text extraction, text search, form filling, digital signatures, Pressure Sensitive Ink, certificate and password security, annotation creation and manipulation and much more.

- **PDF View Control**

The PDF View Control is a utility class that provides the functionality for developers to interact with rendering PDF documents per their requirements. With Foxit's renowned and widely used PDF rendering technology at its core, the View Control provides fast and high quality rendering, zooming, scrolling and page navigation features. The View Control derives from platform related viewer classes such as Android.View.ViewGroup and allows for extension to accommodate specific user needs.

- **UI Extensions Component**

The UI Extensions **Component** is an open source library that provides a customizable user interface with built-in support for text selection, markup annotation, outline navigation, reading bookmarks, full-text searching, form filling, text reflow, attachment, digital/handwritten signature, document

editing and password encryption. These features in the UI Extensions Component are implemented using the PDF Core API and PDF View Control. Developers can utilize these ready-to-use UI implementations to build a PDF viewer quickly with the added benefit of complete flexibility and control to customize the UI design as desired.

From version 4.0, Foxit PDF SDK for Android made a big change and optimization for the UI Extensions Component. Now, it wraps the basic UI implementations to PDFReader class, such as panel controller, toolbar settings, and alert view, etc. Building a full-featured PDF Reader is getting simpler and easier. Furthermore, users can flexibly customize the features they want through a configuration file.

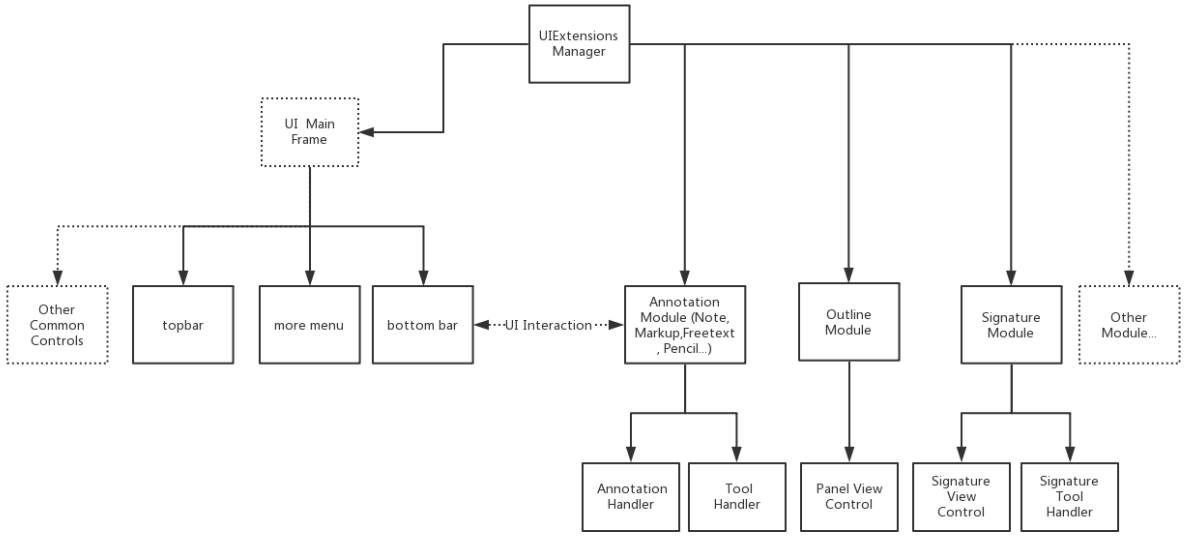
From version 5.0, every element in the built-in UI can be configurable through an API. More advanced APIs and more powerful configuration file are provided for developers to further customize the UI elements, such as adding/removing a button to/from the toolbar, showing/hiding a specific menu/function panel, and etc.

From version 6.0, Foxit PDF SDK for Android removed the PDFReader class and moved the wrapped APIs in PDFReader class to UI Extensions Component.

1.2.3 UI Extensions Component Overview

The UI Extensions Component uses "module" mechanism which refines each feature into a module. All of the modules except LocalModule (used for file management) will be loaded by default if UI Extensions is added. Users can customize module through implementing Module interface class, and then call **UIExtensionsManager#registerModule** to register the custom module to current UIExtensions manager. When not in use, you can call **UIExtensionsManager#unregisterModule** to unregister it from current UIExtensions manager.

UIExtensionsManager contains the main-frame UI, such as top/bottom toolbar, and other UI components which are shared between each module. Meanwhile, through UIExtensionsManager, each feature module can also be loaded separately. And when loaded, the feature module will adapt and adjust the main-frame UI, as well as establish the connection of message event response. Each feature module may contain its module-specific UI components, and have its self-contained message event handling logic. UIExtensionsManager will also be responsible for distributing messages and events received from View Control component to each feature module. The following figure shows the detailed relationship between UIExtensionsManager and modules.



The relationship between UIExtensionsManager and modules

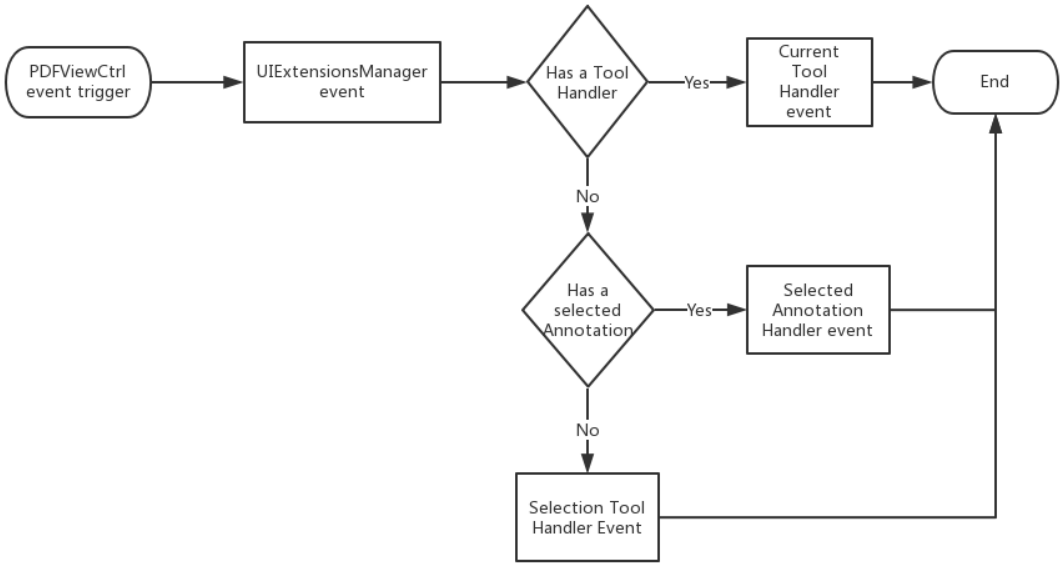
Tool handler and annotation handler will process the events from touch screen or gestures of PDFViewCtrl. When the touch screen and gestures occur, PDFViewCtrl will send the corresponding events to UIExtensionsManager:

a) If a tool handler exists currently, UIExtensionsManager will send the corresponding events to the current tool handler, and then event-handing process ends.

b) If an annotation is selected currently, UIExtensionsManager will send the corresponding events to the annotation handler corresponding to the currently selected annotation, and then event-handing process ends.

c) If currently no tool handler exists and no annotation is selected, UIExtensionsManager will send the corresponding events to selection tool handler. Text Selection tool is used for processing the related events for text selection. For example, select a piece of text, and add Highlight annotation. Blank Selection tool is used for processing the related events for blank space. For example, add a Note annotation on the blank space.

Note: Tool Handler and Annotation Handler will not respond the events at the same time. Tool Handler is primarily used for annotation creation (currently, the creation of link annotation is not supported), signature creation and text selection. Annotation Handler is mainly used for annotation editing and form filling. The following figure shows the event response flow chart between Tool Handler and Annotation Handler.



The event response flow chart between Tool Handler and Annotation Handler

1.2.4 Key Features of Foxit PDF SDK for Android

Foxit PDF SDK for Android has several main features which help app developers quickly implement the functions that they really need and reduce the development cost.

Features	Description
PDF Document	Open and close files, set and get metadata.
PDF Page	Parse, render, read, and edit PDF pages.
Render	Graphics engine created on a bitmap for platform graphics device.
Reflow	Rearrange page content.
Crop	Crop PDF pages for betting reading.
Text Select	Select text in a PDF document.
Text Search	Search text in a PDF document, and provide indexed Full-Text Search.
Outline	Directly locate and link to point of interest within a document.
Reading Bookmark	Mark progress and interesting passages as users read.

Annotation	Create, edit and remove annotations.
Layers	Add, edit, and remove optional content groups.
Attachments	Add, edit, and remove document level attachments.
Form	Fill form with JavaScript support, export and import form data by XFDF/FDF/XML file. Support to create TextField, CheckBox, RadioButton, ComboBox, ListBox, and Signature Field.
XFA	Support static and dynamic XFA.
Signature	Sign a PDF document, verify a signature, add or delete a signature field. Add and verify third-party digital signature. Support Long term validation of signatures (LTV).
Fill	Fill flat forms (i.e. non-interactive forms) with text and symbols.
Security	Protect PDFs with password or certificate.
Pan and Zoom	Adjust the magnification and position of the view area to match the area in an adjustable rectangle in the Pan & Zoom window's thumbnail view of the page.
Print	Print PDF document.
RMS	Support Microsoft RMS decryption with the standard IRMv1 and IRMv2.
Comparison	Compare two PDF documents, and mark the differences between them.
Scanning	Scan and convert paper documents to PDFs.
Speak	Support to read out the text of PDF file.
Split Screen	Support split screen.
Right-to-Left	Support Right-to-Left.
Out of Memory	Recover from an OOM condition.

***Note**: Outline is the technical term used in the PDF specification for what is commonly known as bookmarks in traditional desktop PDF viewers. Reading bookmarks are commonly used on mobile and tablet PDF viewers to mark progress and interesting passages as users read but are not technically outline and are stored at app level rather than within the PDF itself.*

Support robust PDF applications with Foxit PDF SDK for Android

Development of robust PDF applications is challenging on mobile platforms which has limited memory. When memory allocation fails, applications may crash or exit unexpectedly. To deal with this issue, Foxit PDF SDK for Android provides an out-of-memory (**OOM**) mechanism to support applications.

OOM is an evolved feature in Foxit PDF SDK for Android because of its complexity. The key of OOM mechanism is that Foxit PDF SDK for Android will monitor the usage of memory and take recovery operations automatically once OOM is detected. During the recovery process, Foxit PDF SDK for Android reloads the document and page automatically and restores the status to the original before OOM. It means the current reading page and location, as well as page view mode (single or continuous page) can be recovered. However, the data generated from editing will be lost.

Foxit PDF SDK for Android provides a property "**shouldRecover**" in PDFViewCtrl class. By default, the value of "**shouldRecover**" is "**true**". If you do not want to enable the auto-recovery when OOM is detected, you can set "**shouldRecover**" to "**false**" as follows:

```
PDFViewCtrl pdfViewerCtrl = new PDFViewCtrl(getActivity().getApplicationContext());
pdfViewerCtrl.shouldRecover = false;
```

At that time, the application will throw an exception, and may crash or exit unexpectedly.

1.3 Evaluation

Foxit PDF SDK allows users to download trial version to evaluate SDK. The trial version has no difference from the standard licensed version except for the free 10-day trial limitation and the trial watermarks in the generated pages. After the evaluation period expires, customers should contact the Foxit sales team and purchase licenses to continue using Foxit PDF SDK.

1.4 License

Developers should purchase licenses to use Foxit PDF SDK in their solutions. Licenses grant developers permission to release their apps which utilize Foxit PDF SDK. However, users are prohibited to distribute any documents, sample code, or source code in the released package of Foxit PDF SDK to any third party without written permission from Foxit Software Incorporated.

1.5 About this Guide

This guide is intended for the developers who need to integrate Foxit PDF SDK for Android into their own apps. It aims at introducing the following sections:

- Section 1: gives an introduction of Foxit PDF SDK, especially for Android platform SDK.

- Section 2: illustrates the package structure and running demos.
- Section 3: describes how to quickly create a full-featured PDF Reader.
- Section 4: introduces how to customize the user interface.
- Section 5: shows how to use Foxit PDF SDK Core API.
- Section 6: shows how to create a custom tool.
- Section 7: shows how to implement Foxit PDF SDK using Cordova.
- Section 8: shows how to implement Foxit PDF SDK using React Native.
- Section 9: shows how to implement Foxit PDF SDK using Xamarin.
- Section 10: lists some frequently asked questions.
- Section 11: provides support information.

2 Getting Started

It is very easy to setup Foxit PDF SDK for Android and see it in action! It takes just a few minutes and we will show you how to use it on the Android platform. The following sections introduce the structure of the installation package and how to run a demo.

2.1 System Requirements

Android devices' requirements:

- Android 4.4 (API 19) or higher
- 32/64-bit ARM (armeabi-v7a/arm64-v8a) or 32/64-bit Intel x86 CPU

Android Studio 3.2 or newer (support AndroidX)

The runtime environment for our demos:

- Android Studio 4.1
- JDK 1.8
- Gradle Version 4.6
- Gradle Build Tool 3.2

***Note**: Starting with version 7.2, Foxit PDF SDK for Android will only support AndroidX, and no longer support Android support library.*

2.2 What is in the Package

Download the "foxitpdfsdk_9_0_android.zip" package, and extract it to a new directory like "foxitpdfsdk_9_0_android" as shown in Figure 2-1. The package contains:

docs:	A folder containing API references, developer guide, and upgrade warnings.
icc_profile	The default icc profile files used for output preview feature
libs:	A folder containing license files, AAR files, and UI Extensions Component source code.
samples:	A folder containing Android sample projects.

getting_started_android.pdf: A quick guide for Foxit PDF SDK for Android.
legal.txt: Legal and copyright information.
release_notes.txt: Release information.

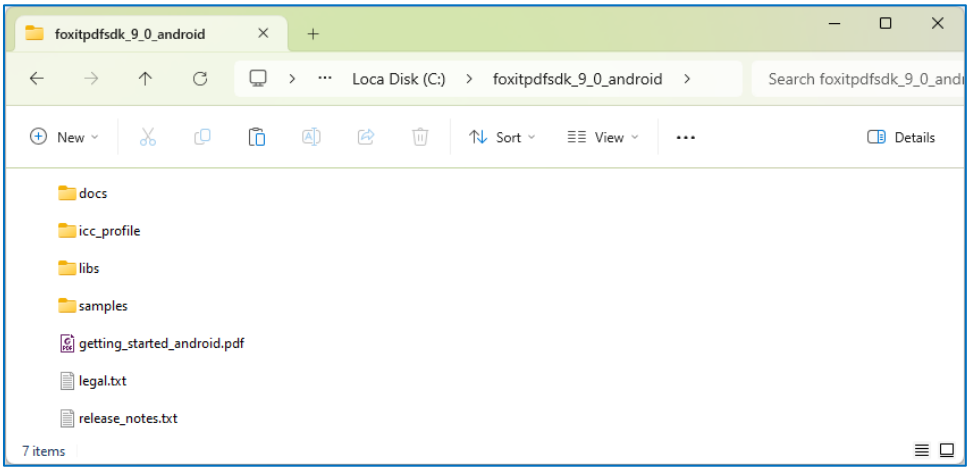


Figure 2-1

In the "libs" folder as shown in Figure 2-2, there are items that make up the core components of Foxit PDF SDK for Android, and third-party libraries used for Microsoft RMS support.

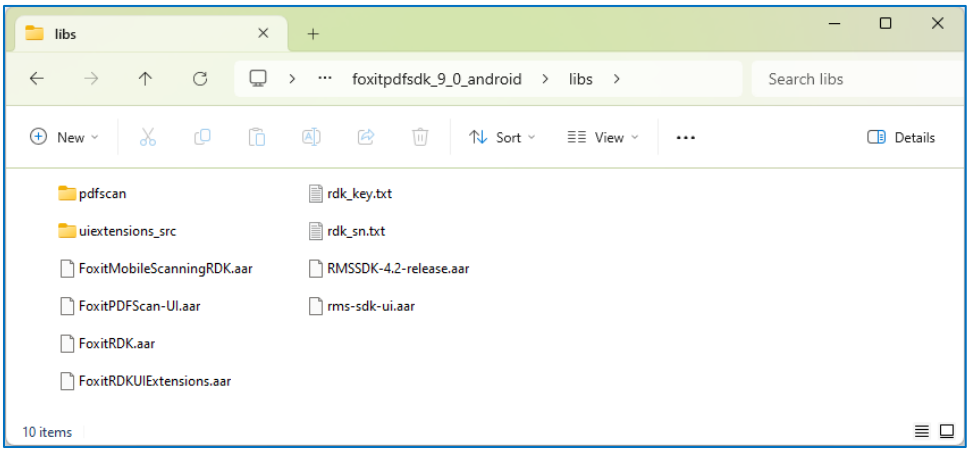


Figure 2-2

- ***uiextensions_src*** project - found in the "libs" folder. It is an open source library that contains some ready-to-use UI module implementations, which can help developers rapidly embed a fully functional PDF reader into their Android app. Of course, developers are not forced to use the default UI, they can freely customize and design the UI for their specific apps through the "uiextensions_src" project.

- ***FoxitRDK.aar*** - contains JAR package which includes all the Java APIs of Foxit PDF SDK for Android, as well as the underlying ".so" libraries. The ".so" library is the heart of the SDK including the core functionalities of Foxit PDF SDK for Android. It is built separately for each architecture, and currently available for armeabi-v7a, arm64-v8a, x86, and x86_64.
- ***FoxitRDKUIExtensions.aar*** - generated by the "**uiextensions_src**" project found in the "libs" folder. It includes the JAR package, built-in UI implementation, and resource files that are needed for the built-in UI implementations, such as images, strings, color values, layout files, and other Android UI resources.
- ***pdfscan*** project - It is an open source library that contains the UI implementations for scanning feature, which can help developers rapidly integrate scanning feature into their Android app, or customize the UI for scanning as desired.
- ***FoxitMobileScanningRDK.aar*** - provides the libraries required by the scanning feature.
- ***FoxitPDFScan-UI.aar*** - provides Android Activities that implement the UI required for the scanning feature.
- ***RMSSDK-4.2-release.aar*** - used for creating rights-enabled applications. For more detailed information, please refer to https://www.microsoft.com/en-ie/download/details.aspx?id=43673.
- ***rms-sdk-ui.aar*** - provides Android Activities that implement the UI required for the RMS SDK functionality. For more detailed information, please refer to https://github.com/AzureAD/rms-sdk-ui-for-android.

Note: *In order to reduce the size of FoxitRDKUIExtensions.aar, Foxit PDF SDK for Android uses shrink-code in the uiextensions_src project. If you do not want to use shrink-code when you build the uiextensions_src project, you can disable it by setting "*minifyEnabled*" to "***false***" in the App's build.gradle. For shrink-code, you can refer to https://developer.android.com/studio/build/shrink-code.html.*

At this point you should just be getting a feel for what Foxit PDF SDK for Android package looks like, we're going to cover everything in detail in a bit.

2.3 How to run a demo

Download and install Android Studio IDE (https://developer.android.com/studio/index.html).

***Note**: In this guide, we do not cover the installation of Android Studio, Android SDK, and JDK. You can refer to Android Studio's developer site if you haven't installed it already.*

Foxit PDF SDK for Android provides three useful demos for developers to learn how to call the SDK as shown in Figure 2-3.

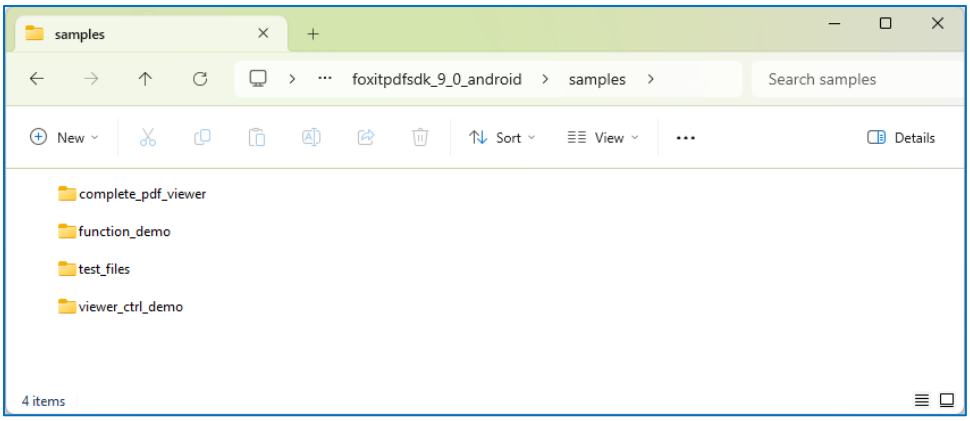


Figure 2-3

2.3.1 Function demo

The function demo is provided to show how to use Foxit PDF SDK for Android to realize some specific features related to PDF with PDF core API. This demo includes the following features:

- **pdf2txt**: extract text from a PDF document to a TXT file.
- **outline**: edit outline (aka bookmark) appearances and titles.
- **annotation**: add annotations and export annotations to a JSON file.
- **docinfo**: export document information of a PDF to a TXT file.
- **render**: render a specified page to Bitmap.
- **signature**: add a signature to PDF, sign PDF and verify the signature.
- **image2pdf**: convert images to PDF files.
- **watermark**: add text, image and PDF-page watermarks to PDF files.
- **search**: search PDF files.
- **graphics_objects**: create a PDF document with graphics objects.

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

a) Load the demo in Android Studio through "File -> New -> Import Project..." or "File -> Open...", and then navigate to where the function_demo project is stored and select the function_demo folder. Continue with "OK".

b) Launch an Android device or an emulator (AVD). In this section, an AVD targeting 10.0 will be used as an example. The test files in the "samples/test_files" that are needed for the demos will be copied to the emulator's storage card automatically when running the demos.

c) Click on "Run -> Run 'app'" to run the demo. After installing the APK on the emulator, tap **Allow** on the pop-up window to allow the demo to access files on the device. Then you can see the features are listed like Figure 2-4.

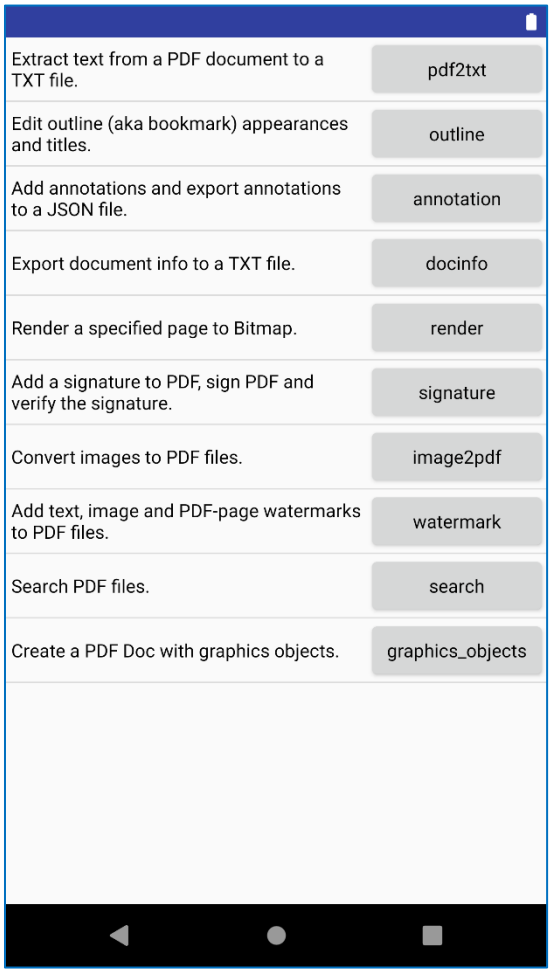


Figure 2-4

d) Click the feature buttons in the above picture to perform the corresponding actions. For example, click "pdf2txt", and then a message box will be popped up as shown in Figure 2-5. It shows where the text file was saved to. Just run the demo and try the features.

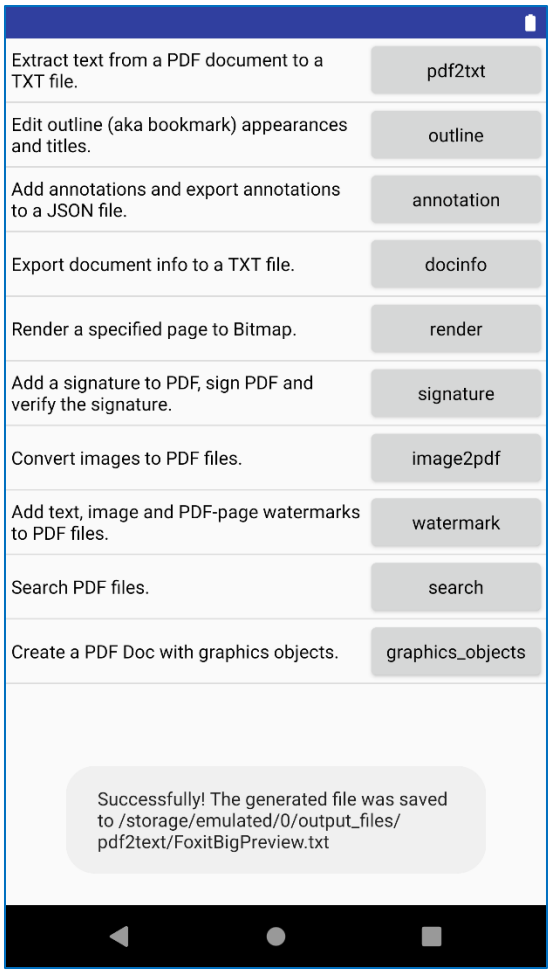


Figure 2-5

2.3.2 Viewer control demo

The viewer control demo demonstrates how to implement the features related to the View Control feature level, such as performing annotations (note, highlight, underline, strikeout, squiggly, etc.), changing layout, text search, outline, and page thumbnail. The logical structure of the code is quite clear and simple so that developers can quickly find the detailed implementation of features which are used widely in PDF apps, such as a PDF viewer. With this demo, developers can take a closer look at the APIs provided in Foxit PDF SDK for Android.

To run the demo in Android Studio, please refer to the setup steps outlined in the Function demo.

Viewer control demo will not copy the test file to the Android device or emulator automatically. It will use the "Sample.pdf" (found in the "samples/test_files" folder) as the test file, so please make sure you have pushed this file into the created folder "input_files" (or "FoxitSDK", which depends on the demo that you set) of Android device or emulator before running this demo.

Figure 2-6 shows what the demo looks like after it was built successfully. Here, an AVD targeting 10.0 will be used as an example to run the demo.



Figure 2-6

Click anywhere in the page, then the Contextual Action bar will appear, and you can click ⋮ (overflow button) to see more action items as shown in Figure 2-7.



Figure 2-7

Now we can choose one item to perform the action and see the result. For example, click "**Outline**", then you will see the outline (outline is the technical term for bookmark in the PDF specification) of the document as shown in Figure 2-8. Try using the other features to see it in action.

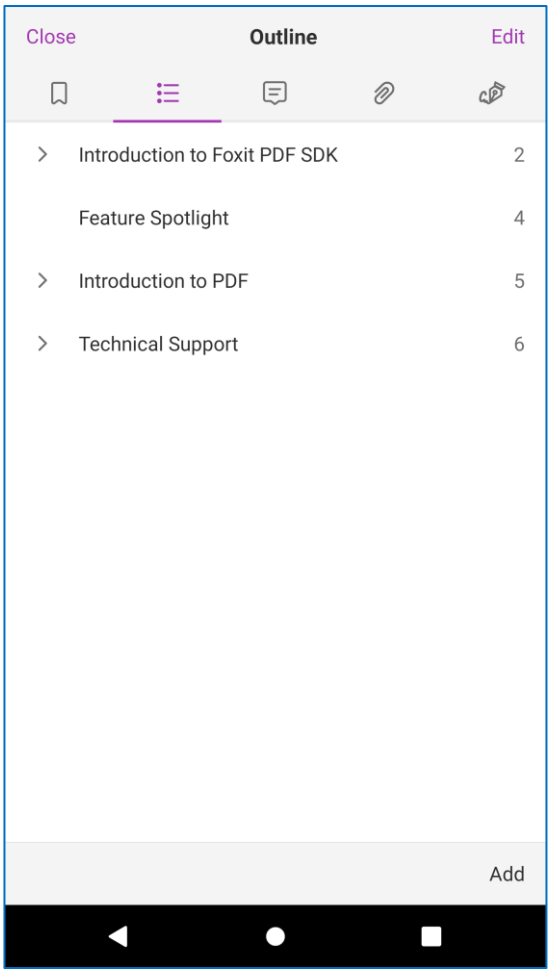


Figure 2-8

2.3.3 Complete PDF viewer demo

The complete PDF viewer demo demonstrates how to use Foxit PDF SDK for Android to realize a completely full-featured PDF viewer which is almost ready-to-use as a real world mobile PDF reader, and from version 6.0, it supported viewing multiple PDF documents. This demo utilizes all of the features and built-in UI implementations which are provided in Foxit PDF SDK for Android.

To run the demo in Android Studio, please refer to the setup steps outlined in the Function demo.

The "complete_pdf_viewer_guide_android.pdf" and "Sample.pdf" in "samples\complete_pdf_viewer\app\src\main\assets" will be copied to the "FoxitSDK" folder of the emulator automatically when running the demo.

Here, an AVD targeting 10.0 will be used as an example to run the demo. After building the demo successfully, on the start screen, it lists the "complete_pdf_viewer_guide_android.pdf" and

"Sample.pdf" documents. If you want to view multiple PDF documents, click  to switch to the tabs reading mode (see Figure 2-9).

Note *The "complete_pdf_viewer_guide_android.pdf" and "Sample.pdf" documents will be automatically deployed to your device so that you don't need to push them into the device manually. But if you want to use some other PDF files to test this demo, you should push them into the device's SD card.*

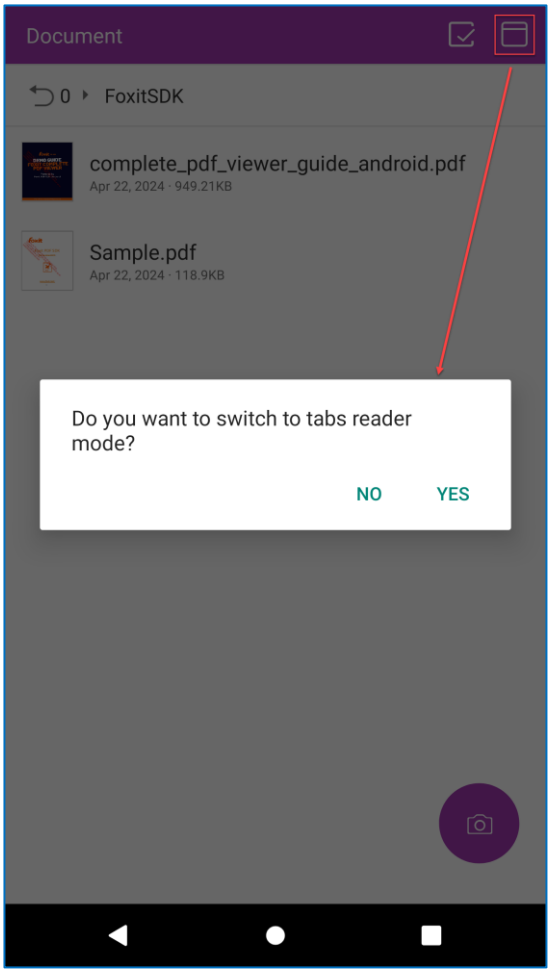


Figure 2-9

Click **YES** to switch to the tabs reading mode. Select the "complete_pdf_viewer_guide_android.pdf" document, and then click the **Back** button , and select the "Sample.pdf", then it will be displayed as shown in Figure 2-10. Now, you can browse the two PDF documents by switching the tabs.



Figure 2-10

This demo realizes a completely full-featured PDF viewer, please feel free to run it and try it.

For example, it provides the page thumbnail feature. You can click the thumbnail menu at the bottom toolbar, and then the thumbnail of the document will be displayed as shown in Figure 2-11.

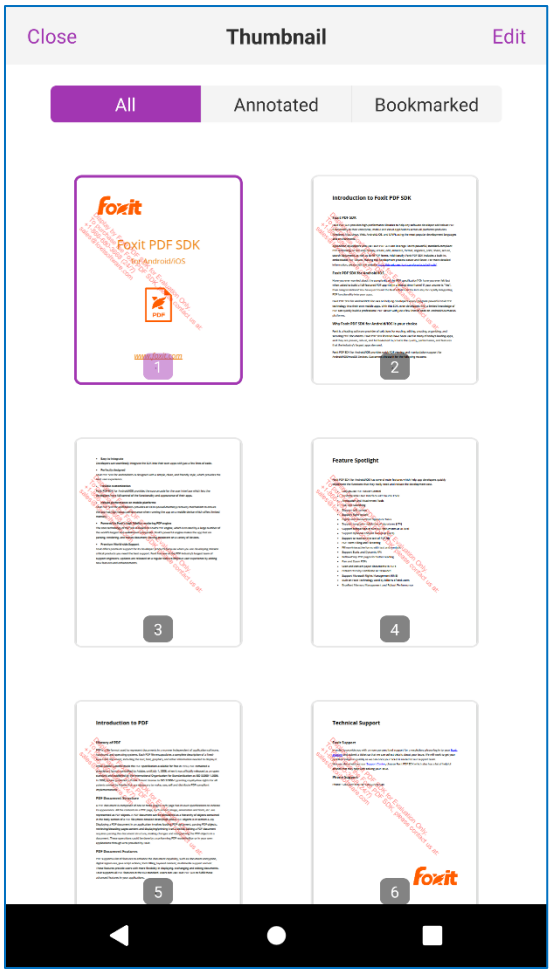

Close
Thumbnail
Edit
All
Annotated
Bookmarked
Foxit PDF SDK
for Android/iOS
1
2
3
4
5
6

Figure 2-11

3 Rapidly building a full-featured PDF Reader

Foxit PDF SDK for Android wrapped all of the UI implementations including the basic UI for app and ready-to-use UI feature modules to UI Extensions Component, so that developers can easily and rapidly build a full-featured PDF Reader with just a few lines of code. This section will help you to quickly get started with using Foxit PDF SDK for Android to make a full-featured PDF Reader app in Android platform with step-by-step instructions provided.

This section will help you to quickly make an Android app using Foxit PDF SDK for Android. It includes the following steps:

- Create a new Android project
- Integrate Foxit PDF SDK for Android into your apps
- Initialize Foxit PDF SDK for Android
- Display a PDF document using PDFViewCtrl
- Open a RMS protected document
- Build a full-featured PDF Reader with UI Extensions Component
- Add the scanning feature based on the full-featured PDF Reader

3.1 Create a new Android project

In this guide, we use Android Studio 4.1.2 along with Android API revision 30.

Open Android Studio, choose **File** -> **New** -> **New Project**... to start the **Select a Project Template** wizard, select "Empty Activity" as shown in Figure 3-1, and then click **Next**.

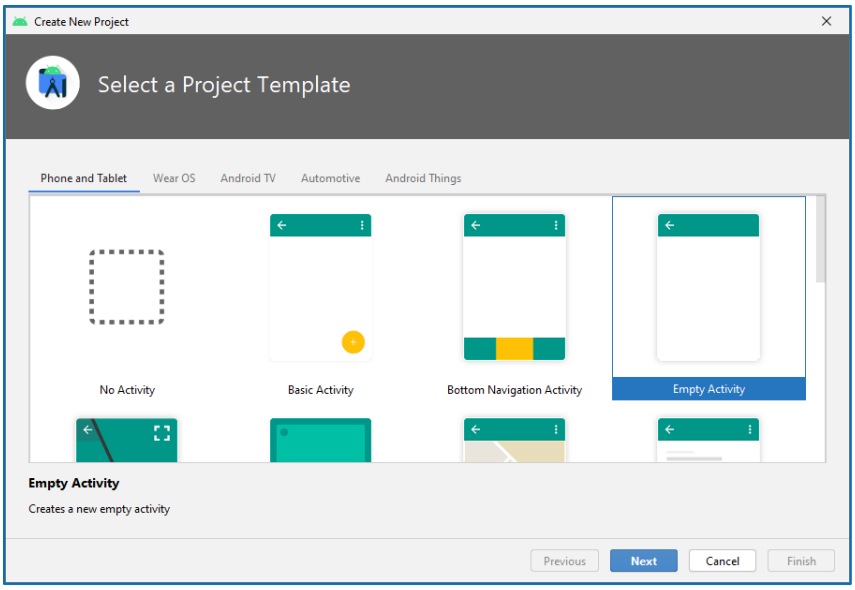


Figure 3-1

Then fill the **Configure your project** dialog as shown in Figure 3-2. After filling, click **Finish**.

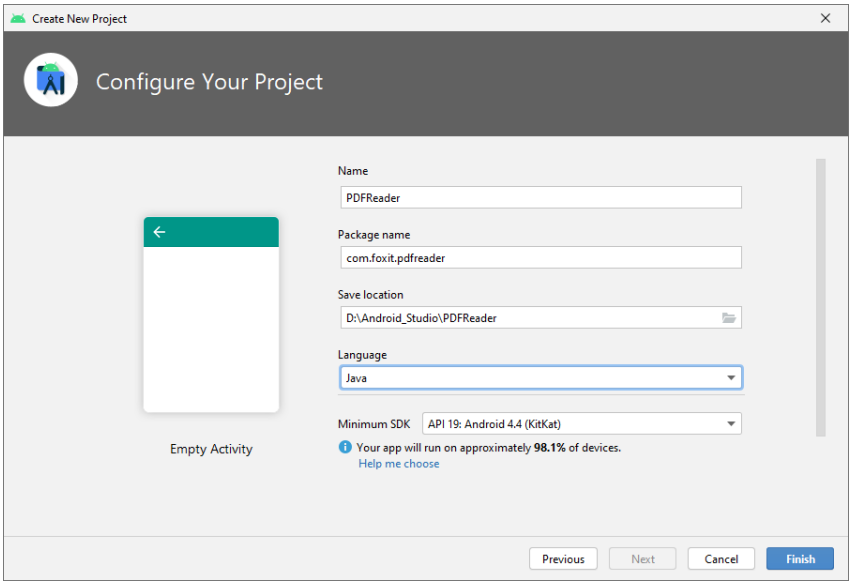


Figure 3-2

3.2 Integrate Foxit PDF SDK for Android into your apps

Note: *In this section, we will use the default built-in UI implementation to develop the app, for simplicity and convenience (use the UI Extensions Component directly, and don't need to build the source code project), we only need to add the following files to the PDFReader project.*

- ***FoxitRDK.aar*** - contains JAR package which includes all the Java APIs of Foxit PDF SDK for Android, as well as the underlying ".so" libraries. The ".so" library is the heart of the SDK including the core functionalities of Foxit PDF SDK for Android. It is built separately for each architecture, and currently available for armeabi-v7a, arm64-v8a, x86, and x86_64.

- ***FoxitRDKUIExtensions.aar*** - generated by the "**uiextensions_src**" project found in the "libs" folder. It includes the JAR package, built-in UI implementation, and resource files that are needed for the built-in UI implementations, such as images, strings, color values, layout files, and other Android UI resources.

- ***Tip:*** *The UI Extensions Component (**FoxitRDKUIExtensions.aar**) are not required for the following three sections "Initialize Foxit PDF SDK for Android", "Display a PDF document using PDFViewCtrl" and "Open a RMS protected document", so you can just add **FoxitRDK.aar** to the project at first. Then, add **FoxitRDKUIExtensions.aar** when you need to use the UI Extensions Component, such like the project described in the section "Build a full-featured PDF Reader with UI Extensions Component".*

To add the above two AAR files into *PDFReader* project, please switch to the "Project" view panel and then follow the steps below:

a) Copy and paste the "***FoxitRDK.aar***" and "***FoxitRDKUIExtensions.aar***" files from the "libs" folder of the download package to "PDFReader\app\libs" folder.

Note:

- *If you want to support Microsoft RMS, you need to copy and paste the "**RMSSDK-4.2-release.aar**" and "**rms-sdk-ui.aar**" files from the "libs" folder to "PDFReader\app\libs" folder.*

- *If you want to support scanning feature, you need to copy and paste the "**FoxitMobileScanningRDK.aar**" and "**FoxitPDFScan-UI.aar**" files from the "libs" folder to "PDFReader\app\libs" folder.*

Synchronize *PDFReader* project, and then it should look like Figure 3-3.

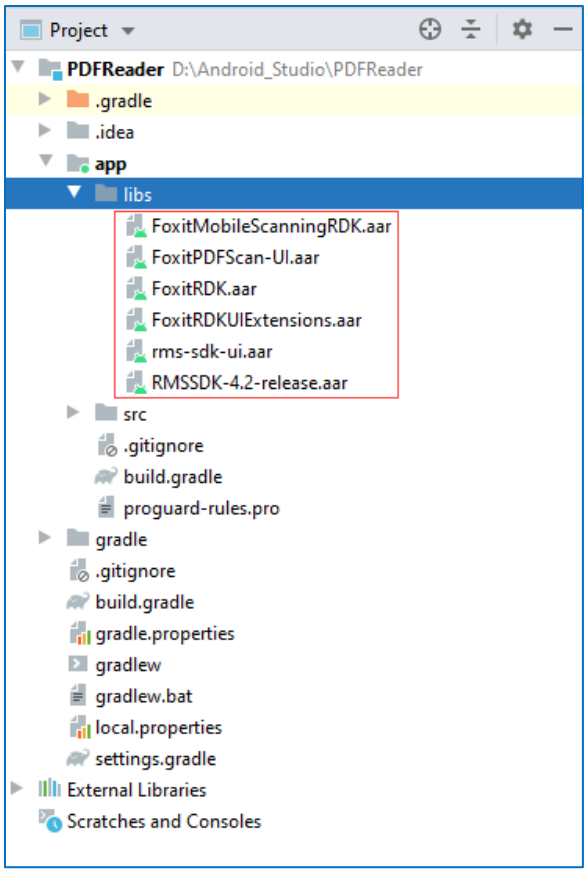


Figure 3-3

b) Define the "libs" directory as a repository. Inside the app's build.gradle file, add the following configuration:

build.gradle:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

c) Enable Multi-Dex. Add the following code into the app's build.gradle file:

```
...
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"
```

```
    defaultConfig {
        applicationId "com.foxit.pdfreader"
        minSdkVersion 19
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        multiDexEnabled true
    }

    ...
}
...

dependencies {
    implementation 'androidx.multidex:multidex:2.0.1'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
}
...
```

d) Include Foxit PDF SDK for Android as a dependency in the project. Inside the app's build.gradle, add "***FoxitRDK.aar***", "***FoxitRDKUIExtensions.aar***" and the related support libraries to the dependencies. For simplicity, update the dependencies for example as follows:

```
dependencies {
    implementation 'androidx.multidex:multidex:2.0.1'
    implementation 'androidx.appcompat:appcompat:1.3.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation(name: 'FoxitRDK', ext: 'aar')
    implementation(name: 'FoxitRDKUIExtensions', ext: 'aar')
    implementation 'com.edmodo:cropper:1.0.1'
    // RMS
    implementation 'com.microsoft.identity.client:msal:2.+'
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
    // Scanning
    implementation(name: 'FoxitPDFScan-UI', ext: 'aar')
```

```
    implementation(name: 'FoxitMobileScanningRDK', ext: 'aar')
    implementation 'com.nostra13.universalimageloader:universal-image-loader:1.9.5'
    // RxJava: Compare
    implementation "io.reactivex.rxjava2:rxjava:2.2.16"
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'
    // Signature
    implementation 'org.bouncycastle:bcpkix-jdk15on:1.64'
    implementation 'org.bouncycastle:bcprov-jdk15on:1.64'
}
```

Note:

- *(**Required**) Foxit PDF SDK for Android has a dependency on **com.google.android.material**. If it does not exist in the dependencies, please make sure to add it:*

  *implementation **'com.google.android.material:material:1.3.0'***

- *(**Optional**) If you want to use the Snapshot feature (such as in the Complete PDF viewer demo, click ••• at the right top toolbar, then you can see the **Screen Capture** option), you should add the following entry to the dependencies:*

  *implementation **'com.edmodo:cropper:1.0.1'***

- *(**Optional**) If you want to open a RMS protected PDF file, you should add the following entries to the dependencies:*

  *implementation **'com.microsoft.identity.client:msal:2.+'***
  *implementation(**name**: **'RMSSDK-4.2-release'**, **ext**: **'aar'**)*
  *implementation(**name**: **'rms-sdk-ui'**, **ext**: **'aar'**)*

  ***Note**: You had better use the verson of RMS SDK 4.2 and MSAL 2+, otherwise there may cause compatibility issues.*

  Please also add the following lines to your repositories section in your project-level "build.gradle" file:

```
allprojects {
    repositories {
        google()
        jcenter()
        maven {
            url 'https://pkgs.dev.azure.com/MicrosoftDeviceSDK/DuoSDK-
Public/_packaging/Duo-SDK-Feed/maven/v1'
        }
```

```
    }
}
```

For more details, please refer to the https://github.com/AzureAD/microsoft-authentication-library-for-android.

- *(**Optional**) If you want to use the scanning feature (such as in the start screen of Complete PDF viewer demo, you can see the scanning feature button), you should add the following entries to the dependencies:*

  *implementation(name: **'FoxitMobileScanningRDK'**, ext: **'aar'**)*
  *implementation(name: **'FoxitPDFScan-UI'**, ext: **'aar'**)*
  *implementation **'com.nostra13.universalimageloader:universal-image-loader:1.9.5'***

- *(**Optional**) If you want to use the comparison feature (such as in the start screen of Complete PDF viewer demo, click at the right top toolbar, then you can see the comparison feature button), you should add the following entries to the dependencies:*

  *implementation **"io.reactivex.rxjava2:rxjava:2.2.16"***
  *implementation **'io.reactivex.rxjava2:rxandroid:2.1.1'***

- *(**Optional**) If you want to use the signature feature, you should add the following entries to the dependencies:*

  *implementation **'org.bouncycastle:bcpkix-jdk15on:1.64'***
  *implementation **'org.bouncycastle:bcprov-jdk15on:1.64'***

Here, we add all the above support libraries to the dependencies, because later we will build a full-featured PDF Reader which contains all the features provided by Foxit PDF SDK for Android. After setting the app's "build.gradle" file, sync it, then the "***FoxitRDK.aar***", "***FoxitRDKUIExtensions.aar***", "***FoxitMobileScanningRDK.aar***", "***FoxitPDFScan-UI.aar***", "***universal-image-loader***", "***material***", "***cropper***", "***rxjava***", "***rxandroid*** **,** "***bcpkix-jdk15on***", "***bcprov-jdk15on***" and ***RMS*** related pacakges will appear in **External Libraries.**

The following code shows the app's "build.gradle" in its entirety.

build.gradle:

```
plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 30
```

```
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.foxit.pdfreader"
        minSdkVersion 19
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        multiDexEnabled true

    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    implementation 'androidx.multidex:multidex:2.0.1'
    implementation 'androidx.appcompat:appcompat:1.3.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation(name: 'FoxitRDK', ext: 'aar')
    implementation(name: 'FoxitRDKUIExtensions', ext: 'aar')
```

```
    implementation 'com.edmodo:cropper:1.0.1'
    // RMS
    implementation 'com.microsoft.identity.client:msal:2.+'
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
    // Scanning
    implementation(name: 'FoxitPDFScan-UI', ext: 'aar')
    implementation(name: 'FoxitMobileScanningRDK', ext: 'aar')
    implementation 'com.nostra13.universalimageloader:universal-image-loader:1.9.5'
    // RxJava: Compare
    implementation "io.reactivex.rxjava2:rxjava:2.2.16"
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'
    // Signature
    implementation 'org.bouncycastle:bcpkix-jdk15on:1.64'
    implementation 'org.bouncycastle:bcprov-jdk15on:1.64'
}
```

Note: *For using AndroidX, the `compileSdkVersion` should be set to at least 28. In this project, we set the `compileSdkVersion` and `targetSdkVersion` to API 30. So, if you also want to use API 30, please make sure you have already installed the SDK Platform Android 11.0, API 30. If you have not already done this, open the Android SDK Manager to download and install it first.*

3.3 Initialize Foxit PDF SDK for Android

It is necessary for apps to initialize and unlock Foxit PDF SDK for Android using a license before calling any APIs. The function **Library.*initialize****(sn, key)* is provided to initialize Foxit PDF SDK for Android. The trial license files can be found in the "libs" folder of the download package. After the evaluation period expires, you should purchase an official license to continue using it. Below you can see an example of how to unlock the SDK library. The next section will show you where to include this code in the *PDFReader* project.

```
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;
...

int errorCode = Library.initialize("sn", "key");
if (errorCode != Constants.e_ErrSuccess)
    return;
```

Note *The parameter "sn" can be found in the "**rdk_sn.txt**" (the string after "SN=") and the "key" can be found in the "**rdk_key.txt**" (the string after "Sign=").*

3.4 Display a PDF document using PDFViewCtrl

So far, we have added Foxit PDF SDK for Android libraries to the *PDFReader* project, and finished the initialization of the Foxit PDF SDK. Now, let's start displaying a PDF document using PDFViewCtrl with just a few lines of code.

Note: *The UI Extensions Component is not required if you only want to display a PDF document.*

To display a PDF document, please follow the steps below:

a) Instantiate a PDFViewCtrl object to show an existing document.

In MainActivity.java, instantiate a PDFViewCtrl object, and call **PDFViewCtrl.openDoc** function to open and render the PDF document.

```
import com.foxit.sdk.PDFViewCtrl;
import android.os.Environment;

...

private PDFViewCtrl pdfViewCtrl = null;
...
pdfViewCtrl = new PDFViewCtrl(this);

String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
pdfViewCtrl.openDoc(path, null);

setContentView(pdfViewCtrl);
```

Note*: Please make sure you have pushed the "Sample.pdf" document into the created folder "FoxitSDK" of the Android device or emulator that will be used to run this project.*

Update MainActivity.java as follows:

```
package com.foxit.pdfreader;

import android.os.Bundle;
import android.os.Environment;
import androidx.appcompat.app.AppCompatActivity;

import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;
```

```
public class MainActivity extends AppCompatActivity {

    private PDFViewCtrl pdfViewCtrl = null;

    // The value of "sn" can be found in the "rdk_sn.txt".
    // The value of "key" can be found in the "rdk_key.txt".
    private static String sn = " ";
    private static String key = " ";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // initialize the library.
        int errorCode = Library.initialize(sn, key);
        if (errorCode != Constants.e_ErrSuccess)
            return;

        pdfViewCtrl = new PDFViewCtrl(this);
        String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
        pdfViewCtrl.openDoc(path, null);

        setContentView(pdfViewCtrl);
    }
}
```

b) Set permissions to write and read the SD card of the Android devices or emulators. You should write additional code to require the authorization of runtime permissions.

In the **MainActivity.java** file, add the code below:

```
import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Build;
import android.os.Environment;

import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.annotation.NonNull;
```

```
...

private static final int REQUEST_EXTERNAL_STORAGE = 1;
private static final String[] PERMISSIONS_STORAGE = {
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.WRITE_EXTERNAL_STORAGE
};
...

// Require the authorization of runtime permissions.
 if (Build.VERSION.SDK_INT >= 23) {
    int permission = ContextCompat.checkSelfPermission(this.getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if (permission != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
        return;
    }
}
...

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[]
grantResults) {
    if (requestCode == REQUEST_EXTERNAL_STORAGE && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        // Open and Reader a PDF document.
        String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
        pdfViewCtrl.openDoc(path, null);
        setContentView(pdfViewCtrl);
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

Then, the whole contents of **MainActivity.java** will be as follows:

```
package com.foxit.pdfreader;

import android.Manifest;
import android.content.pm.PackageManager;
```

```
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.annotation.NonNull;

import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;

public class MainActivity extends AppCompatActivity {

  private PDFViewCtrl pdfViewCtrl = null;

  private static final int REQUEST_EXTERNAL_STORAGE = 1;
  private static final String[] PERMISSIONS_STORAGE = {
      Manifest.permission.READ_EXTERNAL_STORAGE,
      Manifest.permission.WRITE_EXTERNAL_STORAGE
  };

  // The value of "sn" can be found in the "rdk_sn.txt".
  // The value of "key" can be found in the "rdk_key.txt".
  private static String sn = " ";
  private static String key = " ";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Initialize the library.
    int errorCode = Library.initialize(sn, key);
    if (errorCode != Constants.e_ErrSuccess)
      return;

    // Instantiate a PDFViewCtrl object.
    pdfViewCtrl = new PDFViewCtrl(this);
```

```
        // Require the authorization of runtime permissions.
        if (Build.VERSION.SDK_INT >= 23) {
            int permission = ContextCompat.checkSelfPermission(this.getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
            if (permission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
                return;
            }
        }

        // Open and Render a PDF document.
        String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
        pdfViewCtrl.openDoc(path, null);
        setContentView(pdfViewCtrl);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        if (requestCode == REQUEST_EXTERNAL_STORAGE && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            // Open and Render a PDF document.
            String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
            pdfViewCtrl.openDoc(path, null);
            setContentView(pdfViewCtrl);
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

c) Update the AndroidManifest.xml file under the "PDFReader\app\src\main" directory as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.foxit.pdfreader">
```

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:requestLegacyExternalStorage="true"
    tools:replace="android:theme"
    android:theme="@style/Theme.PDFReader">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

***Note**: If you run the project on an Android 10 device or emulator, please add "**android:requestLegacyExternalStorage="true"**" to request using the legacy storage mode.*

In this section, we build and run the project on an AVD targeting 10.0 (API 29).

Now, we have finished building a simple Android app which uses Foxit PDF SDK for Android to display a PDF document with just a few lines of code. The next step is to run the project.

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see that the "Sample.pdf" document is displayed as shown in Figure 3-4. Now, this sample app has some basic PDF features, such as zooming in/out and page turning. Just have a try!



Figure 3-4

3.5 Open a RMS protected document

From version 6.2.1, Foxit PDF SDK for Android supports to open a PDF document protected by Micorosoft Rights Management (RMS).

To open a RMS protected document, you should notice the following key points:

1) Include RMS related libraries as a dependency in the project. Please refer to the section "Integrate Foxit PDF SDK for Android into your apps".

2) Set the associated activity before opening a PDF document, because it has UI operations when opening a RMS protected document.

```
pdfViewCtrl.setAttachedActivity(activity);
```

3) Handle the activity result from the UI operations. Call the API "**pdfViewCtrl**.handleActivityResult()" on the related **onActivityResult** function.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
    super.onActivityResult(requestCode, resultCode, data);
    pdfViewCtrl.handleActivityResult(requestCode, resultCode, data);
}
```

Based on the previous section "Display a PDF document using PDFViewCtrl", update the whole contents of **MainActivity.java** as follows:

```
package com.foxit.pdfreader;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.annotation.NonNull;

import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;

public class MainActivity extends AppCompatActivity {

    private PDFViewCtrl pdfViewCtrl = null;

    private static final int REQUEST_EXTERNAL_STORAGE = 1;
    private static final String[] PERMISSIONS_STORAGE = {
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE
    };

    // The value of "sn" can be found in the "rdk_sn.txt".
    // The value of "key" can be found in the "rdk_key.txt".
    private static String sn = " ";
    private static String key = " ";
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // initialize the library.
    int errorCode = Library.initialize(sn, key);
    if (errorCode != Constants.e_ErrSuccess)
        return;

    // Instantiate a PDFViewCtrl object.
    pdfViewCtrl = new PDFViewCtrl(this);

    // Set the associated activity for RMS UI operations.
    pdfViewCtrl.setAttachedActivity(this);

    // Require the authorization of runtime permissions.
    if (Build.VERSION.SDK_INT >= 23) {
        int permission = ContextCompat.checkSelfPermission(this.getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
        if (permission != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
            return;
        }
    }

    // Open and Render a PDF document.
    String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample_RMS.pdf";
    pdfViewCtrl.openDoc(path, null);
    setContentView(pdfViewCtrl);
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
    if (requestCode == REQUEST_EXTERNAL_STORAGE && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        // Open and Render a PDF document.
        String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample_RMS.pdf";
        pdfViewCtrl.openDoc(path, null);
        setContentView(pdfViewCtrl);
```

```
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
  }

    @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    pdfViewCtrl.handleActivityResult(requestCode, resultCode, data);
  }
}
```

***Note**: Please make sure you have pushed a RMS protected document for example "Sample_RMS.pdf" into the created folder "FoxitSDK" of the Android device or emulator that will be used to run this project.*

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see the following window (Figure 3-5) which prompts you to input your organizational email, and later input the password, then you can open the RMS protected document.

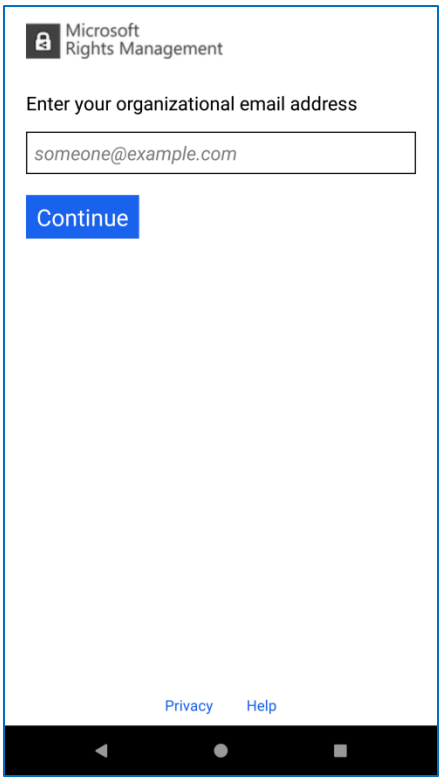


Figure 3-5

3.6 Build a full-featured PDF Reader with UI Extensions Component

Foxit PDF SDK for Android comes with built-in UI design including the basic UI for app and the feature modules UI, which are implemented using Foxit PDF SDK for Android and are shipped in the UI Extensions Component. Hence, building a full-featured PDF Reader is getting simpler and easier. All you need to do is to instantiate a UIExtensionsManager object, and then set it to PDFViewCtrl.

Instantiate a UIExtensionsManager object and set it to PDFViewCtrl

In "MainActivity.java" file, we are now going to add the code necessary for including the UIExtensionsManager. The required code additions are shown below and further down you will find a full example of what the "MainActivity.java" file should look like.

a) Set the system theme to *"No Title"* mode and set the window to *Fullscreen*.

***Note**: The UI Extensions Component has customized the user interface, so you need to set the system theme to "No Title" mode and set the window to Fullscreen. Otherwise, the layout of the built-in features might be affected.*

```
import android.view.Window;
import android.view.WindowManager;
...

// Turn off the title at the top of the screen.
this.requestWindowFeature(Window.FEATURE_NO_TITLE);

// Set the window to Fullscreen.
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

*Please make sure to set the theme style to "**Theme.MaterialComponents.Light.NoActionBar**" in the "PDFReader\app\src\main\res\values**themes.xml**" file as below:*



b) Add code to instantiate a UIExtensionsManager object and set it to PDFViewCtrl.

```
import com.foxit.uiextensions.UIExtensionsManager;
...

private UIExtensionsManager uiExtensionsManager = null;
...

uiExtensionsManager = new UIExtensionsManager(this.getApplicationContext(), pdfViewCtrl);
uiExtensionsManager.setAttachedActivity(this);
uiExtensionsManager.onCreate(this, pdfViewCtrl, savedInstanceState);
pdfViewCtrl.setUIExtensionsManager(uiExtensionsManager);
```

c) Open and render a PDF document, and set the content view.

Call **UIExtensionsManager.openDocument()** function to open and render a PDF document instead of calling PDFViewCtrl.openDoc() function.

```
import android.os.Environment;
```

```
import com.foxit.uiextensions.UIExtensionsManager;
...

String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
uiExtensionsManager.openDocument(path, null);
setContentView(uiExtensionsManager.getContentView());
```

Update MainActivity.java as follows:

Note: *The Activity Lifecycle Events should be handled as below, otherwise some features may not work correctly.*

```
package com.foxit.pdfreader;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.res.Configuration;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.view.KeyEvent;
import android.view.Window;
import android.view.WindowManager;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.annotation.NonNull;

import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;
import com.foxit.uiextensions.UIExtensionsManager;


public class MainActivity extends AppCompatActivity {

    private PDFViewCtrl pdfViewCtrl = null;
    private UIExtensionsManager uiExtensionsManager = null;

    private static final int REQUEST_EXTERNAL_STORAGE = 1;
    private static final String[] PERMISSIONS_STORAGE = {
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE
    };
```

```
    // The value of "sn" can be found in the "rdk_sn.txt".
    // The value of "key" can be found in the "rdk_key.txt".
    private static String sn = " ";
    private static String key = " ";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // initialize the library.
        int errorCode = Library.initialize(sn, key);
        if (errorCode != Constants.e_ErrSuccess)
            return;

        // Turn off the title at the top of the screen.
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);

        // Set the window to Fullscreen.
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

        // Instantiate a PDFViewCtrl object.
        pdfViewCtrl = new PDFViewCtrl(this);

        // Set the associated activity for RMS UI operations.
        pdfViewCtrl.setAttachedActivity(this);

        // Initialize a UIExtensionManager object and set it to PDFViewCtrl.
        uiExtensionsManager = new UIExtensionsManager(this.getApplicationContext(), pdfViewCtrl);
        uiExtensionsManager.setAttachedActivity(this);
        uiExtensionsManager.onCreate(this, pdfViewCtrl, savedInstanceState);
        pdfViewCtrl.setUIExtensionsManager(uiExtensionsManager);

        // Require the authorization of runtime permissions.
        if (Build.VERSION.SDK_INT >= 23) {
            int permission = ContextCompat.checkSelfPermission(this.getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
            if (permission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
                return;
```

```
        }
    }

    // Open and Render a PDF document.
    String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
    uiExtensionsManager.openDocument(path, null);
    setContentView(uiExtensionsManager.getContentView());
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[]
grantResults) {
    if (requestCode == REQUEST_EXTERNAL_STORAGE && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        // Open and Render a PDF document.
        String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
        uiExtensionsManager.openDocument(path, null);
        setContentView(uiExtensionsManager.getContentView());
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}

    @Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    pdfViewCtrl.handleActivityResult(requestCode, resultCode, data);
}

@Override
public void onStart() {
    if (uiExtensionsManager != null) {
        uiExtensionsManager.onStart(this);
    }
    super.onStart();
}

@Override
public void onStop() {
    if (uiExtensionsManager != null) {
```

```
        uiExtensionsManager.onStop(this);
    }
    super.onStop();
}

@Override
public void onPause() {
    if (uiExtensionsManager != null) {
        uiExtensionsManager.onPause(this);
    }
    super.onPause();
}

@Override
public void onResume() {
    if (uiExtensionsManager != null) {
        uiExtensionsManager.onResume(this);
    }
    super.onResume();
}

@Override
protected void onDestroy() {
    if (uiExtensionsManager != null) {
        uiExtensionsManager.onDestroy(this);
    }
    super.onDestroy();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    if (uiExtensionsManager != null) {
        uiExtensionsManager.onConfigurationChanged(this, newConfig);
    }
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
```

```
    if (uiExtensionsManager != null && uiExtensionsManager.onKeyDown(this, keyCode, event))
      return true;
    return super.onKeyDown(keyCode, event);
  }
}
```

Update AndroidManifest.xml

Add "<**uses-permission android:name="android.permission.CAMERA"**/>" to grant the project permissions to access the camera.

Add "<**uses-permission android:name="android.permission.RECORD_AUDIO"**/>" to grant the project permissions to record audio or video. If you do not add it, the audio and video features will not work correctly.

Add "<**uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"**/>" to grant your phone the floating window permissions. If you do not add this permission, the Pan and Zoom feature will not be able to work.

Add
"**android:configChanges="keyboardHidden|orientation|locale|layoutDirection|screenSize"**>" property to make sure that the project will only execute the onConfigurationChanged() function without recalling the activity lifecycle when rotating the screen. If you do not add this property, the signature feature will not work correctly.

Update AndroidManifest.xml as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  package="com.foxit.pdfreader">

  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission android:name="android.permission.RECORD_AUDIO" />
  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
```

```
    android:supportsRtl="true"
    android:requestLegacyExternalStorage="true"
    tools:replace="android:theme"
    android:theme="@style/Theme.PDFReader">
    <activity android:name=".MainActivity"
      android:configChanges="keyboardHidden|orientation|locale|layoutDirection|screenSize">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Run the project

In this section, we build and run the project on an AVD targeting Android 10.0 (API 29). After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see that the "Sample.pdf" document is displayed as shown in Figure 3-6. Up to now, it is a full-featured PDF Reader which includes all of the features in Complete PDF viewer demo and it also supports to open a RMS protected document. Feel free to try it.



Figure 3-6

3.7 Add the scanning feature based on the full-featured PDF Reader

The scanning feature is a stand-alone module which is not shipped in the UI Extensions Component, so if you want to use this feature in your project, you should add the core code below to call the scan module:

```
import com.foxit.pdfscan.PDFScanManager;
import androidx.fragment.app.DialogFragment;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;
...

// Initialize the scan module.
long framework1 = 0;
long framework2 = 0;
PDFScanManager.initializeScanner(this.getApplication(), framework1, framework2);

long compression1 = 0;
long compression2 = 0;
PDFScanManager.initializeCompression(this.getApplication(), compression1, compression2);

if (PDFScanManager.isInitializeScanner() && PDFScanManager.isInitializeCompression()) {
    final PDFScanManager pdfScanManager = PDFScanManager.instance();
```

```
    pdfScanManager.showUI(MainActivity.this);
} else {
    Toast.makeText(getApplicationContext(), " ", Toast.LENGTH_SHORT).show();
}
```

For **PDFScanManager.initializeScanner** and **PDFScanManager.initializeCompression** interfaces, if you set the second and third parameters to 0, then the scanned image will be with watermark. If you do not want to have watermark, you should contact Foxit sales or support team to get the license key.

Based on the previous section, we add a new button to call the scan module.

Update MainActivity.java as follows:

(Assuming that you have copied the "**fx_floatbutton_scan.xml**" from "samples\complete_pdf_viewer\app\src\main\res\drawable" to "PDFReader\app\src\main\res\drawable")

```
package com.foxit.pdfreader;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.res.Configuration;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.view.KeyEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.view.WindowManager;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.Toast;

import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;
import com.foxit.uiextensions.UIExtensionsManager;
import com.foxit.pdfscan.PDFScanManager;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

public class MainActivity extends AppCompatActivity {
```

```
    private PDFViewCtrl pdfViewCtrl = null;
    private UIExtensionsManager uiExtensionsManager = null;

    private static final int REQUEST_EXTERNAL_STORAGE = 1;
    private static final String[] PERMISSIONS_STORAGE = {
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE
    };

    // The value of "sn" can be found in the "rdk_sn.txt".
    // The value of "key" can be found in the "rdk_key.txt".
    private static String sn = " ";
    private static String key = " ";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Initialize the library.
        int errorCode = Library.initialize(sn, key);
        if (errorCode != Constants.e_ErrSuccess)
            return;

        // Turn off the title at the top of the screen.
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);

        // Set the window to Fullscreen.
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

        // Instantiate a PDFViewCtrl object.
        pdfViewCtrl = new PDFViewCtrl(this);

        // Set the associated activity for RMS UI operations.
        pdfViewCtrl.setAttachedActivity(this);

        uiExtensionsManager = new UIExtensionsManager(this.getApplicationContext(), pdfViewCtrl);
        uiExtensionsManager.setAttachedActivity(this);
        uiExtensionsManager.onCreate(this, pdfViewCtrl, savedInstanceState);
        pdfViewCtrl.setUIExtensionsManager(uiExtensionsManager);

        // Require the authorization of runtime permissions.
        if (Build.VERSION.SDK_INT >= 23) {
            int permission = ContextCompat.checkSelfPermission(this.getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
            if (permission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
                return;
            }
        }
```

```
        // Open and Render a PDF document.
        String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
        uiExtensionsManager.openDocument(path, null);

        RelativeLayout rootView = new RelativeLayout(getApplicationContext());
        RelativeLayout.LayoutParams layoutParams = new
RelativeLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup
                .LayoutParams.MATCH_PARENT);
        rootView.addView(uiExtensionsManager.getContentView(), layoutParams);
        rootView.addView(getScanButton());
        setContentView(rootView);
    }

    // Get a scan button.
    private View getScanButton() {
        ImageView ivScan = new ImageView(this);
        ivScan.setImageResource(R.drawable.fx_floatbutton_scan);
        ivScan.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                if (!PDFScanManager.isInitializeScanner()) {
                    long framework1 = 0;
                    long framework2 = 0;
                    PDFScanManager.initializeScanner(MainActivity.this.getApplication(), framework1, framework2);
                }

                if (!PDFScanManager.isInitializeCompression()) {
                    long compression1 = 0;
                    long compression2 = 0;
                    PDFScanManager.initializeCompression(MainActivity.this.getApplication(), compression1,
compression2);
                }

                if (PDFScanManager.isInitializeScanner() && PDFScanManager.isInitializeCompression()) {
                    final PDFScanManager pdfScanManager = PDFScanManager.instance();
                    pdfScanManager.showUI(MainActivity.this);
                } else {
                    Toast.makeText(getApplicationContext(), "You are not authorized to use this add-on module,
please contact us for upgrading your license.", Toast.LENGTH_SHORT).show();
                }
            }
        });

        RelativeLayout.LayoutParams layoutParams = new
RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
RelativeLayout.LayoutParams.WRAP_CONTENT);

        layoutParams.bottomMargin = 150;
        layoutParams.rightMargin = 50;
        layoutParams.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
```

```
        layoutParams.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
        ivScan.setLayoutParams(layoutParams);
        return ivScan;
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[]
grantResults) {
        if (requestCode == REQUEST_EXTERNAL_STORAGE && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            // Open and Render a PDF document.
            String path = Environment.getExternalStorageDirectory().getPath() + "/FoxitSDK/Sample.pdf";
            uiExtensionsManager.openDocument(path, null);
            setContentView(uiExtensionsManager.getContentView());
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        pdfViewCtrl.handleActivityResult(requestCode, resultCode, data);
    }

    @Override
    public void onStart() {
        if (uiExtensionsManager != null) {
            uiExtensionsManager.onStart(this);
        }
        super.onStart();
    }

    @Override
    public void onStop() {
        if (uiExtensionsManager != null) {
            uiExtensionsManager.onStop(this);
        }
        super.onStop();
    }

    @Override
    public void onPause() {
        if (uiExtensionsManager != null) {
            uiExtensionsManager.onPause(this);
        }
        super.onPause();
    }

    @Override
    public void onResume() {
```

```
        if (uiExtensionsManager != null) {
            uiExtensionsManager.onResume(this);
        }
        super.onResume();
    }

    @Override
    protected void onDestroy() {
        if (uiExtensionsManager != null) {
            uiExtensionsManager.onDestroy(this);
        }
        super.onDestroy();
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        if (uiExtensionsManager != null) {
            uiExtensionsManager.onConfigurationChanged(this, newConfig);
        }
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (uiExtensionsManager != null && uiExtensionsManager.onKeyDown(this, keyCode, event))
            return true;
        return super.onKeyDown(keyCode, event);
    }
}
```

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see the following window (Figure 3-7), tap the scan button to start scanning documents.



Figure 3-7

3.8 Handle Scoped Storage

The release of Android 11 offers improvements to scoped storage. With Scoped Storage, things are both more restrictive and easier at the same time. A compatible app is given its own folder for user-facing data. Apps already have a private sandboxed folder for storage of their required files and this is unavailable to any other app. Scoped Storage gives the ability to create a second folder for files the app creates.

If you want to **disable** the scoped storage in your project, you can do any of the following:

- In the App's **build.gradle**, set the targetSdkVersion <=28.
- In the App's **build.gradle**, set the targetSdkVersion =29, and in the **AndroidManifest.xml** file, add "**android:requestLegacyExternalStorage="true"**" in the application node.

If targetSdkVersion >= 30, you can add the following configurations to reduce the effects of scoped storage to your project:

In the **AndroidManifest.xml:**

➢ Add "<**uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"**/>" in the manifest node.

- Add "**android:requestLegacyExternalStorage="true"**" and "**android:preserveLegacyExternalStorage="true"**" in the application node.

If you need to adapt to scoped storage when scoped storage is enabled, please refer to the complete PDF Viewer demo for details.

In the **MainActivity.java**, you should add the code below to require the authorization of runtime permissions:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
    if (!Environment.isExternalStorageManager()) {
        Intent intent = new Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION);
        intent.setData(Uri.parse("package:" + getApplicationContext().getPackageName()));
        startActivityForResult(intent, REQUEST_ALL_FILES_ACCESS_PERMISSION);
        return;
    }
} else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
    int permission = ContextCompat.checkSelfPermission(this.getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if (permission != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
        return;
    }
}
```

For more information, you can refer to the MainActivity.java file of the Viewer Control Demo.

4 Customizing User Interface

Foxit PDF SDK for Android provides a simple, clean and friendly user interface for developers to quickly build a full-featured PDF app without needing to take much time on the design. Furthermore, customizing the user interface is straightforward. Foxit PDF SDK for Android provides the source code of the UI Extensions Component that contains ready-to-use UI module implementations, which lets the developers have full control of styling the appearance as desired.

From version 4.0, developers can flexibly customize the features they want through a configuration file.

From version 5.0, every element in the built-in UI can be configurable. More advanced APIs and more powerful configuration file are provided for developers to further customize the UI elements, such as showing or hiding a specific panel, top/bottom toolbar, the items in the top/bottom toolbar, and the items in the View setting bar and More Menu view.

From version 6.3, the configuration file has been enhanced which provides more optional settings to customize the UI including the rights management and the properties of UI elements.

From version 8.0, the built-in UI in the UI Extensions Component has changed dramatically.

The following section will introduce how to customize the feature modules, rights management and UI elements through a configuration file, or APIs, or the source code.

4.1 Customize the UI through a configuration file

Through a configuration file, developers can easily choose the feature modules, set the rights management and the properties of UI elements without needing to write any additional code or redesign the app's UI.

4.1.1 Introduction to JSON file

The configuration file can be provided as a JSON file or implemented directly in code. We recommend you to use the JSON format which is more intuitive and clearer to view and configure the items.

You can refer to the JSON file found in "samples\complete_pdf_viewer\app\src\main\res\raw" folder of Foxit PDF SDK for Android package. It looks like as follows:

```
{
   "modules": {
      "readingbookmark": true,
      "outline": true,
//       "annotations":true,
      "annotations": {
         "highlight": true,
         "underline": true,
         "squiggly": true,
         "strikeout": true,
         "insert": true,
         "replace": true,
         "line": true,
         "rectangle": true,
         "oval": true,
         "arrow": true,
         "pencil": true,
         "eraser": true,
         "typewriter": true,
         "textbox": true,
         "callout": true,
         "note": true,
         "stamp": true,
         "polygon": true,
         "cloud": true,
         "polyline": true,
         "measure": true,
         "image": true,
         "audio": true,
         "video": true,
         "redaction": true
      },
      "thumbnail": true,
      "attachment": true,
      "signature": true,
      "fillSign": true,
      "search": true,
```

```
    "navigation": true,
    "form": true,
    "selection": true,
    "encryption": true
    "multipleSelection": true
  },
  "permissions": {
      "runJavaScript": true,
      "copyText": true,
      "disableLink": false
  },
  "uiSettings": {
    "pageMode": "Single",
    "continuous": false,
    "reflowBackgroundColor": "#FFFFFF",
    "zoomMode": "FitWidth",
    "colorMode": "Normal",
    "mapForegroundColor": "#5d5b71",
    "mapBackgroundColor": "#00001b",
    "disableFormNavigationBar": false,
    "highlightForm": true,
    "highlightFormColor": "#200066cc",
    "highlightLink": true,
    "highlightLinkColor": "#16007fff",
    "fullscreen": true,
    "showPenOnlySwitch": true, // Only use for pencil and eraser
    "enableTopbarDraggable": 2,
    "annotations": {
          "continuouslyAdd": true,
          "highlight": {
            "color": "#ffff00", "opacity": 1.0
          },
          "areaHighlight": {
            "color": "#ffff00", "opacity": 1.0
          },
          "underline": {
            "color": "#66cc33", "opacity": 1.0
          },
          "squiggly": {
            "color": "#993399", "opacity": 1.0
```

```
    },
    "strikeout": {
      "color": "#ff0000", "opacity": 1.0
    },
    "insert": {
      "color": "#993399", "opacity": 1.0
    },
    "replace": {
      "color": "#0000ff", "opacity": 1.0
    },
    "line": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2
    },
    "rectangle": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null
    },
    "oval": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null
    },
    "arrow": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2
    },
    "pencil": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2
    },
    "polygon": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null
    },
    "cloud": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null
    },
    "polyline": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2
    },
    "typewriter": {
      "textColor": "#0000ff",
      "opacity": 1.0,
      "textFace": "Courier",
      "textSize": 18
```

```
    },
    "textbox": {
      "color": "#ff0000",
      "textColor": "#0000ff",
      "opacity": 1.0,
      "textFace": "Courier",
      "textSize": 18
    },
    "callout": {
      "color": "#ff0000",
      "textColor": "#0000ff",
      "opacity": 1.0,
      "textFace": "Courier",
      "textSize": 18
    },
    "note": {
      "color": "#ff0000",
      "opacity": 1.0,
      "icon": "Comment"
    },
    "attachment": {
      "color": "#ff0000",
      "opacity": 1.0,
      "icon": "PushPin"
    },
    "image": {
       "rotation": 0,
      "opacity": 1.0
    },
    "measure": {
       "color": "#ff0000",
       "opacity": 1.0,
       "thickness": 2,
       "scaleFromUnit": "inch",
       "scaleToUnit": "inch",
       "scaleFromValue": 1,
       "scaleToValue": 1
    },
```

```
        "redaction": {
          "fillColor": "#000000",
          "textColor": "#ff0000",
          "textFace": "Courier",
          "textSize": 12
        }
    },
   "form": {
    "textField": {
      "textColor": "#000000",
      "textFace": "Courier", // textFace:"Courier"/"Helvetica"/"Times"
      "textSize": 0 // 0 means auto-adjust font size (textSize >= 0)
    },
    "checkBox": {
      "textColor": "#000000"
    },
    "radioButton": {
      "textColor": "#000000"
    },
    "comboBox": {
      "textColor": "#000000",
      "textFace": "Courier", // textFace:"Courier"/"Helvetica"/"Times"
      "textSize": 0, // 0 means auto-adjust font size (textSize >= 0)
      "customText": false
    },
    "listBox": {
      "textColor": "#000000",
      "textFace": "Courier", // textFace:"Courier"/"Helvetica"/"Times"
      "textSize": 0, // 0 means auto-adjust font size (textSize >= 0)
      "multipleSelection": false
    }
   },
   "signature": {
       "color": "#000000", "thickness": 8
   }
  }
}
```

Note:

- *The values in the above JSON file are the default settings for the configuration items. If some configuration items are not in the JSON file, the default settings will be used. For example, if you comment out "**"highlight"**: **true**,", it is still enabled.*

- *Only the attachment annotation is not controlled by the subitems in "**annotations**". Click the **Home** button at the top toolbar to select the **Comment**, then you can find the attachment annotation, which is as shown in Figure 4-1.*

  *"**"attachment"**: **true**," controls the attachments panel and attachment annotation. If you set it to "false", both of them will be disabled. If you want to hide all the tools in the **Comment**, you should set both "**annotations**" and "**attachment**" to "**false**".*



Figure 4-1

4.1.2 Configuration Items Description

The JSON configuration file includes three parts: feature modules, rights management, and UI settings (for example, UI elements properties). This section will set forth the configuration items in detail.

Configure feature module

***Note**: The value type of the feature module items is **bool**, where "**true**" means that the feature module will be enabled, and "**false**" means that the feature module will be disabled. The default value is "true".*

Feature Module	Description
readingbookmark	User-defined bookmark
outline	PDF document bookmark
annotations (highlight, underline, squiggly, strikeout, insert, replace, line, rectangle, oval, arrow, pencil, eraser, typewriter, textbox, callout, note, stamp, polygon, cloud, polyline, measure, image, audio, video, redaction)	Annotation module collection
thumbnail	PDF page thumbnail display and page management
attachment	PDF document attachments
signature	Digital signatures and handwritten signatures
fillSign	Fill flat forms (i.e. non-interactive forms) with text and symbols.
search	Text search
navigation	PDF page navigation
form	Form Filling and form data importing and exporting
selection	Text selection
encryption	PDF encryption
multipleSelection	Multiple annotations selection

Configure rights management

***Note**: The value type of the configuration items is **bool**, where "**true**" means that the permission will be enabled, and "**false**" means that the permission will be disabled. The default value of **runJavaScript** and **copyText** is "true", and the default value of **disableLink** is "false".*

Rights Management	Description
runJavaScript	whether to allow to execute JavaScript
copyText	whether to allow to copy text
disableLink	whether to disable hyperlink

Configure UI settings

UI Items	Description/ Property Items	Value Type	Available Value	Default value	Note
pageMode	Page display mode	String	Single/ Facing/ CoverLeft/ CoverMiddle/ CoverRight/ Reflow	Single	For dynamic XFA files, it doesn't support Reflow mode.
continuous	Whether to view pages continuously	Bool	true/false	false	True means continuous pages, false means discontinuous pages. It is invalid for "Reflow" mode.
reflowBackgroundColor	Background color of reflow page	RGB	---	#FFFFFF	
zoomMode	Page zoom mode	String	FitWidth/FitPage	FitWidth	
colorMode	Page color display mode	String	Normal/Night/Map	Normal	"Night" is a special "Map" mode.
mapForegroundColor	Foreground color of page display	RGB	---	#5d5b71	It is vaild only when "colorMode" is set to "Map".
mapBackgroundColor	Background color of page dispay	RGB	---	#00001b	It is vaild only when "colorMode" is set to "Map".
disableFormNavigationBar	Whether to disable the supplementary navigation bar of the form	Bool	true/false	false	
highlightForm	Whether to highlight form field	Bool	true/false	true	

UI Items	Description/ Property Items	Value Type	Available Value	Default value	Note
highlightFormColor	The highlight color of forms	ARGB		#200066cc	It include alpha channel, and it is invalid for dynamic xfa document.
highlightLink	Whether to highlight hyperlink	Bool	true/false	true	
highlightLinkColor	The highlight color of links	ARGB		#16007fff	It include alpha channel.
fullscreen	Whether to display in full screen mode	Bool	true/false	true	It will be in full screen mode immediately when opening a document if "fullscreen" is set to "true". If the user clicks on the page, the toolbar will be displayed. After 5 seconds, if it is in full screen mode, the toolbar and other auxiliary tool buttons will be hidden automatically.
showPenOnlySwitch	Whether to show the pen only switch for pencil/ highlighter/ eraser	Bool	true/false	true	
enableTopbarDraggable	Whether to enable dragging topbar	Integer	0,1,2,3	2	0: the topbar dragging is disabled. 1: the topbar dragging is enabled only for Phone.

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
						2: the topbar dragging is enabled only for Tablet. 3: the topbar dragging is enabled for both Phone and Tablet.
annotations	continuouslyAdd		Bool	true/false	true	Whether to add annotation continuously
	highlight	color	RGB		#ffff00	
		opacity	numeric	[0.0-1.0]	1.0	
	areaHighlight	color	RGB		#ffff00	
		opacity	numeric	[0.0-1.0]	1.0	Use the default configuration if out of range.
	underline	color	RGB		#66cc33	
		opacity	numeric	[0.0-1.0]	1.0	
	squiggly	color	RGB		#993399	
		opacity	numeric	[0.0-1.0]	1.0	
	strikeout	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
	insert	color	RGB		#993399	
		opacity	numeric	[0.0-1.0]	1.0	
	replace	color	RGB		#0000ff	
		opacity	numeric	[0.0-1.0]	1.0	
	line	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
	rectangle	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
		fillColor	RGB		null	
	oval	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
		thickness	numeric	[1-12]	2	
		fillColor	RGB		null	
	arrow	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
	pencil	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
	polygon	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
		fillColor	RGB		null	
	cloud	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
		fillColor	RGB		null	
	polyline	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
	typewriter	textColor	RGB		#0000ff	
		opacity	numeric	[0.0-1.0]	1.0	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=1	18	
	textbox	color	RGB		#ff0000	
		textColor	RGB		#0000ff	
		opacity	numeric	[0.0-1.0]	1.0	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=1	18	
	callout	color	RGB		#ff0000	

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
		textColor	RGB		#0000ff	
		opacity	numeric	[0.0-1.0]	1.0	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=1	18	
	note	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		icon	String	Comment/ Key/ Note/ Help/ NewParagraph/ Paragraph/ Insert	Comment	If set to an invalid value, the default value will be used.
	attachment	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		icon	String	Graph/ PushPin/ Paperclip/ Tag	PushPin	If set to an invalid value, the default value will be used.
	image	rotation	numeric	0/90/180/270	0	
		opacity	numeric	[0.0-1.0]	1.0	
	measure	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
		scaleFromUnit	String	pt/m/cm/mm/inch/p/ft/yd	inch	The original unit of the scale
		scaleToUnit	String	pt/m/cm/mm/inch/p/ft/yd	inch	The target unit of the scale
		scaleFromValue	numeric		1	The original value of the scale
		scaleToValue	numeric		1	The target value of the scale

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
	redaction	fillColor	RGB		#000000	
		textColor	RGB		#ff0000	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=1	12	
form	textField	textColor	RGB		#000000	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=0	0	0 means adjusting the font size automatically.
	checkBox	textColor	RGB		#000000	
	radioButton	textColor	RGB		#000000	
	comboBox	textColor	RGB		#000000	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=0	0	0 means adjusting the font size automatically.
		customText			false	Whether to allow to customize text.
	listBox	textColor	RGB		#000000	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
		textSize	Integer	>=0	0	0 means adjusting the font size automatically.
		multipleSelection			false	Whether to allow to support multiple selection.
signature		color	RGB		#000000	
		thickness	numeric	[1-12]	8	

4.1.3 Instantiate a UIExtensionsManager object with the configuration file

In section 3.6, we have already introduced how to instantiate UIExtensionsManager, and in this way all the built-in UI framework would be loaded by default. In this section, we will provide another method to instantiate UIExtensionsManager that uses the configuration file, so that developers can easily customize the UI as desired.

Please refer to the following code to instantiate a UIExtensionsManager object with the configuration file.

Note: *Here, we assume that you have already put a JSON file named "uiextensions_config.json" to "PDFReader\app\src\main\res\raw" (note that you need to create the "raw" folder by yourself).*

In "MainActivity.java":

```
import com.foxit.uiextensions.config.Config;
...

private PDFViewCtrl pdfViewCtrl = null;
private UIExtensionsManager uiExtensionsManager = null;
// Initialize a PDFViewCtrl object.
pdfViewCtrl = new PDFViewCtrl(this);

// Get the config file, and set it to UIExtensionsManager.
InputStream stream =
this.getApplicationContext().getResources().openRawResource(R.raw.uiextensions_config);
Config config = new Config(stream);

// Initialize a UIExtensionManager object with Configuration file, and set it to PDFViewCtrl.
```

```
uiExtensionsManager = new UIExtensionsManager(this.getApplicationContext(), pdfViewCtrl,config);
pdfViewCtrl.setUIExtensionsManager(uiExtensionsManager);
uiExtensionsManager.setAttachedActivity(this);
uiExtensionsManager.onCreate(this, pdfViewCtrl, savedInstanceState);
```

Note: *Here, we use a configuration file to instantiate the UIExtensionsManager. If you do not want to use configuration file, please refer to the section 3.6.*

4.1.4 Examples for customizing UI through a configuration file

In this section, we will show you how to customize feature modules, rights management and UI settings (for example, UI elements properties) in your project. You will find it is extremely easy! You only need to modify the configuration file. Below you can see some examples of how to do it.

Note*: For your convenience, we will try it in the "**complete_pdf_viewer**" demo found in the "samples" folder.*

Load the "**complete_pdf_viewer**" demo in Android Studio. Find the configuration file "uiextensions_config.json" under "complete_pdf_viewer\app\src\main\res\raw".

Example1: Disable "readingbookmark" and "navigation" feature modules.

In the JSON file, set the values of "readingbookmark" and "navigation" to "false" as follows:

```
"readingbookmark": false,
"navigation": false,
```

Then, rebuild and run the demo to see the result. Following lists the comparison diagrams:

Before: After:





The **"readingbookmark"** and **"navigation"** feature modules are removed.

Example2: Disable hyperlinks.

In the JSON file, set the value of "disableLink" to "true" as follows:

```
"permissions": {
    "runJavaScript": true,
    "copyText": true,
    "disableLink": true
},
```

Then, rebuild and run the demo to see the result, and you will find that there is no any response when clicking the hyperlinks.

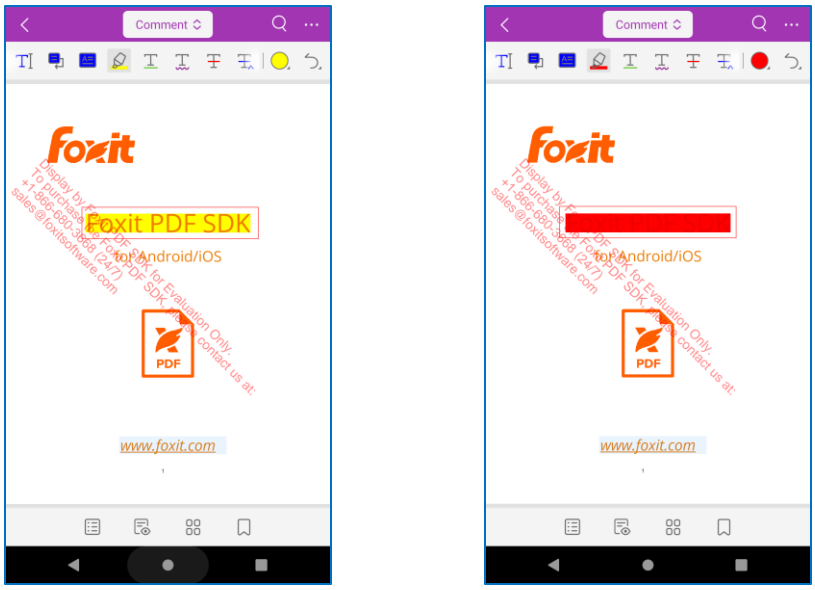
Example3: Set the highlight color from yellow to red.

In the JSON file, set the color property of "highlight" to "#ff0000" as follows:

```
"highlight": {
  "color": "#ff0000", "opacity": 1.0 },
```

Then, rebuild and run the demo to see the result. Following lists the comparison diagrams:

Before: After:



The highlight color has been changed to red.

4.2 Customize UI elements through APIs

In version 4.0, Foxit PDF SDK for Android supported customizing to show or hide the whole top or bottom toolbar, and from version 5.0, it provided APIs to customize to show or hide a specific panel, the items in the top/bottom toolbar, View setting bar and More Menu view, which is convenient for developers to modify the UI elements in the context of the built-in UI framework.

From version 8.0, the built-in UI in the UI Extensions Component has changed dramatically.

***Note**: For your convenience, we will show you how to customize UI elements through APIs in the "**complete_pdf_viewer**" demo found in the "samples" folder. We assume that you have not modified the "uiextensions_config.json" file in the demos, which means that all of the built-in UI in the UI Extensions Component are enabled.*

4.2.1 Customize top/bottom toolbar

In the top/bottom toolbar, you can do the following operations:

1. Show or hide the top/bottom toolbar.
2. Add a custom item at any position.
3. Remove a specific item.
4. Remove all of the items in the toolbar.

5. Show or hide a specific item.
6. Add a custom toolbar.
7. Remove a specific toolbar.
8. Set background color for the toolbar.
9. Get the number of the items in a specific location of the toolbar.
10. Remove a specific tab in the center of the top toolbar.

Table 4-1 lists the related APIs which are used to customize the top/bottom toolbar.

Table 4-1

void **enableTopToolbar**(boolean isEnabled)	Enable or disable top toolbar.
void **enableBottomToolbar**(boolean isEnabled)	Enable or disable bottom toolbar.
boolean **addItem**(BarName barName, BaseBar.TB_Position gravity, BaseItem item, int index);	Add a custom item to the toolbar.
boolean **addItem**(BarName barName, BaseBar.TB_Position gravity, int textId, int resId, int index, IItemClickListener clickListener);	Add a default item to the toolbar.
boolean **addItem**(BarName barName, BaseBar.TB_Position gravity, CharSequence text, int index, IItemClickListener clickListener);	Add a default text-only item to the toolbar.
boolean **addItem**(BarName barName, BaseBar.TB_Position gravity, Drawable drawable, int index, IItemClickListener clickListener);	Add a default image-only item to the toolbar.
IBaseItem **getItemByIndex**(BarName barName, BaseBar.TB_Position gravity, int index);	Get the item by index.
void **setItemVisibility**(BarName barName, BaseBar.TB_Position gravity, int index, int visibility);	Set the enabled state of this item view.
int **getItemVisibility**(BarName barName, BaseBar.TB_Position gravity, int index);	Returns the visibility status for this view.
int **getItemsCount**(BarName barName, BaseBar.TB_Position gravity);	Get the items count by IBarsHandler.BarName and BaseBar.TB_Position.
boolean **removeItem**(BarName barName, BaseBar.TB_Position gravity, int index);	Remove the item at the specified position in the toolbar.
boolean **removeItem**(BarName barName, BaseBar.TB_Position gravity, BaseItem item);	Remove the item by IBarsHandler.BarName, BaseBar.TB_Position and the specified item.
void **removeAllItems**(BarName barName);	Remove all items from the toolbar.

boolean **addCustomToolBar**(BarName barName, View view);	Add custom toolbar by BarName.
boolean **removeToolBar**(BarName barName);	Remove toolbar by BarName.
void **setBackgroundColor**(BarName barName, int color);	Set background color for the toolbar.
void **setBackgroundResource**(BarName barName, int resid);	Set background to a given resource.
BaseItem **getItem**(BarName barName, BaseBar.TB_Position gravity, int tag);	Get the item by tag, if tag does not exist, it returns null.
void removeTab (int type)	Remove the tab in the center of the topbar and its sub-toolbar.

There are two important enumerations which are defined to locate the position that you want to add a new item or remove an existing item.

```
enum BarName {
      TOP_BAR,
      BOTTOM_BAR;
   }

enum TB_Position {
      Position_LT,
      Position_CENTER,
      Position_RB;
   }
```

Note:

1. For Tablet device, it has removed the bottom toolbar.
2. To add an item to the top toolbar, please set the **BaseBar.TB_Position** to **Position_LT** or **Position_RB.** To add an item to the bottom toolbar, please set the **BaseBar.TB_Position** to **Position_CENTER**. Otherwise, the items may overlap.
3. For Phone device, the bottom toolbar is only one part, and the top toolbar is divided into two parts, so that there are three parts for the toolbar, and each part has a separate index. For Tablet device, it has no bottom toolbar. (See Figure 4-2)
4. For the best UI display, it is recommended to control the character number of text for top toolbar within 15 and for bottom toolbar within 8. If don't, the view layout may be in disorder.

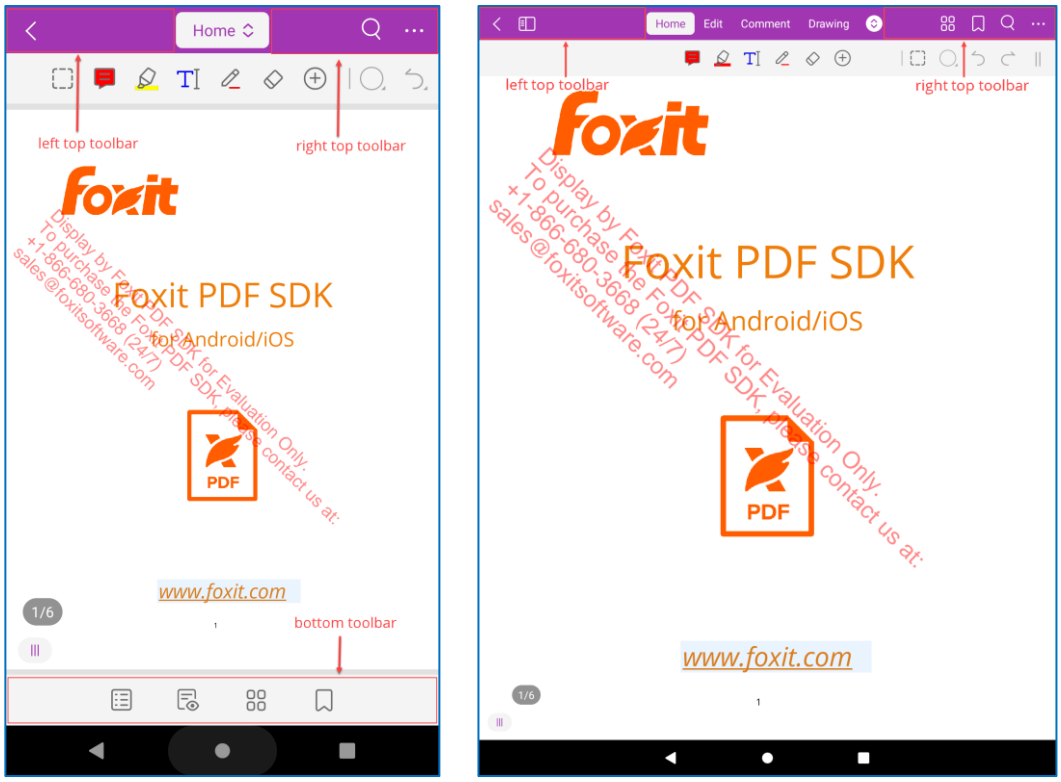


Figure 4-2

In the following examples, we will show you how to customize the top/bottom toolbar through APIs in the "**complete_pdf_viewer**" demo found in the "samples" folder.

Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "**mUiExtensionsManager** = **new** UIExtensionsManager(getActivity().getApplicationContext(), **pdfViewerCtrl**, config);").

***Note**: The built-in UI is a bit different on tablets and phones. Some of the following examples are applicable for phones and tablets, and some are only applicable for phones. In this guide, if the custom results on phones and tablets are similar, we only list the result on phones.*

Example1: Hide the whole top toolbar. (For Phone and Tablet)

```
mUiExtensionsManager.enableTopToolbar(false);
```

Before:



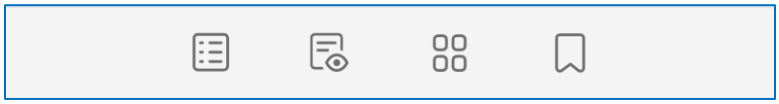
After:



Example2: Hide the whole bottom toolbar. (Only for Phone)

```
mUiExtensionsManager.enableBottomToolbar(false);
```

Before:



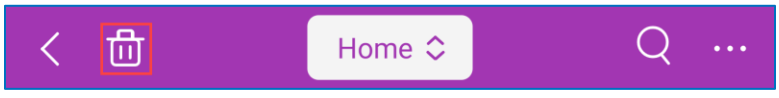
After:



Example3: Add an item in the left top toolbar at the second position. (For Phone and Tablet)

```
BaseItemImpl mTopItem1 = new BaseItemImpl(getContext(), R.drawable.rd_delete_menu);
mTopItem1.setImageTintList(ColorStateList.valueOf(Color.WHITE));
mTopItem1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        UIToast.getInstance(getActivity()).show("Add an item in the left top toolbar at the second position.");
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, mTopItem1, 1);
```

The result after running the demo:



Example4: Add an item in the right top toolbar at the first position. (For Phone and Tablet)

```
BaseItemImpl mTopItem2 = new BaseItemImpl(getContext(), R.drawable.rd_delete_menu);
```

```
mTopItem2.setImageTintList(ColorStateList.valueOf(Color.WHITE));
mTopItem2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        UIToast.getInstance(getActivity()).show("Add an item in the right top toolbar at the first position");
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_RB, mTopItem2, 0);
```

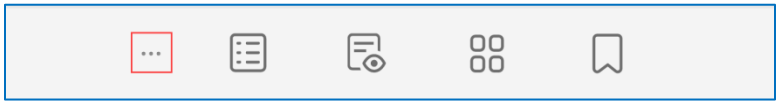
The result after running the demo:



Example5: Add an item to the bottom toolbar at the first position. (Only for Phone)

```
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.BOTTOM_BAR,
BaseBar.TB_Position.Position_CENTER,
        "", R.drawable.ic_bar_item_more, 0, new IBarsHandler.IItemClickListener() {
            @Override
            public void onClick(View v) {
                UIToast.getInstance(getActivity()).show("Add an item to the bottom toolbar at the first position.");
            }
        });
```

The result after running the demo:

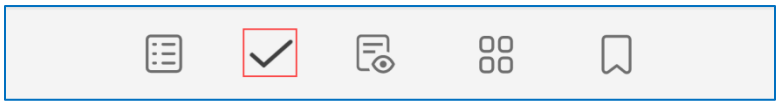


Example6: Add an item with custom style to the bottom toolbar at the second position. (Only for Phone)

```
BaseItemImpl mSettingBtn = new BaseItemImpl(this.getContext(), R.drawable.rd_annot_create_ok_selector);
mSettingBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        UIToast.getInstance(getActivity()).show("Add an item with custom style to the bottom toolbar at the
second position.");
    }
```

```
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.BOTTOM_BAR,
BaseBar.TB_Position.Position_CENTER, mSettingBtn, 1);
```

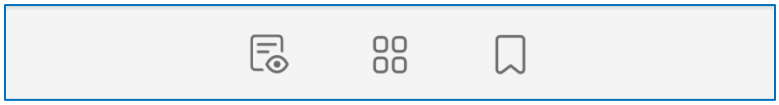
The result after running the demo:



Example7: Remove an item by index (remove the first item in the bottom toolbar). (Only for Phone)

```
mUiExtensionsManager.getBarManager().removeItem(IBarsHandler.BarName.BOTTOM_BAR,
BaseBar.TB_Position.Position_CENTER,0);
```

The result after running the demo:



Example8: Remove an item by BaseItem object (remove a custom item from the top toolbar that you added before). (For Phone and Tablet)

```
mUiExtensionsManager.getBarManager().removeItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, mTopItem1);
```

Before: (See **Example3**)



After:



Example9: Remove all the items in the bottom toolbar. (Only for Phone)

```
mUiExtensionsManager.getBarManager().removeAllItems(IBarsHandler.BarName.BOTTOM_BAR);
```

Before:



After:



Example10: Add two items in the left top toolbar to control to show and hide the "more menu" item. (For Phone and Tablet)

```
// Get and save the item that you want to show or hide.
final IBarsHandler barsHandler = (IBarsHandler) mUiExtensionsManager.getBarManager();
final IBaseItem moreItem = barsHandler.getItemByIndex(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_RB, 1);

// Add a buttom in the left top toolbar to hide the "moreItem" item.
BaseItemImpl topItem = new BaseItemImpl(getContext(), R.drawable.rd_delete_menu);
topItem.setImageTintList(ColorStateList.valueOf(Color.WHITE));
topItem.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    // Hide the "moreItem" item.
    mUiExtensionsManager.getBarManager().removeItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_RB, moreItem);
  }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, topItem, 1);

// Add a buttom in the left top toolbar to show the "moreItem" item.
BaseItemImpl topItem2 = new BaseItemImpl(getContext(), R.drawable.common_add_icon);
topItem2.setImageTintList(ColorStateList.valueOf(Color.WHITE));
topItem2.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    // Show the "moreItem" item.
    if (AppDisplay.isPad())
      mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_RB, moreItem, 3);
```

```
    else
        mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_RB, moreItem, 1);
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, topItem2, 2);
```

The result after running the demo, the top toolbar will look like as follows:



Click , then the "more menu" will be hidden as follows:



Click , then the "more menu" will appear as follows:



Example11: Remove the whole bottom toolbar. (Only for Phone)

```
mUiExtensionsManager.getBarManager().removeToolBar(IBarsHandler.BarName.BOTTOM_BAR);
```

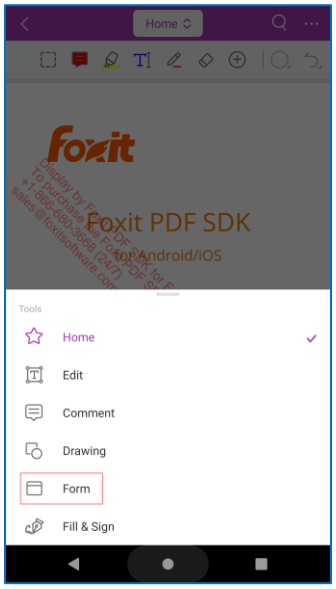
Example12: Add a custom toolbar. (add a custom layout file "test_top_layout") (For Phone and Tablet)

```
View topView = View.inflate(getContext(), R.layout.test_top_layout, null);
mUiExtensionsManager.getBarManager().addCustomToolBar(IBarsHandler.BarName.TOP_BAR, topView);
```

Example13: Remove the "Form" tab from the list in the center of the top toolbar. (For Phone and Tablet)

```
mUiExtensionsManager.getMainFrame().removeTab(ToolbarItemConfig.ITEM_FORM_TAB);
```

Before:



After:



Example14: Remove the "View" item in the bottom toolbar for Phone, or remove the "View" tab from the list in the center of the top toolbar for Table. (For Phone and Tablet)

```
if (AppDisplay.isPad())
    mUiExtensionsManager.getMainFrame().removeTab(ToolbarItemConfig.ITEM_VIEW_TAB);
else
    mUiExtensionsManager.getBarManager().removeItem(IBarsHandler.BarName.BOTTOM_BAR,
BaseBar.TB_Position.Position_CENTER,1);
```

for phone:

Before:



After:



for tablet:

Before:



After:



4.2.2 Customize to add or remove a specific panel

You can add a custom panel or remove a specific panel through the APIs listed in the Table 4-2. The Complete PDF Viewer demo includes "Bookmarks", "Outline", "Annotations", "Attachments" and "Digital Signatures" panels, just taps at the bottom toolbar (for phones) or taps at the left top toolbar (for tablets) to find it, see Figure 4-3.

Table 4-2

Public void **addPanel** (PanelSpec panelSpec)	Add a child panel view. If the panel type PanelSpec#getPanelType() already exists, will not be added again.
public void **addPanel** (int index, PanelSpec panelSpec)	Add a child panel view at the specified position in the group. If the panel type PanelSpec#getPanelType() already exists, will not be added again.
public void **removePanel** (int panelType)	Remove the child panel view at the specified position in the group.
public void **removePanel** (PanelSpec panelSpec)	Remove the child panel view.

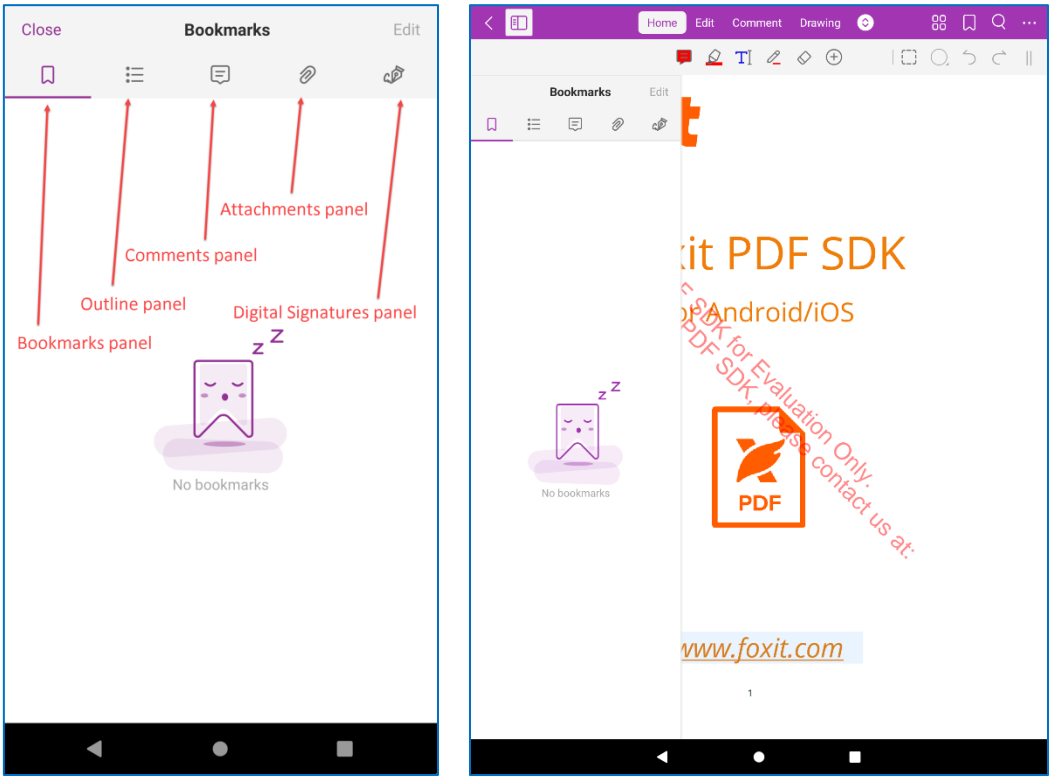


Figure 4-3

In this section, we only give two examples:

- **Example1** shows you how to remove a specific panel through APIs in the "**complete_pdf_viewer**" demo found in the "samples" folder. Just take the "Outline" panel as an example, and for other panels, you only need to change the **PanelSpec**. The corresponding relation between panels and **PanelSpec** are as follows:

Panel	PanelSpec	integer
Bookmarks	PanelSpec.BOOKMARKS	0
Outline	PanelSpec.OUTLINE	1
Comments	PanelSpec.ANNOTATIONS	2
Attachments	PanelSpec.ATTACHMENTS	3
Digital Signatures	PanelSpec.SIGNATURES	4

- **Example2** shows you how to add a custom panel through APIs. First you should create a Java class (named **CustomPanel**, for example) that implements the "**PanelSpec.java**" interface. After finishing implementing the related methods and events, then add the panel.

Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "**mUiExtensionsManager** = **new** UIExtensionsManager(getActivity().getApplicationContext(), **pdfViewerCtrl**, config);").

Example1: Add an item in the left top toolbar at the second position to remove the "Outline" panel. (For Phone and Tablet)

```
BaseItemImpl removePanel = new BaseItemImpl(getActivity(), R.drawable.rd_delete_menu);
removePanel.setImageTintList(ColorStateList.valueOf(Color.WHITE));
removePanel.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    mUiExtensionsManager.getPanelManager().removePanel(PanelSpec.OUTLINE);
  }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, removePanel, 1);
```

After running the demo, tap  at the top toolbar, and tap  at the bottom toolbar (for phones) or tap  at the left top toolbar (for tablets), then you will see the "Outline" panel has been hidden (See Figure 4-4).

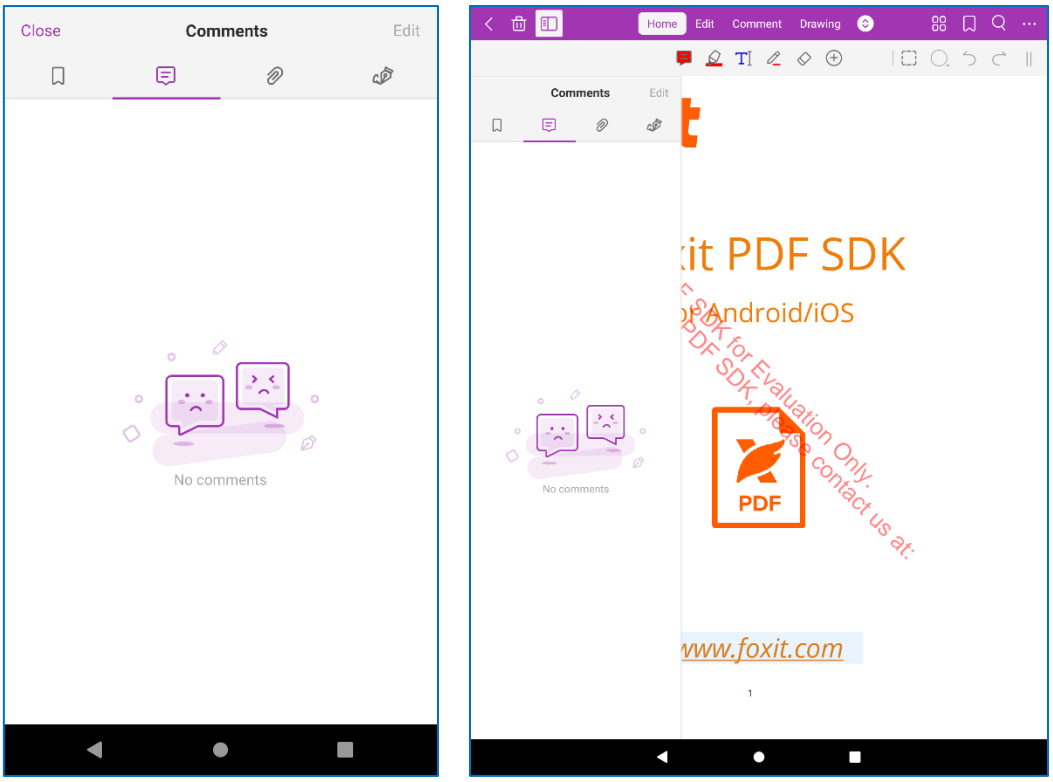


Figure 4-4

Example2: Add an item in the left top toolbar at the second position to add a custom panel. (For Phone and Tablet)

```
BaseItemImpl customPanel = new BaseItemImpl(getActivity(), R.drawable.common_add_icon);
customPanel.setImageTintList(ColorStateList.valueOf(Color.WHITE));
customPanel.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mUiExtensionsManager.getPanelManager().addPanel(new CustomPanel(getContext()));
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, customPanel, 1);
```

Assume that you have aready created a custom panel. Create a Java class named **CustomPanel** that implements the "**PanelSpec.java**" interface:

```
package com.foxit.home;

import android.content.Context;
import android.content.res.ColorStateList;
```

```
import android.view.View;
import android.widget.TextView;

import com.foxit.uiextensions.controls.panel.PanelSpec;
import com.foxit.uiextensions.controls.toolbar.BaseBar;
import com.foxit.uiextensions.controls.toolbar.IBaseItem;
import com.foxit.uiextensions.controls.toolbar.impl.BaseItemImpl;
import com.foxit.uiextensions.controls.toolbar.impl.TopBarImpl;

public class CustomPanel implements PanelSpec {

  private Context mContext;

  public CustomPanel(Context context){
    mContext= context;
  }

  @Override
  public int getIcon() {
    return R.drawable.toolbar_thumbnail_icon;
  }

  @Override
  public ColorStateList getIconTint() {
    return null;
  }

  @Override
  public int getPanelType() {
    return 100;
  }

  @Override
  public View getTopToolbar() {
    TopBarImpl topBar = new TopBarImpl(mContext);
    IBaseItem baseItem = new BaseItemImpl(mContext);
    baseItem.setText("Custom Panel");
    topBar.addView(baseItem, BaseBar.TB_Position.Position_CENTER);
    return topBar.getContentView();
  }

  @Override
  public View getContentView() {
    TextView textView = new TextView(mContext);
    textView.setText("This is empty");
    return textView;
  }

  @Override
  public void onActivated() {
```

```
    }

    @Override
    public void onDeactivated() {

    }
}
```

After running the demo, tap  at the top toolbar, and tap  at the bottom toolbar (for phones) or tap  at the left top toolbar (for tablets), then you will see the custom panel as shown in Figure 4-5.

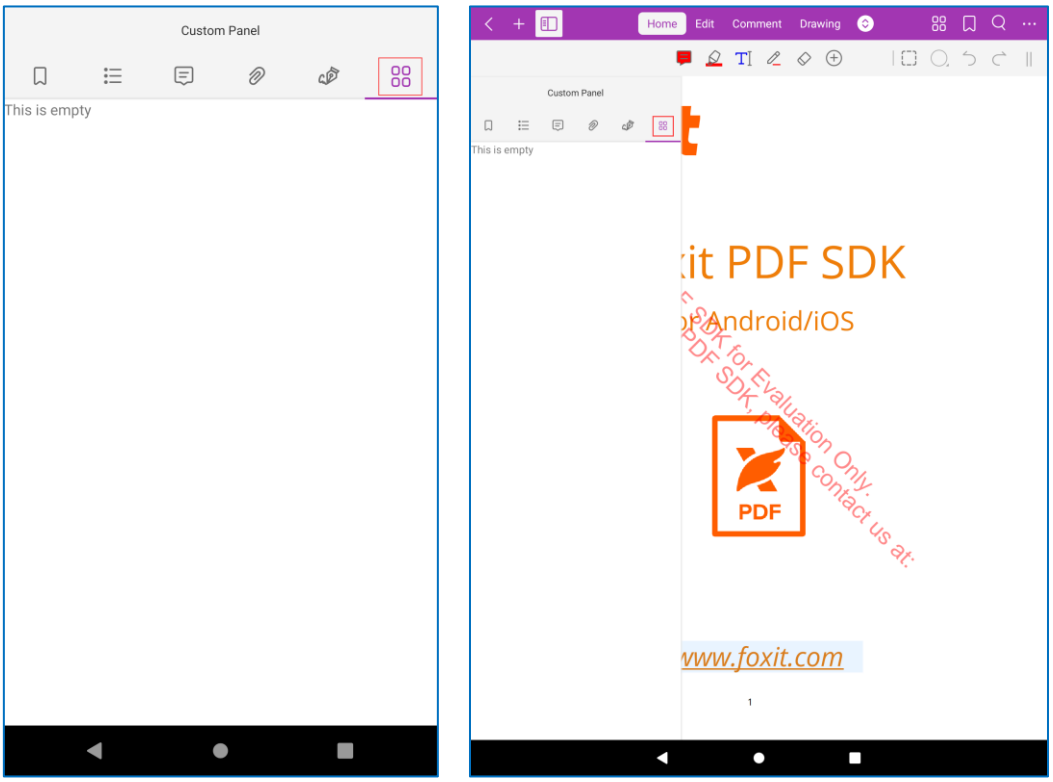


Figure 4-5

4.2.3 Customize to hide the UI elements in the View setting bar

To hide the UI elements in the View setting bar (See Figure 4-6, just taps  at the bottom toolbar (for phones) or tap  icon at the top toolbar to select the **View** (for tablets) to find it), you only need to use the following API:

In IViewSettingsWindow class,

```
public void setVisible (int type, boolean visible);
```

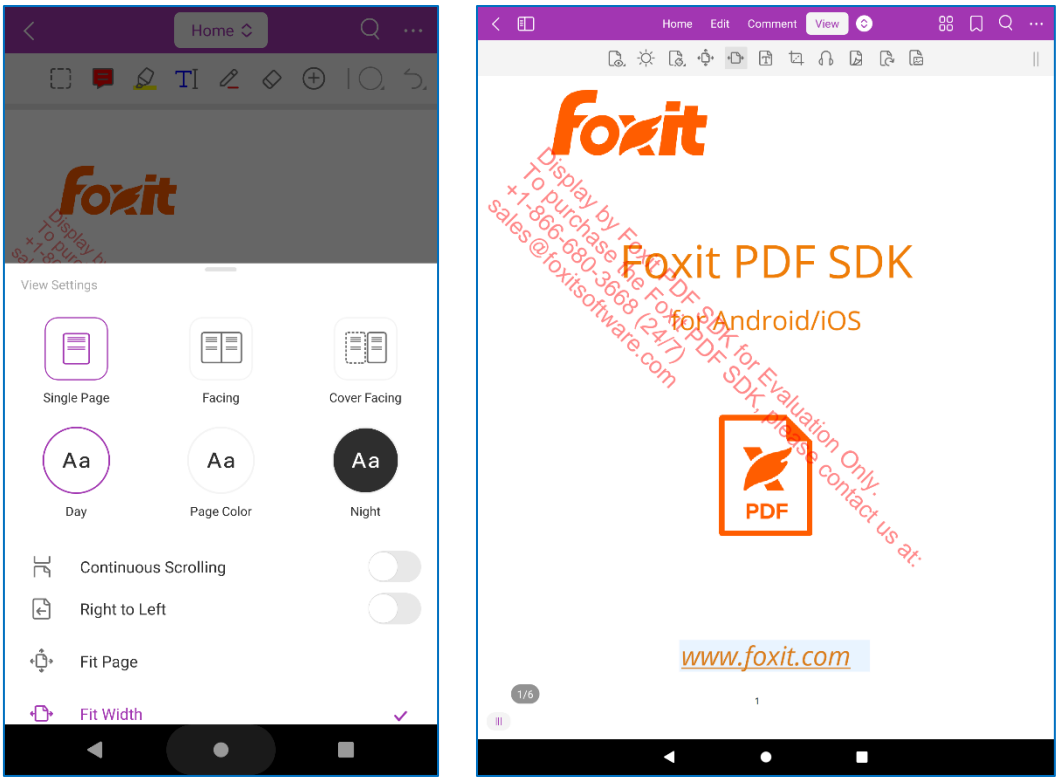


Figure 4-6

The value of the parameter "**type**" can be set as follows, which maps the items in the View setting bar.

type	integer
IViewSettingsWindow.TYPE_SINGLE_PAGE	1
IViewSettingsWindow.TYPE_FACING_PAGE	2
IViewSettingsWindow.TYPE_COVER_PAGE	4
IViewSettingsWindow.TYPE_DAY	8
IViewSettingsWindow.TYPE_PAGE_COLOR	16
IViewSettingsWindow.TYPE_NIGHT	32
IViewSettingsWindow.TYPE_CONTINUOUS_PAGE	64
IViewSettingsWindow.TYPE_RIGHT_TO_LEFT	608
IViewSettingsWindow.TYPE_FIT_PAGE	128
IViewSettingsWindow.TYPE_FIT_WIDTH	256
IViewSettingsWindow.TYPE_REFLOW	288
IViewSettingsWindow.TYPE_CROP	320
IViewSettingsWindow.TYPE_TTS	384
IViewSettingsWindow.TYPE_AUTO_FLIP	512
IViewSettingsWindow.TYPE_ROTATE_VIEW	544

IViewSettingsWindow.TYPE_PAN_ZOOM	576

In this section, we only take "**Fit Width**" item as an example to show you how to hide the UI elements in the View setting bar through APIs in the "**complete_pdf_viewer**" demo found in the "samples" folder. For other UI elements, you only need to change the "**type**".

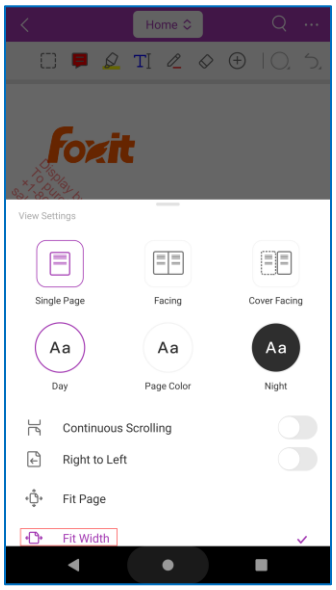
Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "**mUiExtensionsManager**.onCreate(getActivity(), **pdfViewerCtrl**, savedInstanceState);").

Example1: To hide the Fit Width item in the View setting bar. (For Phone and Tablet)

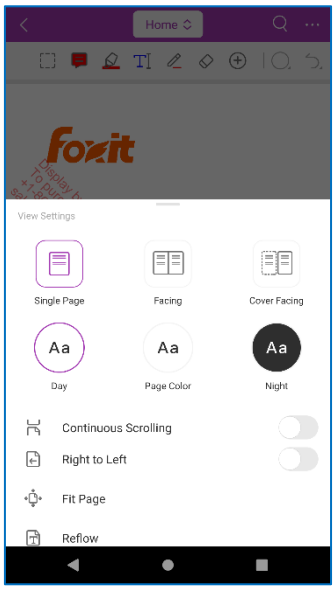
```
mUiExtensionsManager.getSettingWindow().setVisible(IViewSettingsWindow.TYPE_FIT_WIDTH, false);
```

for phone:

Before:



After:



for tablet:

Before: After:





4.2.4 Customize to add or hide the UI elements in the More Menu view

To show or hide the More Menu item, please see section 4.2.1 "Customizing top/bottom toolbar" (see example 10).

You can add a custom group or item, as well as hide a specific group or item through the APIs listed in the Table 4-3. Table 4-3 just lists the APIs that are used in the following examples, for more other APIs, please refer to the API Reference.

Clicks ••• at the right top toolbar, you can see the UI elements in the More Menu view as shown in Figure 4-7.

Table 4-3

IMenuGroup.**addItem**(IMenuItem item)	Add menu item with IMenuItem.
IMenuGroup.**addItem**(CharSequence title)	Add a new item to the group. This item displays the given title for its label.
IMenuGroup.**addItem**(Drawable icon, CharSequence title)	Add a new item to the group. This item displays the given icon and title for its label.
IMenuGroup.**setVisible**(boolean visible);	Change the visibility for the menu group.
IMenuItem.**setVisible**(boolean visible)	Set whether the item is visible.

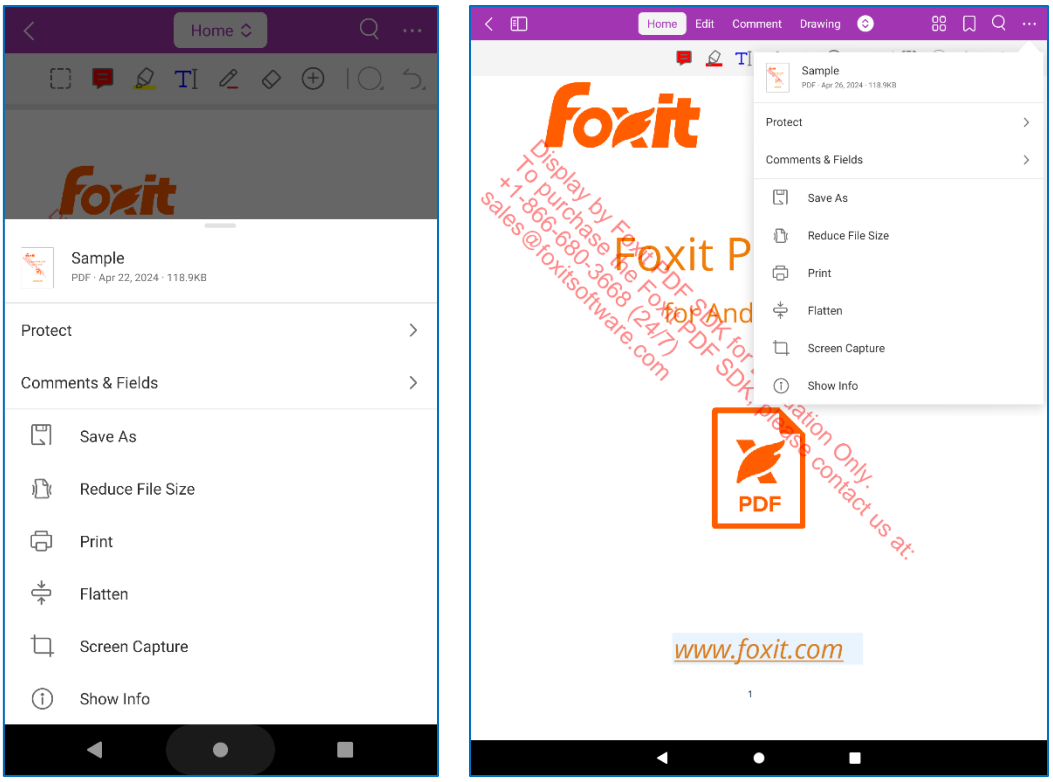


Figure 4-7

Complete PDF Viewer has two groups in the More Menu view as follows:

Group	integer
GROUP_ACTION_MENU_PRIMARY	1000
GROUP_ACTION_MENU_SECONDARY	1001

Each group has several items as follows:

Group	Item	integer
GROUP_ACTION_MENU_PRIMARY	ITEM_PRIMARY_PROTECT	1
	ITEM_PRIMARY_COMMENT_FIELDS	2
GROUP_ACTION_MENU_SECONDARY	ITEM_SECONDARY_SAVE_AS	1
	ITEM_SECONDARY_REDUCE_FILE_SIZE	2
	ITEM_SECONDARY_PRINT	3
	ITEM_SECONDARY_FLATTEN	4
	ITEM_SECONDARY_SCREEN	5

ITEM_PRIMARY_PROTECT and ITEM_PRIMARY_COMMENT_FIELDS have several sub-items as follows:

Item	sub-Item	integer
ITEM_PRIMARY_PROTECT	ITEM_PROTECT_REDACTION	1
	ITEM_PROTECT_REMOVE_PASSWORD	2
	ITEM_PROTECT_FILE_ENCRYPTION	3
	ITEM_PROTECT_TRUSTED_CERTIFICATES	4
ITEM_PRIMARY_COMMENT_FIELDS	ITEM_COMMENTS_FIELDS_IMPORT_COMMENTS	1
	ITEM_COMMENTS_FIELDS_EXPORT_COMMENTS	2
	ITEM_COMMENTS_FIELDS_SUMMARIZE_COMMENTS	3
	ITEM_COMMENTS_FIELDS_RESET_FORM_FIELDS	4
	ITEM_COMMENTS_FIELDS_IMPORT_FORM_DATA	5
	ITEM_COMMENTS_FIELDS_EXPORT_FORM_DATA	6

In this section, we only give three examples:

- **Example1** and **Example2** shows you how to hide a specific group or item in the More Menu view through APIs in the "**complete_pdf_viewer**" demo found in the "samples" folder. Just take the group GROUP_ACTION_MENU_PRIMARY and the item ITEM_SECONDARY_PRINT as examples, and for other groups and items, please refer to the examples.
- **Example3** shows you how to add a custom group with several items through APIs.

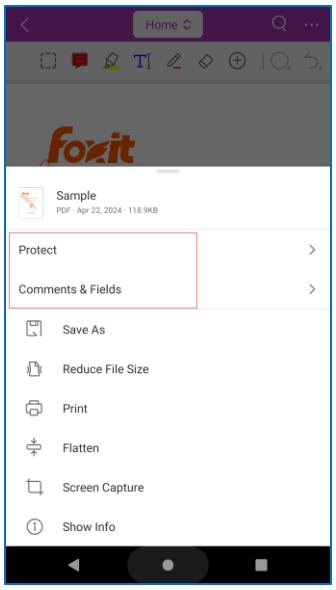
Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "**mUiExtensionsManager** = **new** UIExtensionsManager(getActivity().getApplicationContext(), **pdfViewerCtrl**, config);").

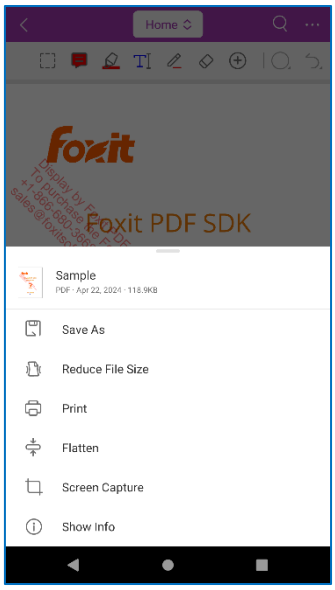
***Note**: The built-in UI is a bit different on tablets and phones. The following examples are applicable for phones and tablets, if the custom results on phones and tablets are similar, we only list the result on phones.*

Example1: Hide the group "GROUP_ACTION_MENU_PRIMARY" in the More Menu view. (For Phone and Tablet)

```
IMenuView  menuView  = mUiExtensionsManager.getMenuView();
IMenuGroup menuGroup = menuView.getGroup(MoreMenuConstants.GROUP_ACTION_MENU_PRIMARY);
menuGroup.setVisible(false);
```

Before: After:

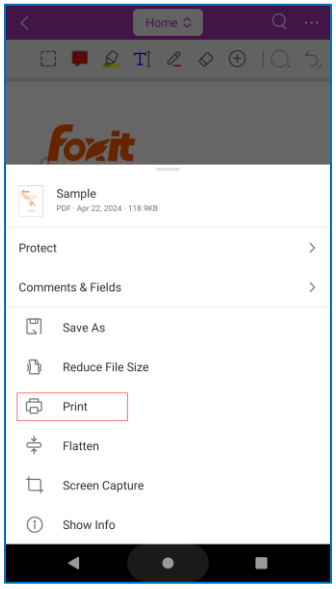




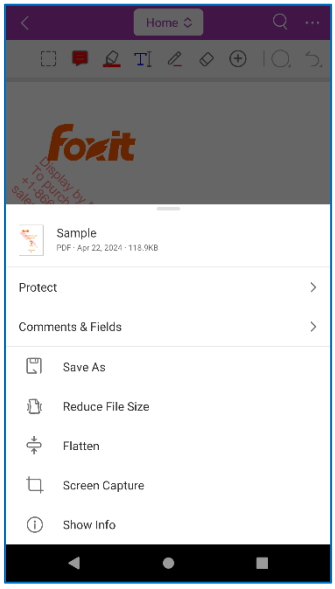
Example2: Hide the item "ITEM_SECONDARY_PRINT" in the More Menu view. (For Phone and Tablet)

```
IMenuView menuView  = mUiExtensionsManager.getMenuView();
IMenuGroup menuGroup = menuView.getGroup(MoreMenuConstants.GROUP_ACTION_MENU_SECONDARY);
IMenuItem menuItem = menuGroup.getItem(MoreMenuConstants.ITEM_SECONDARY_PRINT);
menuItem.setVisible(false);
```

Before:



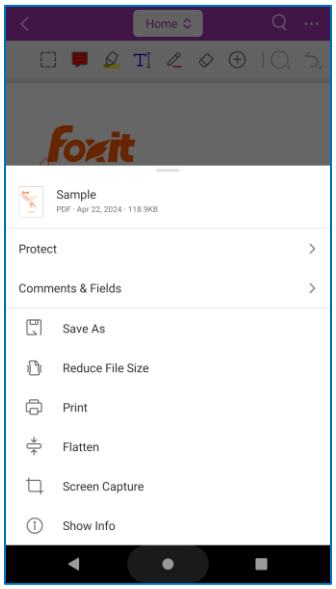
After:



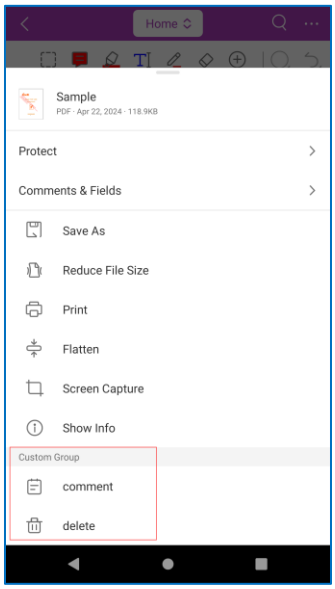
Example3: Add a custom group with two items in the More Menu view. (For Phone and Tablet)

```
IMenuView menuView = mUiExtensionsManager.getMenuView();
IMenuGroup customGroup = menuView.addGroup("Custom Group");
IMenuItem item1 =
customGroup.addItem(AppResource.getDrawable(getActivity(),R.drawable.rd_comment_menu), "comment");
item1.setOnMenuItemClickListener(new IMenuItem.OnMenuItemClickListener() {
    @Override
    public void onClick(IMenuItem item) {
        UIToast.getInstance(getActivity()).show("I am item1");
    }
});
IMenuItem item2 = customGroup.addItem(AppResource.getDrawable(getActivity(),R.drawable.rd_delete_menu),
"delete");
item2.setOnMenuItemClickListener(new IMenuItem.OnMenuItemClickListener() {
    @Override
    public void onClick(IMenuItem item) {
        UIToast.getInstance(getActivity()).show("I am item2");
    }
});
```

Before:



After:



4.3 Customize UI implementation through source code

In the previous sections, we have introduced how to customize the user interface through a configuration file or APIs in detail. Those changes are in the context of the built-in UI framework of Foxit PDF SDK for Android. If you do not want to use the ready-made UI framework, you can redesign it through modifying the source code of the UI Extensions Component.

To customize the UI implementation, you need to follow these steps:

***Note**: In this section, we only introduce how to customize the UI implementation in the UI Extensions Component. For the UI of scanning feature, you can take the tutorial in this section as reference.*

First, add the following files into your app. They are all found in the "libs" folder.

- ***uiextensions_src*** project - It is an open source library that contains some ready-to-use UI module implementations, which can help developers rapidly embed a fully functional PDF reader into their Android app. Of course, developers are not forced to use the default UI, they can freely customize and design the UI for their specific apps through the "uiextensions_src" project.
- ***FoxitRDK.aar*** - contains JAR package which includes all the Java APIs of Foxit PDF SDK for Android, as well as the underlying ".so" libraries. The ".so" library is the heart of the SDK including the core functionalities of Foxit PDF SDK for Android. It is built separately for each architecture, and currently available for armeabi-v7a, arm64-v8a, x86, and x86_64.

***Note:** The **uiextensions_src** project has a dependency on **FoxitRDK.aar**. It is best to put them in the same directory. If they are not in the same directory, you will need to modify the reference path in the "**build.gradle**" file of the **uiextensions_src** project manually.*

Second, find the specific layout XML files that you want to customize in the ***uiextensions_src*** project, then modify them based on your requirements.

Now, for your convenience, we will show you how to customize the UI implementation in the "**viewer_ctrl_demo**" project found in the "samples" folder.

UI Customization Example

Step 1: Add the ***uiextensions_src*** project into the demo making sure that it is in the same folder as the **FoxitRDK.aar** file**.** This folder should already be in the right location if you have not changed the default folder hierarchy.

Note *The demo already includes references to the* ***FoxitRDK.aar*** *file, so we just need to add the* ***uiextensions_src*** *project through configuring the "settings.gradle" file. When to include the* ***uiextensions_src*** *project as a dependency, the reference to the* ***FoxitRDKUIExtensions.aar*** *needs to be removed.*

Load the "**viewer_ctrl_demo**" in Android Studio. Then, follow the steps below:

a) In the "settings.gradle" file, add the following code to include the ***uiextensions_src*** *project.*

settings.gradle:

```
include ':app'
include ':uiextensions_src'
project(':uiextensions_src').projectDir = new File('../../libs/uiextensions_src/')
```

Rebuild the gradle, then the ***uiextensions_src*** project will be added as shown in Figure 4-8.

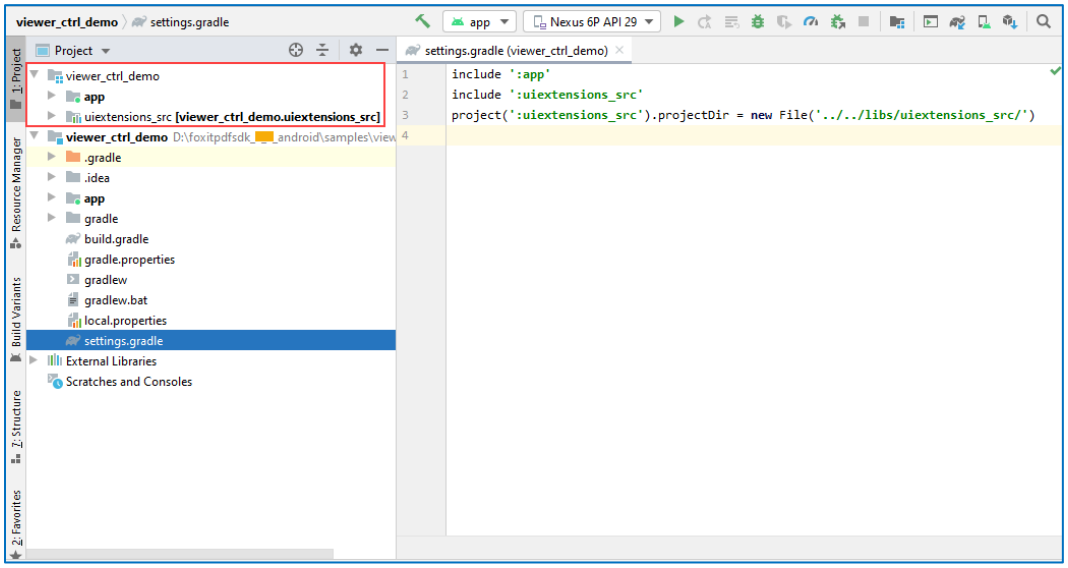


Figure 4-8

b) Include the ***uiextensions_src*** project as a dependency into the demo. Inside the app's "build.gradle" file, add the *implementation project(**":uiextensions_src"**)* line and comment out the *implementation(**name:'FoxitRDKUIExtensions', ext:'aar'**)* line as follows:

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.multidex:multidex:2.0.1'
    implementation (name: 'FoxitRDK', ext: 'aar')
   // implementation(name:'FoxitRDKUIExtensions', ext:'aar')
```

```
    implementation project(":uiextensions_src")
    implementation 'com.edmodo:cropper:1.0.1'
    implementation 'com.microsoft.identity.client:msal:2.2.0'
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
    implementation 'org.bouncycastle:bcpkix-jdk15on:1.60'
    implementation 'org.bouncycastle:bcprov-jdk15on:1.60'
    // RxJava
    implementation "io.reactivex.rxjava2:rxjava:2.2.16"
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'
}
```

After making the change, rebuild this gradle. Then, select "**File** -> **Project Structure...**" to open the **Project Structure** dialog. In the dialog click "**Dependencies** -> **app**", then you can see that the demo has a dependency on the ***uiextensions_src*** project as shown in Figure 4-9.

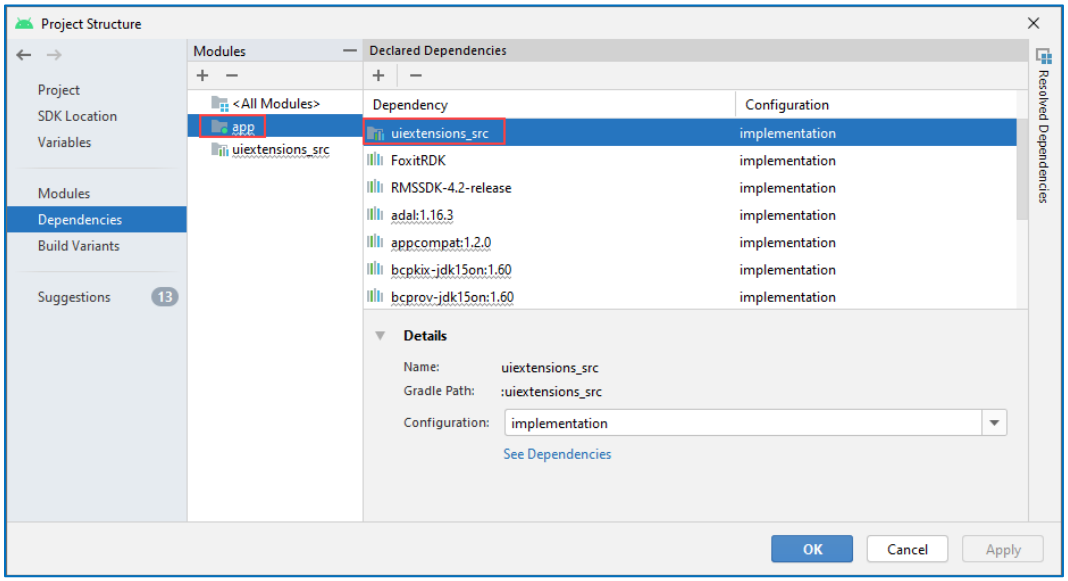


Figure 4-9

Congratulations! You have completed the first step.

Step 2: Find and modify the layout files related to the UI that you want to customize.

Now we will show you a simple example that changes one button's icon in the search panel as shown in Figure 4-10.

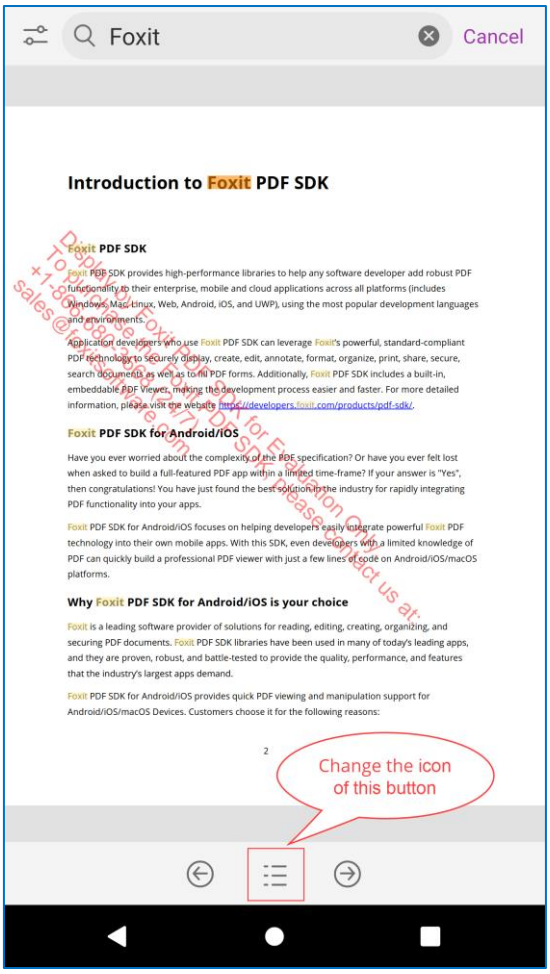


Figure 4-10

To replace the icon we only need to find the place which stores the icon for this button, then use another icon with the same name to replace it.

In the project, click "**uiextensions_src** -> **src** -> **main** -> **res** -> **layout**" as shown in Figure 4-11.

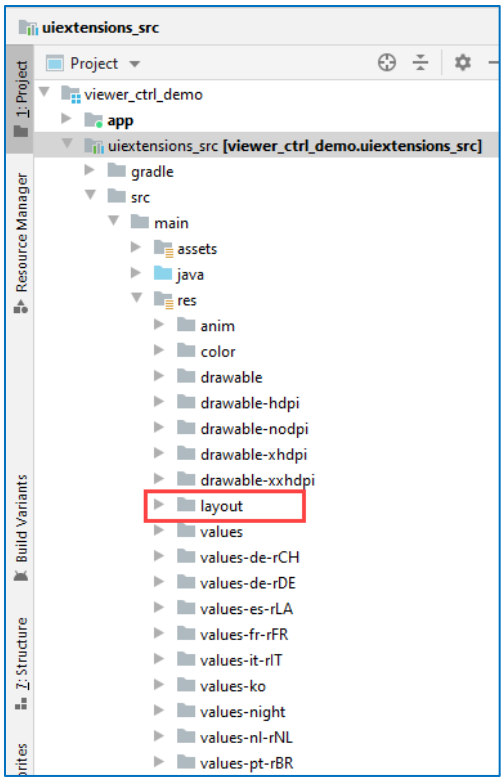


Figure 4-11

In the layout list, find the "**search_layout.xml**" file, and double-click it. Find the button in the Preview window, and click it to navigate to the related code as shown in Figure 4-12.

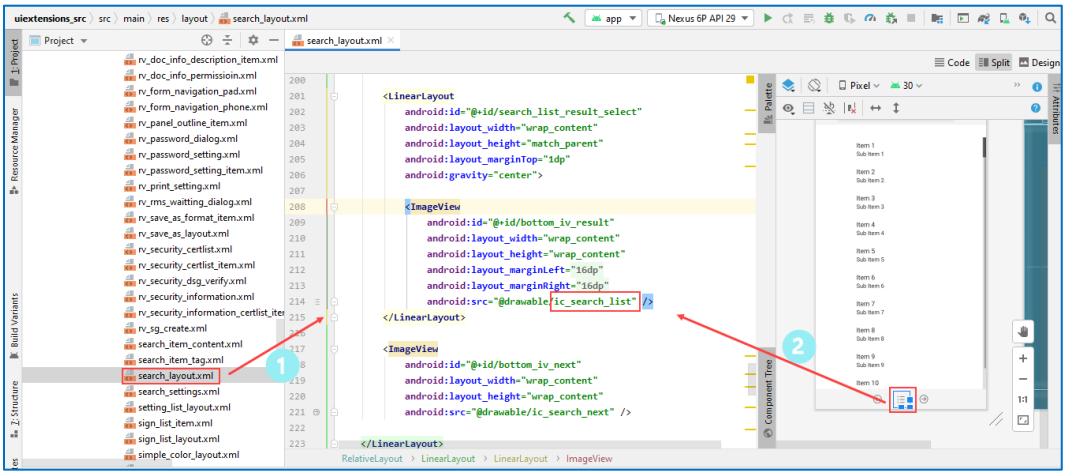


Figure 4-12

So, it can be seen that the icon is stored in the "drawable" folder with the name of "**ic_search_list**". Just replace it with your own icon.

For example, we use the icon of the search next button ("ic_search_next.xml" stored in the same folder with "ic_search_list.xml") to replace it. Then, rerun the demo, try the search feature and we can see that the icon of the bottom search button has changed as shown in Figure 4-13.

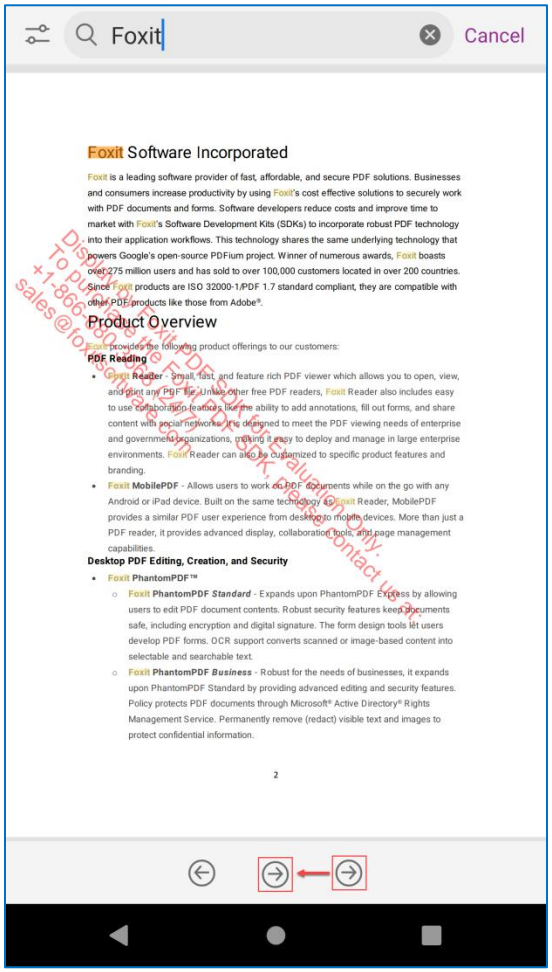


Figure 4-13

This is just a simple example to show how to customize the UI implementation. You can refer to it and feel free to customize and design the UI for your specific apps through the ***uiextensions_src*** project.

5 Working with SDK API

Foxit PDF SDK for Android wrapped all of the features implementations into the UI Extensions Component. If you are interested in the detailed process of the features implementations, please go through this section.

In this section, we will introduce a set of major features and list some examples to show you how to implement the features using Foxit PDF SDK Core API.

5.1 Render

PDF rendering is realized through the Foxit renderer, a graphic engine that is used to render page to a bitmap or a platform device context. Foxit PDF SDK provides APIs to set rendering options/flags, for example set flag to decide whether to render form fields and signature, whether to draw image anti-aliasing and path anti-aliasing. To do rendering, you can use the following APIs:

- To render page and annotations, first use function Renderer.setRenderContentFlags to decide whether to render page and annotation both or not, and then use function Renderer.startRender to do the rendering. Function Renderer.startQuickRender can also be used to render page but only for thumbnail purpose.
- To render a single annotation, use function Renderer.renderAnnot.
- To render on a bitmap, use function Renderer.startRenderBitmap.
- To render a reflowed page, use function Renderer.startRenderReflowPage.

Widget annotation is always associated with form field and form control in Foxit PDF SDK. For how to render widget annotations, here is a recommended flow:

- After loading a PDF page, first render the page and all annotations in this page (including widget annotations).
- Then, if use pdf.interform.Filler object to fill the form, the function pdf.interform.Filler.render should be used to render the focused form control instead of the function Renderer.renderAnnot.

Example:

5.1.1 How to render a specified page to a bitmap

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Progressive;
import com.foxit.sdk.common.Renderer;
import com.foxit.sdk.common.fxcrt.Matrix2D;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
...

public Bitmap renderPageToBitmap(PDFPage pdfPage, int drawPageWidth, int drawPageHeight) {
   try {
      // If the page hasn't been parsed yet, throw an exception.
      if (!pdfPage.isParsed()) {
         throw new PDFException(Constants.e_ErrNotParsed, "PDF Page should be parsed first");
      }

      // Pepare matrix to render on the bitmap.
      Matrix2D matrix2D = pdfPage.getDisplayMatrix(0, 0, drawPageWidth, drawPageHeight,
Constants.e_Rotation0);
      // Create a bitmap according to the required drawPageWidth and drawPageHeight.
      Bitmap bitmap = Bitmap.createBitmap(drawPageWidth, drawPageHeight, Bitmap.Config.RGB_565);
      // Fill the bitmap with white color.
      bitmap.eraseColor(Color.WHITE);
      Renderer renderer = new Renderer(bitmap, true);
      // Set the render flag, both page content and annotation will be rendered.
      renderer.setRenderContentFlags(Renderer.e_RenderPage | Renderer.e_RenderAnnot);
      // Start to render the page progressively.
      Progressive progressive = renderer.startRender(pdfPage, matrix2D, null);
      int state = Progressive.e_ToBeContinued;
      while (state == Progressive.e_ToBeContinued) {
         state = progressive.resume();
      }

      if (state != Progressive.e_Finished) return null;
      return bitmap;
   } catch (PDFException e) {
```

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

5.2 Text Page

Foxit PDF SDK provides APIs to extract, select, search and retrieve text in PDF documents. PDF text contents are stored in TextPage objects which are related to a specific page. TextPage class can be used to retrieve information about text in a PDF page, such as single character, single word, text content within specified character range or rectangle and so on. It also can be used to construct objects of other text related classes to do more operations for text contents or access specified information from text contents:

- To search text in text contents of a PDF page, construct a TextSearch object with TextPage object.
- To access text such like hypertext link, construct a PageTextLinks object with TextPage object.
- To highlight the selected text on the PDF page, construct a TextPage object for calculating text area by selection.

Example:

5.2.1 How to get the text area on a page by selection

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.fxcrt.RectF;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.TextPage;
import com.foxit.sdk.common.fxcrt.PointF;
...

// Get the text area on page by selection. The starting selection position and ending selection position are specified by startPos and endPos.
public ArrayList<RectF> getTextRectsBySelection(PDFPage page, PointF startPos, PointF endPos) {
    try {
        // If the page hasn't been parsed yet, throw an exception.
        if (!page.isParsed()) {
```

```
            throw new PDFException(Constants.e_ErrNotParsed, "PDF Page should be parsed first");
        }

        // Create a text page from the parsed PDF page.
        TextPage textPage = new TextPage(page, TextPage.e_ParseTextNormal);
        if (textPage == null || textPage.isEmpty()) return null;
        int startCharIndex = textPage.getIndexAtPos(startPos.getX(), startPos.getY(), 5);
        int endCharIndex = textPage.getIndexAtPos(endPos.getX(), endPos.getY(), 5);
        // API getTextRectCount requires that start character index must be lower than or equal to end character index.
        startCharIndex = startCharIndex < endCharIndex ? startCharIndex : endCharIndex;
        endCharIndex = endCharIndex > startCharIndex ? endCharIndex : startCharIndex;
        int count = textPage.getTextRectCount(startCharIndex, endCharIndex - startCharIndex);

        if (count > 0) {
            ArrayList<RectF> array = new ArrayList<RectF>();
            for (int i = 0; i < count; i ++) {
                RectF rectF = textPage.getTextRect(i);
                if (rectF == null || rectF.isEmpty()) continue;
                array.add(rectF);
            }
            // The return rects are in PDF unit, if caller need to highlight the text rects on the screen, then these rects should be converted in device unit first.
            return array;
        }
    } catch (PDFException e) {
        e.printStackTrace();
    }
    return null;
}
...
```

5.3 Text Search

Foxit PDF SDK provides APIs to search text in a PDF document, a XFA document, a text page or in a PDF annotation's appearance. It offers functions to do a text search and get the searching result:

- To specify the searching pattern and options, use functions TextSearch.setPattern, TextSearch.setStartPage (only useful for a text search in PDF document), TextSearch.setEndPage (only useful for a text search in PDF document) and TextSearch.setSearchFlags.

- To do the searching, use function TextSearch.findNext or TextSearch.findPrev.
- To get the searching result, use function TextSearch.getMatchXXX().

Example:

5.3.1 How to search a text pattern in a PDF

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.fxcrt.RectF;
import com.foxit.sdk.common.fxcrt.RectFArray;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.TextPage;
import com.foxit.sdk.common.fxcrt.PointF;
import com.foxit.sdk.pdf.TextSearch;
...

try {
   String pdfpath = "XXX/Sample.pdf";
   PDFDoc doc = new PDFDoc(pdfpath);
   doc.load(null);

   // Create a text search handler for searching in PDF document.
   TextSearch textSearch = new TextSearch(doc, null, TextPage.e_ParseTextNormal);
   // Set the start page index which searching will begin. By default, end page will be the last page.
   textSearch.setStartPage(0);
   // Set the text to be searched.
   textSearch.setPattern("foxit");
   // Set the search flags to be matching case and matching whole word.
   textSearch.setSearchFlags(TextSearch.e_SearchMatchCase|TextSearch.e_SearchMatchWholeWord);
   while (textSearch.findNext()) {
      // If true, then we found a matched result.
      // Get the found page index.
      int pageIndx = textSearch.getMatchPageIndex();
      // Get the start character index of the matched text on the found page.
      int startCharIndex = textSearch.getMatchStartCharIndex();
      // Get the end character index of the matched text on the found page.
      int endCharIndex = textSearch.getMatchEndCharIndex();
      // Get the rectangular region of the matched text on the found page.
      RectFArray matchRects = textSearch.getMatchRects();
   }
```

```
}
catch (Exception e) {}
...
```

5.4 Bookmark (Outline)

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. It is a tree-structured hierarchy, so function PDFDoc.getRootBookmark must be called first to get the root of the whole bookmark tree before accessing to the bookmark tree. Here, "root bookmark" is an abstract object which can only have some child bookmarks without next sibling bookmarks and any data (includes bookmark data, destination data and action data). It cannot be shown on the application UI since it has no data. Therefore, a root bookmark can only call function Bookmark.getFirstChild.

After the root bookmark is retrieved, following functions can be called to access other bookmarks:

- To access the parent bookmark, use function Bookmark.getParent.
- To access the first child bookmark, use function Bookmark.getFirstChild.
- To access the next sibling bookmark, use function Bookmark.getNextSibling.
- To insert a new bookmark, use function Bookmark.insert.
- To move a bookmark, use function Bookmark.moveTo.

Example:

5.4.1 How to travel the bookmarks of a PDF in depth first order

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.Bookmark;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.actions.Destination;
...
private void DepthFistTravelBookmarkTree(Bookmark bookmark, PDFDoc doc) {
  if(bookmark == null || bookmark.isEmpty())
    return;
  try {
    DepthFistTravelBookmarkTree(bookmark.getFirstChild(), doc);
    while (true) {
```

```
// Get bookmark title.
String title = bookmark.getTitle();
Destination dest = bookmark.getDestination();
if (dest != null && !dest.isEmpty())
{
   float left, right, top, bottom;
   float zoom;
   int pageIndex = dest.getPageIndex(doc);
   // left,right,top,bottom,zoom are only meaningful with some special zoom modes.
   int mode = dest.getZoomMode();
   switch (mode) {
      case Destination.e_ZoomXYZ:
         left = dest.getLeft();
         top = dest.getTop();
         zoom = dest.getZoomFactor();
         break;
      case Destination.e_ZoomFitPage:
         break;
      case Destination.e_ZoomFitHorz:
         top = dest.getTop();
         break;
      case Destination.e_ZoomFitVert:
         left = dest.getLeft();
         break;
      case Destination.e_ZoomFitRect:
         left = dest.getLeft();
         bottom = dest.getBottom();
         right = dest.getRight();
         top = dest.getTop();
         break;
      case Destination.e_ZoomFitBBox:
         break;
      case Destination.e_ZoomFitBHorz:
         top = dest.getTop();
         break;
      case Destination.e_ZoomFitBVert:
         left = dest.getLeft();
         break;
      default:
```

```
                break;
            }
          }
          bookmark = bookmark.getNextSibling();
          if (bookmark == null || bookmark.isEmpty())
            break;
          DepthFistTravelBookmarkTree(bookmark.getFirstChild(), doc);
        }
      }
      catch (Exception e) {
      }
}
```

5.5 Reading Bookmark

Reading bookmark is not a PDF bookmark, in other words, it is not PDF outlines. It is the bookmark in applicable level. It is stored in the metadata (XML format) of catalog. It allows user to add or remove a reading bookmark according to their reading preferences and navigate to one PDF page easily by selecting one reading bookmark.

In order to retrieve the reading bookmark, function PDFDoc.getReadingBookmarkCount could be called to count the reading bookmarks, and function PDFDoc.getReadingBookmark could be called to get a reading bookmark by index.

This class offers several functions to get/set properties of reading bookmarks, such as title, destination page index and creation/modified date time.

5.5.1 How to add a custom reading bookmark and enumerate all the reading bookmarks

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.DateTime;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.ReadingBookmark;
...
// Add a new reading bookmark to pdf document, the returned bookmark stores the title and the page index.
ReadingBookmark addReadingBookmark(PDFDoc pdfDoc, String title, int pageIndex) {
  int count = pdfDoc.getReadingBookmarkCount();
```

```
    return pdfDoc.insertReadingBookmark(count,title,pageIndex);
}

// Enumerate all the reading bookmarks from the pdf document.
void getReadingBookmark(PDFDoc pdfDoc) {
    try {
        int count = pdfDoc.getReadingBookmarkCount();
        for (int i = 0; i < count; i ++) {
            ReadingBookmark readingBookmark = pdfDoc.getReadingBookmark(i);
            if(readingBookmark.isEmpty()) continue;
            // Get bookmark title.
            String title = readingBookmark.getTitle();
            // Get the page index which associated with the bookmark.
            int pageIndex = readingBookmark.getPageIndex();
            // Get the creation date of the bookmark.
            DateTime creationTime = readingBookmark.getDateTime(true);
            // Get the modification date of the bookmark.
            DateTime modificationTime = readingBookmark.getDateTime(false);
        }
    } catch (PDFException e) {
        e.printStackTrace();
    }
}
```

5.6 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. Foxit PDF SDK provides applications APIs to access attachments such as loading attachments, getting attachments, inserting/removing attachments, and accessing properties of attachments.

Example:

5.6.1 How to embed a specified file to a PDF document

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.Attachments;
```

```
import com.foxit.sdk.pdf.FileSpec;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.objects.PDFNameTree;
...
try {
    String pdfpath = "XXX/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
    doc.load(null);

    // Embed the specified file to PDF document.
    String filePath = "/xxx/fileToBeEmbedded.xxx";
    PDFNameTree nameTree = new PDFNameTree(doc, PDFNameTree.e_EmbeddedFiles);
    Attachments attachments = new Attachments(doc, nameTree);
    FileSpec fileSpec = new FileSpec(doc);
    fileSpec.setFileName(filePath);
    if (!fileSpec.embed(filePath)) return;
    attachments.addEmbeddedFile(filePath, fileSpec);
} catch (PDFException e) {
    e.printStackTrace();
}

...
```

5.6.2 How to export the embedded attachment file from a PDF and save it as a single file

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.Attachments;
import com.foxit.sdk.pdf.FileSpec;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.objects.PDFNameTree;
...
try {
    String pdfpath = "XXX/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
    doc.load(null);

    PDFNameTree nameTree = new PDFNameTree(doc, PDFNameTree.e_EmbeddedFiles);
    Attachments attachments = new Attachments(doc, nameTree);
    // Extract the embedded attachment file.
    int count = attachments.getCount();
```

```
    for (int i = 0; i < count; i ++) {
        String key = attachments.getKey(i);
        if (key != null) {
            FileSpec fileSpec1 = attachments.getEmbeddedFile(key);
            String exportedFile = "/somewhere/" + fileSpec1.getFileName();
            if (fileSpec1 != null && !fileSpec1.isEmpty()) {
                fileSpec1.exportToFile(exportedFile);
            }
        }
    }
} catch (PDFException e) {
    e.printStackTrace();
}
...
```

5.7 Annotation

An annotation associates an object such as note, line, and highlight with a location on a page of a PDF document. PDF includes a wide variety of standard annotation types as listed in Table 5-1. Among these annotation types, many of them are defined as markup annotations for they are used primarily to mark up PDF documents. The 'Markup' column in Table 5-1 shows whether an annotation is a markup annotation.

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

Table 5-1

Annotation type	Description	Markup	Supported by SDK
Text(Note)	Text annotation	Yes	Yes
Link	Link Annotation	No	Yes
FreeText (TypeWriter/TextBox/Callout)	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

Annotation type	Description	Markup	Supported by SDK
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	No	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	No	Yes
PrinterMark	PrinterMark annotation	No	No
TrapNet	Trap network annotation	No	No
Watermark*	Watermark annotation	No	No
3D	3D annotation	No	No
Redact	Redact annotation	Yes	Yes

Note: Foxit SDK supports a customized annotation type called PSI (pressure sensitive ink) annotation that is not described in PDF Reference. 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:

5.7.1 How to add annotations to a PDF page

```
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.fxcrt.RectF;
import com.foxit.sdk.common.fxcrt.RectFArray;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.TextSearch;
import com.foxit.sdk.pdf.annots.Annot;
import com.foxit.sdk.pdf.annots.Note;
import com.foxit.sdk.pdf.annots.QuadPoints;
import com.foxit.sdk.pdf.annots.QuadPointsArray;
import com.foxit.sdk.pdf.annots.TextMarkup;
import com.foxit.sdk.common.fxcrt.PointF;
```

```
...
// Add text annot.
try {
    String pdfpath = "xxx/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
    doc.load(null);
    PDFPage pdfPage = doc.getPage(1);
    RectF rect = new RectF(100, 100, 120, 120);
    Note note = new Note(pdfPage.addAnnot(Annot.e_Note, rect));
    if (note == null || note.isEmpty()){
        return;
    }
    note.setIconName("Comment");
    // Set color to blue.
    note.setBorderColor(0xff0000ff);
    note.setContent("This is the note comment, write any content here.");
    note.resetAppearanceStream();

    // The following code demonstrates how to add hightlight annotation on the searched text.
    TextSearch textSearch = new TextSearch(pdfPage.getDocument(), null, TextPage.e_ParseTextNormal);
    if (textSearch == null || textSearch.isEmpty()){
        return;
    }
    // Suppose that the text for highlighting is "foxit".
    textSearch.setPattern("foxit");
    boolean bMatched = textSearch.findNext();
    if (bMatched) {
        RectFArray rects = textSearch.getMatchRects();
        int rectCount = rects.getSize();
        // Fill the quadpoints array according to the text rects of matched result.
        QuadPointsArray arrayOfQuadPoints = new QuadPointsArray();
        for (int i = 0; i < rectCount; i++) {
            rect = rects.getAt(i);
            QuadPoints quadPoints = new QuadPoints();
            PointF point = new PointF();
            point.set(rect.getLeft(), rect.getTop());
            quadPoints.setFirst(point);
            point.set(rect.getRight(), rect.getTop());
            quadPoints.setSecond(point);
```

```
            point.set(rect.getLeft(), rect.getBottom());
            quadPoints.setThird(point);
            point.set(rect.getRight(), rect.getBottom());
            quadPoints.setFourth(point);
            arrayOfQuadPoints.add(quadPoints);
        }
        // Just set an empty rect to markup annotation, the annotation rect will be calculated according to the quadpoints that set to it later.
        rect = new RectF(0, 0, 0, 0);
        TextMarkup textMarkup = new TextMarkup(pdfPage.addAnnot(Annot.e_Highlight, rect));
        // Set the quadpoints to this markup annot.
        textMarkup.setQuadPoints(arrayOfQuadPoints);
        // set to red.
        textMarkup.setBorderColor(0xffff0000);
        // set to thirty-percent opacity.
        textMarkup.setOpacity(0.3f);
        // Generate the appearance.
        textMarkup.resetAppearanceStream();
    }
} catch (Exception e) {}
```

5.7.2 How to delete annotations in a PDF page

```
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.annots.Annot;
...

try {
    String pdfpath = "xxx/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
     doc.load(null);
    PDFPage pdfPage = doc.getPage(1);
    Annot annot = pdfPage.getAnnot(0);
    if (annot == null || annot.isEmpty())
        return;
    // Remove the first annot, so the second annot will become first.
```

```
    pdfPage.removeAnnot(annot);
} catch (Exception e) {}
```

5.7.3 How to register listeners to receive annotation events

Annotation event listeners should always be registered in advance before receiving the annotation events. See the following code snippet.

```
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.annots.Annot;
import com.foxit.uiextensions.UIExtensionsManager;
import com.foxit.uiextensions.annots.AnnotEventListener;

public class RegisterListener {
    AnnotEventListener mAnnotEventListener = new AnnotEventListener() {
        @Override
        public void onAnnotAdded(PDFPage page, Annot annot) {
        }

        @Override
        public void onAnnotWillDelete(PDFPage page, Annot annot) {
        }

        @Override
        public void onAnnotDeleted(PDFPage page, Annot annot) {
        }

        @Override
        public void onAnnotModified(PDFPage page, Annot annot) {
        }

        @Override
        public void onAnnotChanged(Annot lastAnnot, Annot currentAnnot) {
        }
    };

    void registerYourAnnotEventListener(UIExtensionsManager extensionsManager) {
        // Call registerAnnotEventListener to register the annot event listener to receive annot events
        extensionsManager.getDocumentManager().registerAnnotEventListener(mAnnotEventListe
ner);
    }

}
```

5.8 Form

Form (AcroForm) is a collection of fields for gathering information interactively from the user. Foxit PDF SDK provides APIs to view and edit form field programmatically. Form fields are commonly used in PDF documents to gather data. The Form class offers functions to retrieve form fields or form controls, import/export form data and other features, for example:

- To retrieve form fields, please use functions Form.getFieldCount and Form.getField.
- To retrieve form controls from a PDF page, please use functions Form.getControlCount and Form.getControl.
- To import form data from an XML file, please use function Form.importFromXML; to export form data to an XML file, please use function Form.exportToXML.
- To retrieve form filler object, please use function Form.getFormFiller.

To import form data from a FDF/XFDF file or export such data to a FDF/XFDF file, please refer to functions PDFDoc.importFromFDF and PDFDoc.exportToFDF.

Example:

5.8.1 How to import and export form data from or to a XML file

```
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.interform.Form;
...

// Check if the document has a form.
try {
   String pdfpath = "xxx/Sample.pdf";
   PDFDoc doc = new PDFDoc(pdfpath);
   doc.load(null);
   boolean hasForm = doc.hasForm();
   if(hasForm) {
      // Create a form object from document.
      Form form = new Form(doc);
      // Export the form data to a XML file.
      form.exportToXML("/somewhere/export.xml");
      // Or import the form data from a XML file.
      form.importFromXML("/somewhere/export.xml");
```

```
    }
}catch (Exception e) {}
```

5.9 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:

5.9.1 How to encrypt a PDF file with password

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.SecurityHandler;
import com.foxit.sdk.pdf.StdEncryptData;
import com.foxit.sdk.pdf.StdSecurityHandler;
...

// Encrypt the source pdf document with specified owner password and user password, the encrypted PDF will be saved to the path specified by parameter savePath.
public boolean encryPDF(PDFDoc pdfDoc, byte[] ownerPassword, byte[] userPassword, String savePth) {
    if (pdfDoc == null || (ownerPassword == null && userPassword == null) || savePth == null)
        return false;
    // The encryption setting data. Whether to encrypt meta data:true, User permission: modify,assemble,fill form. Cipher algorithm:AES 128.
    StdEncryptData encryptData = new StdEncryptData(true,
            PDFDoc.e_PermModify | PDFDoc.e_PermAssemble | PDFDoc.e_PermFillForm,
            SecurityHandler.e_CipherAES, 16);

    StdSecurityHandler securityHandler = new StdSecurityHandler();
    try {
        if (!securityHandler.initialize(encryptData, userPassword, ownerPassword)) return false;
        pdfDoc.setSecurityHandler(securityHandler);
        if (!pdfDoc.saveAs(savePth, PDFDoc.e_SaveFlagNormal)) return false;
        return true;
    } catch (PDFException e) {
```

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

5.10 Signature

PDF Signature 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.

***Note**: Foxit PDF SDK provides default Signature callbacks which supports the following two types of signature filter and subfilter:*

(1) filter: Adobe.PPKLite subfilter: adbe.pkcs7.detached

(2) filter: Adobe.PPKLite subfilter: adbe.pkcs7.sha1

If you use one of the above signature filter and subfilter, you can sign a PDF document and verify the validity of signature by default without needing to register a custom callback.

Example:

5.10.1 How to sign a PDF document and verify the signature

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Progressive;
import com.foxit.sdk.common.fxcrt.RectF;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.Signature;
...

// Sample code demonstrate signing and verifying of PDF signature.
public void addNewSignatureAndSign(PDFPage page, RectF rect) {
    try {
```

```
    // Add a new signature on the specified page rect.
    Signature signature = page.addSignature(rect);
    // Set the appearance flags, if the specified flag is on, then the associated key will be displayed on the signature appearance.
    signature.setAppearanceFlags(Signature.e_APFlagLabel | Signature.e_APFlagDN | Signature.e_APFlagText
            | Signature.e_APFlagLocation | Signature.e_APFlagReason | Signature.e_APFlagSigner);
    // Set signer.
    signature.setKeyValue(Signature.e_KeyNameSigner, "Foxit");
    // Set location.
    signature.setKeyValue(Signature.e_KeyNameLocation, "AnyWhere");
    // Set reason.
    signature.setKeyValue(Signature.e_KeyNameReason, "ANyReason");
    // Set contact info.
    signature.setKeyValue(Signature.e_KeyNameContactInfo, "AnyInfo");
    // Set domain name.
    signature.setKeyValue(Signature.e_KeyNameDN, "AnyDN");
    // Set description.
    signature.setKeyValue(Signature.e_KeyNameText, "AnyContent");
    // Filter "Adobe.PPKLite" is supported by default.
    signature.setFilter("Adobe.PPKLite");
    // SubFilter "adbe.pkcs7.sha1" or "adbe.pkcs7.detached" are supported by default.
    signature.setSubFilter("adbe.pkcs7.detached");

    // The input PKCS#12 format certificate, which contains the public and private keys.
    String certPath = "/somewhere/cert.pfx";
    // Password for that certificate.
    byte[] certPassword = "123".getBytes();
    String signedPDFPath = "/somewhere/signed.pdf";
    // Start to sign the signature, if everything goes well, the signed PDF will be saved to the path specified by "save_path".
    Progressive progressive = signature.startSign(certPath, certPassword, Signature.e_DigestSHA1, signedPDFPath, null, null);
    if (progressive != null) {
      int state = Progressive.e_ToBeContinued;
      while (state == Progressive.e_ToBeContinued) {
        state = progressive.resume();
      }
      if (state != Progressive.e_Finished) return;
    }
```

```
    // Get the signatures from the signed PDF document, then verify them all.
    PDFDoc pdfDoc = new PDFDoc(signedPDFPath);
    int err = pdfDoc.load(null);
    if (err != Constants.e_ErrSuccess) return;
    int count = pdfDoc.getSignatureCount();
    for (int i = 0; i < count; i++) {
      Signature sign = pdfDoc.getSignature(i);
      if (sign != null && !sign.isEmpty()) {
        Progressive progressive_1 = sign.startVerify(null, null);
        if (progressive_1 != null) {
          int state = Progressive.e_ToBeContinued;
          while (state == Progressive.e_ToBeContinued) {
            state = progressive_1.resume();
          }
          if (state != Progressive.e_Finished) continue;
        }
        int verifiedState = sign.getState();
        if ((verifiedState & sign.e_StateVerifyValid) == sign.e_StateVerifyValid) {
          Log.d("Signature", "addNewSignatureAndSign: Signature" + i + "is valid.");
        }
      }
    }
  } catch (PDFException e) {
    e.printStackTrace();
  }
}
```

5.10.2 How to set customized time information for signature

The function setSignTime currently doesn't allow changing the data format. But we can resolve it by passing the date string to the signature dictionary. See the following code snippet.

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Progressive;
import com.foxit.sdk.common.fxcrt.RectF;
import com.foxit.sdk.pdf.PDFDoc;
```

```
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.Signature;
...

// Sample code demonstrate signing and verifying of PDF signature.
public void addNewSignatureAndSign(PDFPage page, RectF rect) {
   try {
      // Add a new signature on the specified page rect.
      Signature signature = page.addSignature(rect);
      // Set the appearance flags, if the specified flag is on, then the associated key will be displayed on the signature appearance.
      signature.setAppearanceFlags(Signature.e_APFlagLabel | Signature.e_APFlagDN | Signature.e_APFlagText
            | Signature.e_APFlagLocation | Signature.e_APFlagReason | Signature.e_APFlagSigner
| Signature.e_APFlagSigningTime);
      // Set signer.
      signature.setKeyValue(Signature.e_KeyNameSigner, "Foxit");
      // Set location.
      signature.setKeyValue(Signature.e_KeyNameLocation, "AnyWhere");
      // Set reason.
      signature.setKeyValue(Signature.e_KeyNameReason, "ANyReason");
      // Set contact info.
      signature.setKeyValue(Signature.e_KeyNameContactInfo, "AnyInfo");
      // Set domain name.
      signature.setKeyValue(Signature.e_KeyNameDN, "AnyDN");
      // Set description.
      signature.setKeyValue(Signature.e_KeyNameText, "AnyContent");
        //DateTime dateTime = new DateTime();
        // ...
        //signature.setSignTime(dateTime);
      // The default format of the Signature date is yyMMddhhmmss-TimeZone.
      //Please refer to the following codes if you need to set the time of the custom format.
      PDFDictionary dictionary = signature.getSignatureDict();
      dictionary.setAtString("M", "2022/02/13 11:00:00"/* formatted time string*/);
      // Filter "Adobe.PPKLite" is supported by default.
      signature.setFilter("Adobe.PPKLite");
      // SubFilter "adbe.pkcs7.sha1" or "adbe.pkcs7.detached" are supported by default.
      signature.setSubFilter("adbe.pkcs7.detached");

      // The input PKCS#12 format certificate, which contains the public and private keys.
```

```
        String certPath = "/somewhere/cert.pfx";
        // Password for that certificate.
        byte[] certPassword = "123".getBytes();
        String signedPDFPath = "/somewhere/signed.pdf";
        // Start to sign the signature, if everything goes well, the signed PDF will be saved to the path specified by
"save_path".
        Progressive progressive = signature.startSign(certPath, certPassword, Signature.e_DigestSHA1,
signedPDFPath, null, null);
        if (progressive != null) {
            int state = Progressive.e_ToBeContinued;
            while (state == Progressive.e_ToBeContinued) {
                state = progressive.resume();
            }
            if (state != Progressive.e_Finished) return;
        }

        // Get the signatures from the signed PDF document, then verify them all.
        PDFDoc pdfDoc = new PDFDoc(signedPDFPath);
        int err = pdfDoc.load(null);
        if (err != Constants.e_ErrSuccess) return;
        int count = pdfDoc.getSignatureCount();
        for (int i = 0; i < count; i++) {
            Signature sign = pdfDoc.getSignature(i);
            if (sign != null && !sign.isEmpty()) {
                Progressive progressive_1 = sign.startVerify(null, null);
                if (progressive_1 != null) {
                    int state = Progressive.e_ToBeContinued;
                    while (state == Progressive.e_ToBeContinued) {
                        state = progressive_1.resume();
                    }
                    if (state != Progressive.e_Finished) continue;
                }
                int verifiedState = sign.getState();
                if ((verifiedState & sign.e_StateVerifyValid) == sign.e_StateVerifyValid) {
                    Log.d("Signature", "addNewSignatureAndSign: Signature" + i + "is valid.");
                }
            }
        }
    } catch (PDFException e) {
```

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

6 Creating a custom tool

With Foxit PDF SDK for Android, creating a custom tool is a simple process. There are several tools implemented in the UI Extensions Component already. These tools can be used as a base for developers to build upon or use as a reference to create a new tool. In order to create your own tool quickly, we suggest you take a look at the ***uiextensions_src*** project found in the "libs" folder.

To create a new tool, the most important step is to create a Java class that implements the "**ToolHandler.java**" interface.

In this section, we will make a Regional Screenshot Tool to show how to create a custom tool with Foxit PDF SDK for Android. This tool can help the users who only want to select an area in a PDF page to capture, and then save it as an image. Now, let's do it.

For convenience, we will build this tool based on the "**viewer_ctrl_demo**" project found in the "samples" folder. Steps required for implementing this tool are as follows:

- Create a Java class named **ScreenCaptureToolHandler** that implements the "**ToolHandler.java**" interface.
- Handle the **onTouchEvent** and **onDraw** events.
- Instantiate a **ScreenCaptureToolHandler** object, and then register it to the UIExtensionsManager.
- Set the **ScreenCaptureToolHandler** object as the current tool handler.

Step 1: Create a Java class named **ScreenCaptureToolHandler** that implements the "**ToolHandler.java**" interface.

a) Load the "**viewer_ctrl_demo**" project in Android Studio. Create a Java class named "**ScreenCaptureToolHandler**" in the "com.foxit.pdf.viewctrl" package.

b) Let the **ScreenCaptureToolHandler.java** class implement the **ToolHandler** interface as shown in Figure 6-1.

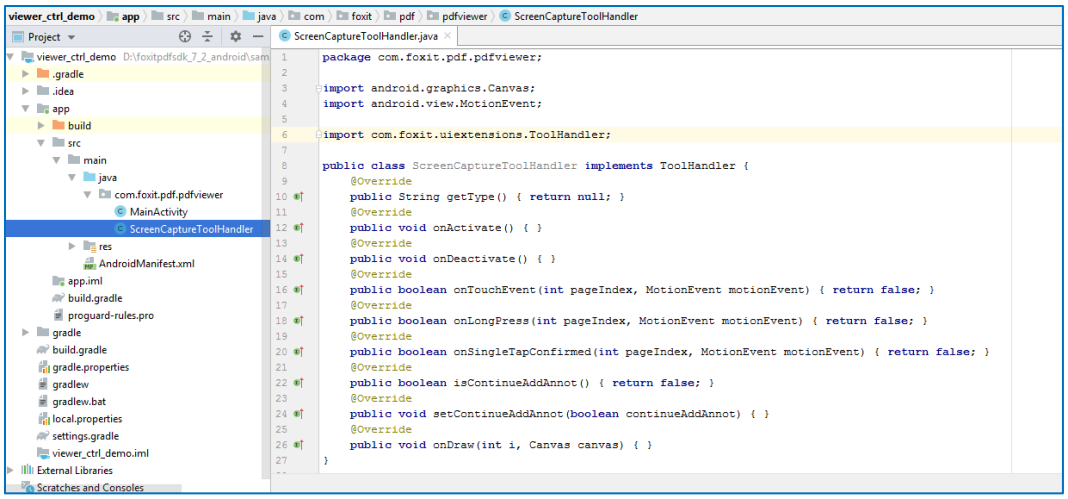


Figure 6-1

Step 2: Handle the **onTouchEvent** and **onDraw** events.

Update **ScreenCaptureToolHandler.java** as follows:

```
package com.foxit.pdf.pdfviewer;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.PointF;
import android.graphics.Rect;
import android.graphics.RectF;
import android.os.Environment;
import android.view.MotionEvent;
import android.widget.Toast;

import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.PDFException;
import com.foxit.sdk.common.Progressive;
import com.foxit.sdk.common.fxcrt.Matrix2D;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.common.Renderer;
import com.foxit.uiextensions.ToolHandler;
import com.foxit.uiextensions.UIExtensionsManager;
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class ScreenCaptureToolHandler implements ToolHandler {

    private Context mContext;
    private PDFViewCtrl mPdfViewCtrl;

    public ScreenCaptureToolHandler(Context context, PDFViewCtrl pdfViewCtrl) {
        mPdfViewCtrl = pdfViewCtrl;
        mContext = context;
    }

    @Override
    public String getType() {
        return "";
    }

    @Override
    public void onActivate() {

    }

    @Override
    public void onDeactivate() {

    }

    private PointF mStartPoint = new PointF(0, 0);
    private PointF mEndPoint = new PointF(0, 0);
    private PointF mDownPoint = new PointF(0, 0);
    private Rect mRect = new Rect(0, 0, 0, 0);
    private RectF mNowRect = new RectF(0, 0, 0, 0);
    private int mLastPageIndex = -1;
```

```
    // Handle OnTouch event
@Override
public boolean onTouchEvent(int pageIndex, MotionEvent motionEvent) {
    // Get the display view point in device coordinate system
    PointF devPt = new PointF(motionEvent.getX(), motionEvent.getY());
    PointF point = new PointF();
    // Convert display view point to page view point.
    mPdfViewCtrl.convertDisplayViewPtToPageViewPt(devPt, point, pageIndex);
    float x = point.x;
    float y = point.y;

    switch (motionEvent.getAction()) {
        // Handle ACTION_DOWN event: get the coordinates of the StartPoint.
        case MotionEvent.ACTION_DOWN:
            if (mLastPageIndex == -1 || mLastPageIndex == pageIndex) {
                mStartPoint.x = x;
                mStartPoint.y = y;
                mEndPoint.x = x;
                mEndPoint.y = y;
                mDownPoint.set(x, y);
                if (mLastPageIndex == -1) {
                    mLastPageIndex = pageIndex;
                }
            }
            return true;

        // Handle ACTION_Move event.
        case MotionEvent.ACTION_MOVE:
            if (mLastPageIndex != pageIndex)
                break;
            if (!mDownPoint.equals(x, y)) {
                mEndPoint.x = x;
                mEndPoint.y = y;

                // Get the coordinates of the Rect.
                getDrawRect(mStartPoint.x, mStartPoint.y, mEndPoint.x, mEndPoint.y);

                // Convert the coordinates of the Rect from float to integer.
                mRect.set((int) mNowRect.left, (int) mNowRect.top, (int) mNowRect.right, (int)
```

```
mNowRect.bottom);

                // Refresh the PdfViewCtrl, then the onDraw event will be triggered.
                mPdfViewCtrl.refresh(pageIndex, mRect);
                mDownPoint.set(x, y);
            }
            return true;

        // Save the selected area as a bitmap.
        case MotionEvent.ACTION_UP:
            if (mLastPageIndex != pageIndex)
                break;
            if (!mStartPoint.equals(mEndPoint.x, mEndPoint.y)) {
                renderToBmp(pageIndex, Environment.getExternalStorageDirectory().getPath() +
"/FoxitSDK/ScreenCapture.bmp");
                Toast.makeText(mContext, "The selected area was saved as a bitmap stored in the
/FoxitSDK/ScreenCapture.bmp", Toast.LENGTH_LONG).show();
            }
            mDownPoint.set(0, 0);
            mLastPageIndex = -1;
            return true;
        default:
            return true;
    }
    return true;
  }

  // Save a bimap to a specified path.
  public static void saveBitmap(Bitmap bm, String outPath) throws IOException {
    File file = new File(outPath);
    file.createNewFile();

    FileOutputStream fileout = null;
    try {
      fileout = new FileOutputStream(file);
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
```

```
        bm.compress(Bitmap.CompressFormat.JPEG, 100, fileout);
        try {
            fileout.flush();
            fileout.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Render the selected area to a bitmap.
    private void renderToBmp(int pageIndex, String filePath) {
        try {
            PDFPage page = mPdfViewCtrl.getDoc().getPage(pageIndex);

            mPdfViewCtrl.convertPageViewRectToPdfRect(mNowRect, mNowRect, pageIndex);
            int width = (int) page.getWidth();
            int height = (int) page.getHeight();

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

            // Create a Renderer object
            Renderer renderer = new Renderer(bmp, true);

            // Get the display matrix.
            Matrix2D matrix = page.getDisplayMatrix(0, 0, width, height, 0);
            Progressive progress = renderer.startRender(page, matrix, null);
            int state = Progressive.e_ToBeContinued;
            while (state == Progressive.e_ToBeContinued) {
                state = progress.resume();
            }

            // Create a bitmap with the size of the selected area.
            bmp = Bitmap.createBitmap(bmp, (int) mNowRect.left, (int) (height - mNowRect.top), (int)
mNowRect.width(), (int) Math.abs(mNowRect.height()));
            try {
                saveBitmap(bmp, filePath);
            } catch (IOException e) {
                e.printStackTrace();
```

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

// Get the coordinates of a Rect.
private void getDrawRect(float x1, float y1, float x2, float y2) {
    float minx = Math.min(x1, x2);
    float miny = Math.min(y1, y2);
    float maxx = Math.max(x1, x2);
    float maxy = Math.max(y1, y2);

    mNowRect.left = minx;
    mNowRect.top = miny;
    mNowRect.right = maxx;
    mNowRect.bottom = maxy;
}

@Override
public boolean onLongPress(int pageIndex, MotionEvent motionEvent) {
    return false;
}

@Override
public boolean onSingleTapConfirmed(int pageIndex, MotionEvent motionEvent) {
    return false;
}

@Override
public boolean isContinueAddAnnot() {
    return false;
}

@Override
public void setContinueAddAnnot(boolean continueAddAnnot) {

}
```

```
    // Handle the drawing event.
    @Override
    public void onDraw(int i, Canvas canvas) {
        if (((UIExtensionsManager) mPdfViewCtrl.getUIExtensionsManager()).getCurrentToolHandler() != this)
            return;
        if (mLastPageIndex != i) {
            return;
        }
        canvas.save();
        Paint mPaint = new Paint();
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setAntiAlias(true);
        mPaint.setDither(true);
        mPaint.setColor(Color.BLUE);
        mPaint.setAlpha(200);
        mPaint.setStrokeWidth(3);
        canvas.drawRect(mNowRect, mPaint);
        canvas.restore();
    }
}
```

Step 3: Instantiate a **ScreenCaptureToolHandler** object and then register it to the UIExtensionsManager.

```
private ScreenCaptureToolHandler screenCapture = null;
...
screenCapture = new ScreenCaptureToolHandler(mContext, pdfViewCtrl);
uiExtensionsManager.registerToolHandler(screenCapture);
```

Step 4: Set the **ScreenCaptureToolHandler** object as the current tool handler.

```
uiExtensionsManager.setCurrentToolHandler(screenCapture);
```

Now, we have really finished creating a custom tool. In order to see what the tool looks like, we need to make it run. Just add an action item and add the code referred in Step 3 and Step 4 to MainActivity.java.

First, add an action item in **Main.xml** found in "app/src/main/res/menu" as follows.

```
<item
  android:id="@+id/ScreenCapture"
  android:title="@string/screencapture"/>
```

In "app/src/main/res/values/strings.xml", add the following string:

```
<string name="screencapture">ScreenCapture</string>
```

Then, add the following code to the **onActionItemClicked()** function in **MainActivity.java**.

```
if (itemId == R.id.ScreenCapture) {
    if (screenCapture == null) {
        screenCapture = new ScreenCaptureToolHandler(mContext, pdfViewCtrl);
        uiExtensionsManager.registerToolHandler(screenCapture);
    }
    uiExtensionsManager.setCurrentToolHandler(screenCapture);
}
```

Please remember to instantiate a **ScreenCaptureToolHandler** object at first, like (**private** ScreenCaptureToolHandler **screenCapture** = **null**;).

After finishing all of the above work, build and run the demo.

***Note**: Here, we run the demo on an AVD targeting 10.0. Please make sure you have pushed the "Sample.pdf" document into the correct folder (which matches the path in the demo) of the emulator's SD card.*

After building the demo and installing the APK on the emulator successfully, tap **Allow** on the pop-up window to allow the demo to access files on the device. Click anywhere on the opened document to display the Contextual Action bar, and click ⋮ (overflow button) to find the **ScreenCapture** action button as shown in Figure 6-2.



Figure 6-2

Click **ScreenCapture**, long press and select a rectangular area, and then a message box will be popped up as shown in Figure 6-3. It shows where the bitmap (selected area) was saved to.


Foxit PDF SDK
for Android/iOS
PDF
www.foxit.com
1
The selected area was saved as
a bitmap stored in the /FoxitSDK/
ScreenCapture.bmp
Display by Foxit PDF SDK for Evaluation Only.
To purchase the Foxit PDF SDK, please contact us at:
+1-866-680-3668 (24/7)
sales@foxitsoftware.com

Figure 6-3

In order to verify whether the tool captures the selected area successfully, we need to find the screenshot. Click **Device Explorer** on the lower right side of the IDE, we can see the screenshot named "ScreenCapture.bmp" in the "FoxitSDK" folder of the SD card as shown in Figure 6-4.

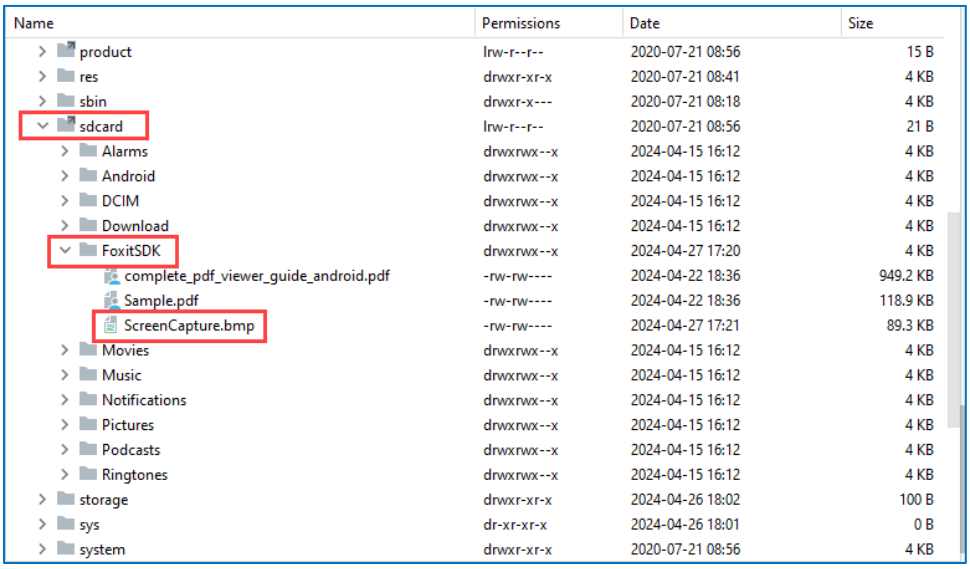


Figure 6-4

Right-click the "ScreenCapture.bmp" picture, click **Save AS...** to save it to a location as desired. Open the picture, we can see the image as shown in Figure 6-5.



Figure 6-5

As you can see we have successfully created a Regional Screenshot Tool. This is just an example to show how to create a custom tool with Foxit PDF SDK for Android. You can refer to it or our demos to develop the tools you want.

7 Implement Foxit PDF SDK for Android using Cordova

When it comes to developing cross-platform mobile applications, Apache Cordova is an ideal open-source framework. The '**cordova-plugin-foxitpdf**' is one of the mobile framework plugins provided by us to use with Foxit PDF SDK for Android. The plugin enables you to achieve powerful PDF viewing features using the Cordova framework. Through this plugin, you can preview any PDF file including PDF 2.0 compliant files, XFA documents, and RMS protected documents, as well as commenting and editing PDF documents.

For the usage of 'cordova-plugin-foxitpdf', please refer to the website https://github.com/foxitsoftware/cordova-plugin-foxitpdf.

8 Implement Foxit PDF SDK for Android using React Native

React Native is an open-source mobile development framework for building native apps using JavaScript and React. The '**react-native-foxitpdf**' is only one of the mobile framework plugins provided by us to use with Foxit PDF SDK for Android. It allows you to achieve powerful PDF viewing features using the React Native framework. Through this plugin, you can preview any PDF file including PDF 2.0 compliant files, XFA documents, and RMS protected documents, as well as commenting and editing PDF documents.

For the usage of 'react-native-foxitpdf' plugin, please refer to the website https://github.com/foxitsoftware/react-native-foxitpdf.

9 Implement Foxit PDF SDK for Android using Xamarin

Xamarin is a cross-platform development framework for building native apps using a shared C# codebase. We provide separate bindings for Android and iOS ('foxit_xamarin_android' and 'foxit_xamarin_ios') for developers to seamlessly integrate powerful PDF functionality of Foxit PDF SDK into their Xamarin apps.

For the usage of 'foxit_xamarin_android' plugin, please refer to the website https://github.com/foxitsoftware/xamarin-foxitpdf.

10 FAQ

10.1 Open a PDF document from a specified PDF file path

How do I open a PDF document from a specified PDF file path?

Foxit PDF SDK for Android provides multiple interfaces to open a PDF document. You can open a PDF document from a specified PDF file path, or from a memory buffer. For from a specified PDF file path, there are two ways to do that.

The **first** one is that just use the ***openDoc*** interface, which includes the operations of creating a PDF document object (***PDFDoc(String path)***), loading the document content (***load***), and setting the PDF document object to view control (***setDoc***). Following is the sample code:

Note: *The* ***openDoc*** *interface is only available for opening a PDF document from a file path. If you want to customize to load a PDF document, you can implement it in the callback function (****FileRead****), and then create a document object with a FireRead instance using* ***PDFDoc(FileRead fileRead)****. Next, also load the document content using* ***load****, and set the PDF document object to view control using* ***setDoc****.*

```
// Assuming A PDFViewCtrl has been created.

// Open an unencrypted PDF document from a specified PDF file path.
String path = Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf";
pdfViewCtrl.openDoc(path, null);
```

The **second** one is that use the ***PDFDoc(String path)*** interface to create a PDF document object , use ***load*** interface to load the document content, and then use ***setDoc*** to set the PDF document object to view control. Following is the sample code:

```
// Assuming A PDFViewCtrl has been created.

String path = Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf";
try {
    // Initialize a PDFDoc object with the path to the PDF file.
    PDFDoc document = new PDFDoc(path);

    // Load the unencrypted document content.
    document.load(null);
```

```
    // Set the document to view control.
    pdfViewCtrl.setDoc(document);
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

10.2 Display a specified page when opening a PDF document

What should I do if I want to display a specified page when opening a PDF document?

To display a specified page when opening a PDF file, the interface ***gotoPage (int pageIndex)*** *should be used.* Foxit PDF SDK for Android utilizes multi-thread to improve rendering speed, so please make sure the document has been loaded successfully before using the ***gotoPage*** interface.

Please implement the callback interface in the **IDocEventListener,** and then call the ***gotoPage*** interface in the ***onDocOpened*** event. Following is the sample code:

```
// Assuming A PDFViewCtrl has been created.

// Register the PDF document event listener.
pdfViewCtrl.registerDocEventListener(docListener);

// Open an unencrypted PDF document from a specified PDF file path.
String path = Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf";
pdfViewCtrl.openDoc(path, null);

...

PDFViewCtrl.IDocEventListener docListener = new PDFViewCtrl.IDocEventListener() {
    @Override
    public void onDocWillOpen() {}

    @Override
    public void onDocOpened(PDFDoc pdfDoc, int errCode) {
      pdfViewCtrl.gotoPage(2);
    }

    @Override
    public void onDocWillClose(PDFDoc pdfDoc) { }
```

```
    @Override
    public void onDocClosed(PDFDoc pdfDoc, int i) { }

    @Override
    public void onDocWillSave(PDFDoc pdfDoc) { }

    @Override
    public void onDocSaved(PDFDoc pdfDoc, int i) { }

};
```

10.3 License key and serial number cannot work

I have downloaded the SDK package from your website without making any changes. Why can't the license key and serial number work?

Generally, the package uploaded to the website is supposed to work. It has been tested before it is uploaded. So, if you find the license key and serial number cannot work, it may be caused by the date of your device. If the device's date is earlier than the **StartDate** in the **rdk_key.txt** file found in the "libs" folder of the download package, the "librdk.so" library will be failed to unlock. Please check the date of your device.

10.4 Add a link annotation to a PDF file

How can I add a link annotation to a PDF file?

To add a link annotation to a PDF file, you should first call the **PDFPage.addAnnot** to add a link annotation to a specified page, then call **Action.Create** to create an action, and set the action to the added link annotation. Following is the sample code for adding a URI link annotation to the first page of a PDF file:

```
private Link linkAnnot = null;
...

String path = Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf";
try {

    // Initialize a PDFDoc object with the path to the PDF file.
    PDFDoc document = new PDFDoc(path);

    // Load the unencrypted document content.
    document.load(null);
```

```
    // Get the first page of the PDF file.
    PDFPage page = document.getPage(0);

    // Add a link annotation to the first page.
        linkAnnot = new Link (page.addAnnot(Annot.e_Link, new RectF(250, 650, 400, 750)));

    // Create a URI action and set the URI.
    URIAction uriAction = new URIAction(Action.create(document, Action.e_TypeURI));
    uriAction.setURI("www.foxitsoftware.com");

    // Set the action to link annotation.
    linkAnnot.setAction(uriAction);

    // Reset appearance stream.
    linkAnnot.resetAppearanceStream();

    // Save the document that has added the link annotation.
    document.saveAs(Environment.getExternalStorageDirectory().getAbsolutePath()+
"/FoxitSDK/sample_annot.pdf", PDFDoc.e_SaveFlagNormal);

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

10.5 Insert an image into a PDF file

How do I insert an image into a PDF file?

There are two ways to help you insert an image into a PDF file. The first one is calling **PDFPage.addImageFromFilePath** interface. You can refer to the following sample code which inserts an image into the first page of a PDF file:

Note: *Before calling* ***PDFPage.addImageFromFilePath*** *interface, you should get and parse the page that you want to add the image.*

```
String path = Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf";
try {

    // Initialize a PDFDoc object with the path to the PDF file.
    PDFDoc document = new PDFDoc(path);

    // Load the unencrypted document content.
    document.load(null);

    // Get the first page of the PDF file.
    PDFPage page = document.getPage(0);

    // Parse the page.
```

```
    if (!page.isParsed()) {
        Progressive parse = page.startParse(e_ParsePageNormal, null, false);
        int state = Progressive.e_ToBeContinued;
        while (state == Progressive.e_ToBeContinued) {
            state = parse.resume();
        }
    }

    // Add an image to the first page.
    page.addImageFromFilePath(Environment.getExternalStorageDirectory().getAbsolutePath()+
"/FoxitSDK/2.png", new PointF(20, 30), 60, 50, true)
    // Save the document that has added the image.
    document.saveAs(Environment.getExternalStorageDirectory().getAbsolutePath()+
"/FoxitSDK/sample_image.pdf", PDFDoc.e_SaveFlagNormal);

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

The second one is that use the **PDFPage.addAnnot** interface to add a stamp annotation to a specified page, and then convert the image to a bitmap and set the bitmap to the added stamp annotation. You can refer to the following sample code which inserts an image as a stamp annotation into the first page of a PDF file:

```
String path = Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf";

try {

    // Initialize a PDFDoc object with the path to the PDF file.
    PDFDoc document = new PDFDoc(path);

    // Load the unencrypted document content.
    document.load(null);

    // Get the first page of the PDF file.
    PDFPage page = document.getPage(0);

    // Add a stamp annotation to the first page.
    Stamp stamp = new Stamp(page.addAnnot(Annot.e_Stamp, new RectF(100, 350, 250, 150)));

    // Load a local image and convert it to a Bitmap.
    Bitmap bitmap = BitmapFactory.decodeFile(Environment.getExternalStorageDirectory().getAbsolutePath()+
"/FoxitSDK/2.png");

    // Set the bitmap to the added stamp annotation.
    stamp.setBitmap(bitmap);

    //Reset appearance stream.
    stamp.resetAppearanceStream();
```

```
    // Save the document that has added the stamp annotation.
    document.saveAs( Environment.getExternalStorageDirectory().getAbsolutePath()+
"/FOXitSDK/sample_image.pdf", PDFDoc.e_SaveFlagNormal);

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

10.6 SetDocModified API

Is there an API for informing the upper-level SDK when PDF document has been modified?

Yes. Foxit PDF SDK for Android has an internal API (**SetDocModified**) to inform the upper-level SDK that the PDF document has been modified. For example:

```
// Assume you have already Initialized a UIExtensionsManager object.
uiExtensionsManager.getDocumentManager().setDocModified(true);
```

10.7 Highlight the links in PDF documents and set the highlight color

How can I set whether to highlight the links in PDF documents? And how to set the highlight color if I want to highlight links?

By default, highlighting links in PDF documents is enabled. If you want to disable it or to set the highlight color, you can do it in the configuration JSON file (only support for version 6.3 or higher) or by calling the APIs.

***Note**: If you want to set the highlight color, please make sure the highlighting links feature is enabled.*

Through JSON file

Set "**"highlightLink"**: **false**," to disable highlighting the links in PDF document.
Set "**"highlightLinkColor"**: **"#16007000"**," to set the highlight color (input the color value as you wish).

Through calling API

UIExtensionsManager.enableLinkHighlight() interface is provided to set whether to enable highlighting the links in PDF documents. If you do not want to highlight links, please set the parameter to "false" as follows:

```
// Assume you have already Initialized a UIExtensionsManager object
uiExtensionsManager.enableLinkHighlight(false);
```

UIExtensionsManager.setLinkHighlightColor() interface is used to set the highlight color. Following is a sample for calling this API:

```
// Assume you have already Initialized a UIExtensionsManager object
uiExtensionsManager.setLinkHighlightColor(0x4b0000ff);
```

10.8 Highlight the form fields in PDF form files and set the highlight color

How can I set whether to highlight the form fields in PDF form files? And how to set the highlight color if I want to highlight form fields?

By default, highlighting form fields in PDF documents is enabled. If you want to disable it or to set the highlight color, you can do it in the configuration JSON file (only support for version 6.3 or higher) or by calling the APIs.

***Note**: If you want to set the highlight color, please make sure the highlighting form fields feature is enabled.*

Through JSON file

Set ""**highlightForm**": **false**," to disable highlighting the form fields in PDF document.

Set ""**highlightFormColor**": **"#2000ffcc"**," to set the highlight color (input the color value as you wish).

Through calling API

UIExtensionsManager.enableFormHighlight() interface is provided to set whether to enable highlighting the form fields in PDF form files. If you do not want to highlight form fields, please set the parameter to "false" as follows:

```
// Assume you have already Initialized a UIExtensionsManager object
uiExtensionsManager.enableFormHighlight(false);
```

UIExtensionsManager.setFormHighlightColor() interface is used to set the highlight color. Following is a sample for calling this API:

```
// Set the highlight color to blue.
uiExtensionsManager.setFormHighlightColor(0x4b0000ff);
```

10.9 Indexed Full Text Search support

Does Foxit PDF SDK for Android support Indexed Full Text Search? If yes, how can I use it to search through PDF files stored offline on my mobile device?

Yes. Foxit PDF SDK for Android supported Indexed Full Text Search from version 5.0.

To use this feature, follows the steps below:

a) Get document source information. Create a document source based on a directory which will be used as the search directory.

```
public DocumentsSource(String directory)
```

b) Create a full text search object, and set a path of database to store the indexed data.

```
public FullTextSearch()
public void setDataBasePath(String pathOfDataBase)
```

c) Start to index the PDF documents which receive from the source.

```
public Progressive startUpdateIndex(DocumentsSource source,
    PauseCallback pause, boolean reUpdate)
```

Note: *You can index a specified PDF file. For example, if the contents of a PDF file have been changed, you can re-index it using the following API:*

```
public boolean updateIndexWithFilePath(java.lang.String filePath)
```

d) Search the specified keyword from the indexed data source. The search results will be returned to external by a specified callback function when a matched one is found.

```
public boolean searchOf(java.lang.String matchString,
    RankMode rankMode,
    SearchCallback searchCallback)
```

Following is a sample for how to use it:

```
String directory = "A search directory...";
FullTextSearch search = new FullTextSearch();
try {
   String dbPath = "The path of data base to store the indexed data...";
   search.setDataBasePath(dbPath);
   // Get document source information.
   DocumentsSource source = new DocumentsSource(directory);

   // Create a Pause callback object implemented by users to pause the updating process.
   PauseUtil pause = new PauseUtil(30);

   // Start to update the index of PDF files which receive from the source.
   Progressive progressive = search.startUpdateIndex(source, pause, false);
   int state = Progressive.e_ToBeContinued;
   while (state == Progressive.e_ToBeContinued) {
      state = progressive.resume();
   }

   // Create a callback object which will be invoked when a matched one is found.
```

```
    MySearchCallback searchCallback = new MySearchCallback();

    // Search the specified keyword from the indexed data source.
    search.searchOf("looking for this text", RankMode.e_RankHitCountASC, searchCallback);
} catch (PDFException e) {
    e.printStackTrace();
}
```

A sample callback for **PauseUtil** is as follows:

```
public class PauseUtil extends PauseCallback{
    private long m_milliseconds = 0;
    private long m_startTime = 0;

    public PauseUtil(long milliSeconds) {
        Date date = new Date();
        m_milliseconds = milliSeconds;
        m_startTime = date.getTime();
    }

    @Override
    public boolean needToPauseNow() {
        // TODO Auto-generated method stub
        if (this.m_milliseconds < 1)
            return false;
        Date date = new Date();
        long diff = date.getTime() - m_startTime;
        if (diff > this.m_milliseconds) {
            m_startTime = date.getTime();
            return true;
        } else
            return false;
    }
}
```

A sample callback for **MySearchCallback** is as follows:

```
public class MySearchCallback extends SearchCallback {
    private static final String TAG = MySearchCallback.class.getCanonicalName();

    @Override
    public void release() {
    }

    @Override
    public int retrieveSearchResult(String filePath, int pageIndex, String matchResult, int
matchStartTextIndex, int matchEndTextIndex) {
        String s = String.format("Found file is :%s \n Page index is :%d Start text index :%d End text
index :%d \n Match is :%s \n\n", filePath, pageIndex, matchStartTextIndex, matchEndTextIndex,
matchResult);
        Log.v(TAG, "retrieveSearchResult: " + s);
```

```
        return 0;
    }
}
```

Note:

- *The indexed full text search provided by Foxit PDF SDK for Android will go through a directory recursively, so that both the files and the folders under the search directory will be indexed.*
- *If you want to abort the index process, you can pass in a pause callback parameter to the* ***startUpdateIndex*** *interface. The callback function* ***needToPauseNow*** *will be invoked once a PDF document is indexed, so that the caller can abort the index process when the callback* ***needToPauseNow*** *return "true".*
- *The location of the indexed database is set by* ***setDataBasePath*** *interface. If you want to clear the indexed database, you shoud do it manually. And now, removing a file from index function is not supported.*
- *Every search result of the* ***searchOf*** *interface is returned* to external by a specified callback. Once the ***searchOf*** *interface returns "true" or "false", it means the searching is finished.*

10.10 Print PDF document

Does Foxit PDF SDK for Android support to print a PDF document? If yes, how can I use it?

Yes. Foxit PDF SDK for Android supported the print feature from version 5.1. You can press the Wireless Print button on the More Menu view in the Complete PDF viewer demo to print the PDF document. Furthermore, you can call the following API to print the PDF documents:

// for iPhone and iTouch

```
public void startPrintJob(Context context, PDFDoc doc, String printJobName, String outputFileName,
IPrintResultCallback callback)
```

Following is a sample for how to use it:

```
// Assume you have already Initialized a UIExtensionsManager object

PDFDoc doc = null;
IPrintResultCallback print_callback = new IPrintResultCallback() {
    @Override
    public void printFinished() {
    }

    @Override
    public void printFailed() {
```

```
    }

    @Override
    public void printCancelled() {
    }
};

try {
    doc = new PDFDoc(Environment.getExternalStorageDirectory().getAbsolutePath()+ "/FoxitSDK/Sample.pdf");
    doc.load(null)
} catch (PDFException e) {
    Assert.fail("unexpect a PDF Exception!!errCode = " + e.getLastError());
    }
uiExtensionsManager.startPrintJob(getActivity(), doc, "print with name", "print_withAPI", print_callback);
}
```

10.11 Night mode color settings

How can I set the night mode color?

if you wan to set the night mode color, please first call the **PDFViewCtrl.setMappingModeBackgroundColor(int)** and **PDFViewCtrl.setMappingModeForegroundColor(int)** APIs to set the color as you wish, and then set the color mode by using **PDFViewCtrl.setColorMode(int)**.

Note: *If the color mode is already set to Renderer.e_ColorModeMapping/ Renderer.e_ColorModeMappingGray, you still need to set it again after calling* ***PDFViewCtrl.setMappingModeBackgroundColor(int)*** *and* ***PDFViewCtrl.setMappingModeForegroundColor(int)****. Otherwise, the settings may not work.*

The above APIs should be called in the source code of the UI Extensions Component, please refer to section 4.3 "Customize UI implementation through source code" to add the "uiextensions_src" project found in the "libs" folder to your project. Then, find the **onValueChanged** function in "com.foxit.uiextensions.pdfreader.impl.MainFrame.java", and set the color as you like.

Following is a sample to set the night mode color:

```
case IViewSettingsWindow.TYPE_DAY:
    mPageColorMode = (Integer) value;
    if (mPageColorMode == IViewSettingsWindow.DAY) {
        mUiExtensionsManager.getPDFViewCtrl().setBackgroundColor(AppResource.getColor(mContext,
R.color.ux_bg_color_docviewer));
        mUiExtensionsManager.getPDFViewCtrl().setNightMode(false);
    } else if (mPageColorMode == IViewSettingsWindow.NIGHT) {
```

```
        mUiExtensionsManager.getPDFViewCtrl().setBackgroundColor(Color.parseColor("#36404A"));
        mUiExtensionsManager.getPDFViewCtrl().setNightMode(true);
        mUiExtensionsManager.setNightColorMode(UIExtensionsManager.NIGHTCOLORMODE_MAPPINGGRAY);
        if (mUiExtensionsManager.getNightColorMode() ==
UIExtensionsManager.NIGHTCOLORMODE_MAPPINGGRAY) {
            mUiExtensionsManager.getPDFViewCtrl().setMappingModeForegroundColor(Color.argb(0xff, 0x00, 0xff,
0x00));
            mUiExtensionsManager.getPDFViewCtrl().setMappingModeBackgroundColor(Color.argb(0xff, 0xff, 0x00,
0x00));
            mUiExtensionsManager.getPDFViewCtrl().setColorMode(Renderer.e_ColorModeMappingGray);
        }
    }
```

10.12 Output exception/crash log information

How can I output exception/crash log information when my app throws exceptions or crashes?

setExceptionLogger interface is provided to output exception/crash log information, which references the third-party library of xCrash. The usage is as follows:

1) Add dependency.

```
dependencies {
            implementation 'com.iqiyi.xcrash:xcrash-android-lib:2.1.4'
}
```

2) Specify one or more ABI(s) you need.

```
android {
    defaultConfig {
        ndk {
            abiFilters 'armeabi', 'armeabi-v7a', 'arm64-v8a', 'x86', 'x86_64'
        }
    }
}
```

3) Call **setExceptionLogger** in your code.

```
public class MainApplication extends Application {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        PDFViewCtrl.setExceptionLogger(this, Environment.getExternalStorageDirectory().getAbsolutePath()+
"/FoxitSDK/crash", new PDFViewCtrl.IExceptionLogger() {
```

```
        @Override
        public void onExceptionLogger(String filePath)

        { Log.d("", "onExceptionLogger: " + filePath); }
      });
  }
}
```

10.13 Reduce size of APK

How do I reduce the size of my APK?

To reduce the size of an APK, you can build multiple APKs that contain files for specific screen densities or ABIs. For details about multiple APKs, see Build Multiple APKs. The following code snippet in module-level **build.gradle** file configures multiple APKs based on screen density and ABI.

```
android {
 ...
  splits {
    // Configures multiple APKs based on screen density.
    density {
      // Configures multiple APKs based on screen density.
      enable true
      // Specifies a list of screen densities Gradle should not create multiple APKs for.
      exclude "ldpi", "xxhdpi", "xxxhdpi"
      // Specifies a list of compatible screen size settings for the manifest.
      compatibleScreens 'small', 'normal', 'large', 'xlarge'
    }
    // Configures multiple APKs based on ABI.
    abi {
      // Enables building multiple APKs per ABI.
      enable true
      // By default all ABIs are included, so use reset() and include to specify that we only
      // want APKs for x86 and x86_64.
      // Resets the list of ABIs that Gradle should create APKs for to none.
      reset()
      // Specifies a list of ABIs that Gradle should create APKs for.
      include "x86", "x86_64", "armeabi-v7a", "arm64-v8a"
      // Specifies that we do not want to also generate a universal APK that includes all ABIs.
      universalApk false
    }
  }
}
```

10.14 Enable shrink-code (set "minifyEnabled" to "true")

Why do I encounter some exceptions at run time when I set "minifyEnabled" to "true" in the App's build.gradle?

When you set "**minifyEnabled**" to "**true**" in the App's build.gradle to enable shrink-code, please note that you should add the contents below to the **proguard-rules.pro** file, otherwise it may throw exceptions at run time.

In the **proguard-rules.pro** file:

```
-dontwarn com.foxit.sdk.**
-keep class com.foxit.sdk.**{ *;}

-dontwarn com.microsoft.rightsmanagement.**
-keep class com.microsoft.rightsmanagement.** {*;}

-dontwarn com.microsoft.aad.adal.**
-keep class com.microsoft.aad.adal.** {*;}

-dontwarn com.edmodo.cropper.**
-keep class com.edmodo.cropper.** {*;}

-dontwarn org.bouncycastle**
-keep class org.bouncycastle.** {*;}
```

10.15 Localization settings

How to change Localization settings with Foxit PDF SDK for Android?

By default, Foxit PDF SDK for Android will automatically switch the UI language according to the current language of your system, provided that the language is supported by Foxit PDF SDK for Android.

Currently, Foxit PDF SDK for Android supports the following languages: English, German(de-CH, de-DE), Latin(es-LA), Frence(fr-FR), Italian(it-IT), Korean(ko), Dutch(ni-NL), Portuguese(pt-BR), Russian(ru-Ru), and Chinese(zh-CN, zh-TW). Those language resource files are located in the "libs\uiextensions_src\src\main\res" folder.

If you want to use your own localization language that is not supported by Foxit PDF SDK for Android, you should first translate all the entries on the UI, and place the localization resource files into the same directory with others language resource files in your own project. Then, change your current system language as you want, or call **Localization.setCurrentLanguage** interface to make it work. For more details about **Localization.setCurrentLanguage** interface, please refer to the API Reference in the "doc" folder.

For example, assume that you want to change the UI language to Japanese in the "complete_pdf_viewer" demo. You can follow the steps below:

a) Copy the "**values**" (for example) folder from "libs\uiextensions_src\src\main\res" to the demo resource directory "samples\complete_pdf_viewer\app\src\main\res", and rename it to "**values-ja-rJA**".

b) Translate (Localize) all the entries in the XML file under "**values-ja-rJA**" folder.

c) To make the localized language work, you can choose one of the following ways:

- Change the language of your current system to Japanese.
- Call **Localization.setCurrentLanguage** interface to set current language to Japanese.

```
Locale locale = new Locale("ja","JA");
Localization.setCurrentLanguage(this.getContext(), locale);
```

10.16 Migrate to AndroidX

How do I migrate Foxit PDF SDK for Android to AndroidX?

From version 7.2, Foxit PDF SDK for Android will only support AndroidX, and no longer support Android support library. Therefore, if you are using the SDK version before 7.2, and want to use the new features released in the newer SDK libraries, you should migrate Foxit PDF SDK for Android to AndroidX.

Requirements for migration

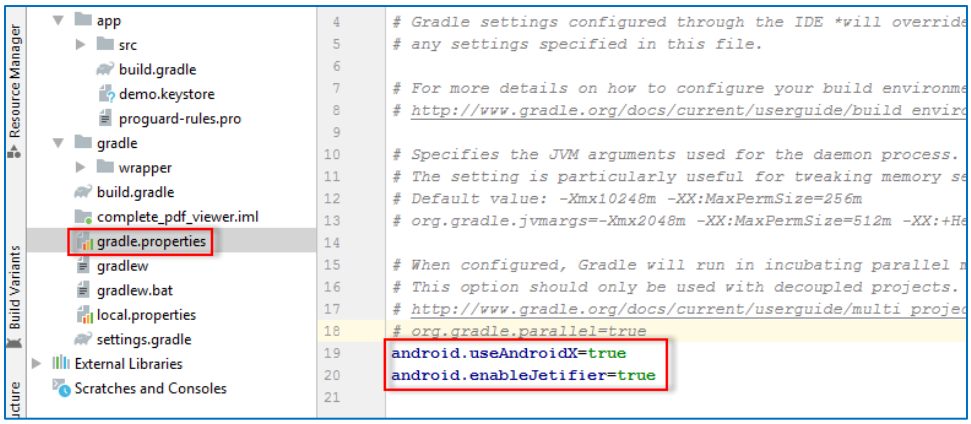
- Android Studio version >= 3.2.0
- Gradle Build Tool >= 3.2.0
- Gradle Version >= 4.6
- compileSdkVersion >= 28

Migration

For migration, you can refer to the official migration guide. Here, we take the **complete_pdf_viewer** demo in the "samples" folder as an example to show you how to migrate to AndroidX.

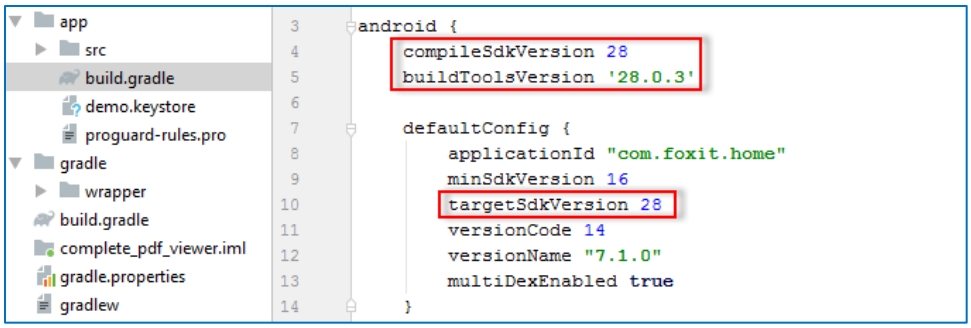
1) Change the project-level "gradle.properties". Add the contents below to "gradle.properties":

android.useAndroidX=true
android.enableJetifier=true



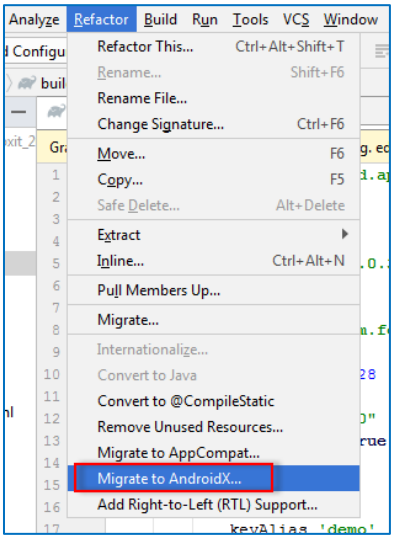
2) Change the related dependencies in the app's "build.gradle".

a) Change **compileSdkVersion** to 28 or higher:



b) Upgrate the version of '**com.android.tools.build:gradle**' in the project-level "build.gradle" to 3.2.0 or higher.

c) Click **Refactor** > **Migrate to AndroidX** to start the migration.



d) Upgrate the third-party libraries. If the third-party libraries have referenced the related Android support libraries, then the third-party libraries also need to migrate to AndroidX.

Before upgrating, the dependencies in app's "build.gradle" is as below:

```
dependencies {
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support:multidex:1.0.+'
    implementation(name: 'FoxitRDK', ext: 'aar')
    implementation(name: 'FoxitRDKUIExtensions', ext: 'aar')
    implementation 'com.edmodo:cropper:1.0.1'
    implementation('com.microsoft.aad:adal:1.1.16') {}
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
    // RxJava
    implementation "io.reactivex.rxjava2:rxjava:2.2.2"
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.0'
    implementation 'org.bouncycastle:bcpkix-jdk15on:1.60'
    implementation 'org.bouncycastle:bcprov-jdk15on:1.60'
}
```

After upgrating, the dependencies in app's "build.gradle" is as below:

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.multidex:multidex:2.0.1'
    implementation(name: 'FoxitRDK', ext: 'aar')
    implementation(name: 'FoxitRDKUIExtensions', ext: 'aar')
    implementation 'com.edmodo:cropper:1.0.1'
    implementation 'com.microsoft.identity.client:msal:2.+'
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
    // RxJava
    implementation "io.reactivex.rxjava2:rxjava:2.2.16"
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'
    implementation 'org.bouncycastle:bcpkix-jdk15on:1.64'
    implementation 'org.bouncycastle:bcprov-jdk15on:1.64'
}
```

After migrating, you may need to modify some packages manually, because in the process of automatic migration, some packages may not be imported correctly. Besides, you may encounter some other problems, in this case, you should deal with the problem according to the actual situation.

10.17 Support Chromebook

Does Foxit PDF SDK for Android support Chromebook?

Yes. Foxit PDF SDK for Android support Chromebook. But you should add "**android:name="com.foxit.uiextensions.FoxitApplication"**" to the AndroidManifest.xml file. And if you want to customize application, please extend **FoxitApplication** class.

10.18 Use UIExtensions for read-only mode

How can I use UIExtensions for read-only mode?

1：Interface Definition

Interfaces of the UIExtensionsManager：

```
/**
 * whether the pdf document can be modified
 *
 * @return whether the pdf document can be modified
 */
public boolean isEnableModification() {
    return mIsEnableModify;
}

/**
 * Set whether the pdf document can be modified. The default is allow modification.
 *
 * @param isEnabled whether the pdf document can be modified
 */
public void enableModification(boolean isEnabled) {
    mIsEnableModify = isEnabled;
}
```

2：Interface Usage

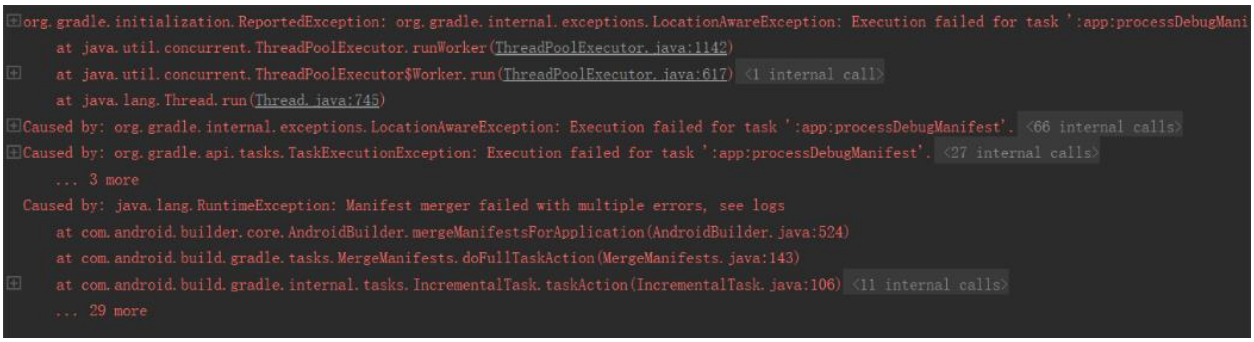
By default, documents without permission control can be edited. If you need to disable editing related interactive functions, you can set them by the following methods:

```
mUiExtensionsManager.enableModification(false);
```

10.19 Compatible with Android Studio 3.2

How to be compatible with Android Studio 3.2

If your Android development IDE is still the Android Studio 3.2, and you will not upgrade it to the Anroid Studio 4.1 for some reasons, the following error will be catched when building the RDK Demo.



To fix this error, you can refine the configurations based on the following steps:

Change the **com.android.tools.build:gradle** of the ./build.gradle that is the Demo project configuration file to 3.3.3:

classpath 'com.android.tools.build:gradle:3.3.3'

Change the **distributionUrl** of the ./gradle/wrapper/gradle-wrapper.properties that is the Demo project configuration file to distributionUrl=https\://services.gradle.org/distributions/gradle-4.10.1-all.zip.

10.20 Update page binding to support Right-to-Left

How can I automatically update page binding to support Right-to-Left?

For most languages, the reading habits we use are left-to-right, which calls for a page binding on the left edge. However, there are also some languages that read from right to left, such as Arabic and Hebrew and several East Asian scripts. In this case, binding on the right edge is preferable for users, which the pages will be arranged from right to left (the first page is on the top right). To do this, we made the adaptation of the right-to-left page layout.

The page binding is used with horizontal scrolling. For vertical scrolling, it has effect only when double-page mode is enabled.

Updating the Page Binding Programmatically

Foxit PDF SDK for Android defines constants **LEFT_EDGE** and **RIGHT_EDGE** in the com.foxit.sdk.PDFViewCtrl class:

- LEFT_EDGE: the document flows from left to right
- RIGHT_EDGE: the document flows from right to left

```
/** The document flows from left to right.*/
```

```
public static final int LEFT_EDGE = 0;

/** The document flows from right to left.*/
public static final int RIGHT_EDGE = 1;

/**
 * The page bindings deciding how the document will be displayed.
 *
 * @see #LEFT_EDGE
 * @see #RIGHT_EDGE
 */
@IntDef({LEFT_EDGE, RIGHT_EDGE})
@Retention(RetentionPolicy.SOURCE)
public @interface PageBinding {
}
```

Then, call the following function to update the page binding to swith the page layout:

```
pdfViewerCtrl.setPageBinding(PDFViewCtrl.RIGHT_EDGE);
```

The UI result after using this function:

In horizontal scrolling: (LTR: left-to-right; RTL: right-to-left)

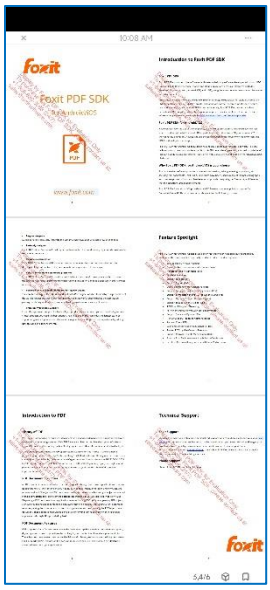


LTR

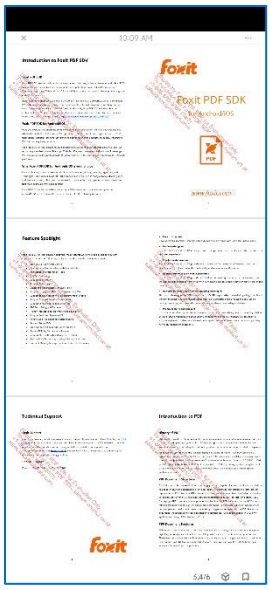


RTL

In vertical scrolling: (has effect in double-page mode (Facing/Coving)):



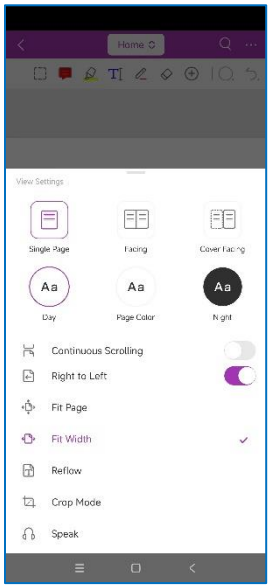
LTR



RTL

Updating the Page Binding via the UI

You can create a demo using **FoxitRDKUIExtensions.aar** or use **complete_pdf_viewer** demo directly to update the page binding. After running the demo, find the **View settings** bar, you can enable or disable the **Right to Letf** as below:



10.21 Issue with opening web PDFs

How can I resolve the issue that some documents may not display correctly or cannot be displayed at all when opening web PDFs using openDocFromUrl(String url, byte[] password, CacheOption cacheOption, HttpRequestProperties properties) **interface?**

The issue occurs when some documents with numerous objects are closed before they finish loading, resulting in an incomplete caching process. When attempting to load the data, there is no current method to determine the data's validity, only whether it has been cached.

To resolve this issue, it is recommended for users to clear cache data and reload the document using the following method. This approach should not impact document opening speed, as the SDK internally loads web pages based on page numbers.

```
/**
  * Clear the cache file by the specified url.
  *
  * @param url The url of the document that used in {@link #openDocFromUrl(String, byte[], CacheOption,
HttpRequestProperties)}
  * @see #openDocFromUrl(String, byte[], CacheOption, HttpRequestProperties)
*/
public void clearCacheFile(String url) {
    docManager.clearUrlCache(url);
}

/**
  * Clear all the cache files.
  *
  * @see #openDocFromUrl(String, byte[], CacheOption, HttpRequestProperties)
*/
public void clearAllCacheFiles() {
    docManager.clearUrlCache(null);
}
```

10.22 Improve efficiency in inserting and rendering watermarks

How can I improve efficiency in inserting and rendering watermarks?

Some customers encounter issues such as delays and incomplete watermark display, when adding a custom watermark to their product.

Cause Analysis:

1. The delay issue may be due to repetitive page parsing. If the watermark is the same, there's no need to create a new one every time.

2. The watermark might be partially displayed because the SDK internally prevents frequent refreshing to maintain efficiency. If the last refresh hasn't finished, the next refresh may not take effect, causing the watermark to be partially missing.

Solution:

1. Use com.foxit.sdk.Task to add a watermark

   Create a task to add the watermark, and wait for all SDK internal threads to finish processing before handling customer logic. This can solve the issue of the watermark being partially missing.

2. Use segmented loading to add a watermark

   For example, add the watermark when loading pages 1-10 for the first time, and when switching to page 11, load pages 11-20 and add the watermark. This way, when switching to the next page, the watermark has already been added, there are no rendering issues, and efficiency can be improved.

Specific Code Implementation:

1. Initialize the number of pages to be preloaded each time, such as setting it to 10, and the collection of pages that have been loaded.

```
private static final int PAGING_SIZE = 10;
private final List<Integer> mLoadPageNums = new ArrayList<>();
```

2. Create a task to add the watermark.

   InsertWatermarkTask:

```
private class InsertWatermarkTask extends Task {

    private final PDFViewCtrl mViewCtrl;
    private final int mPageNum;
    private final List<Integer> mInsertedPages;

    private InsertWatermarkTask(PDFViewCtrl viewCtrl, int pageNum) {
        super(new CallBack() {
            @Override
            public void result(Task task) {
                InsertWatermarkTask task1 = (InsertWatermarkTask) task;

                for (int pageIndex : task1.mInsertedPages) {
                    if (task1.mViewCtrl.isPageVisible(pageIndex)) {
                        int pageWidth = task1.mViewCtrl.getPageViewWidth(pageIndex);
                        int pageHeight = task1.mViewCtrl.getPageViewHeight(pageIndex);
```

```
                task1.mViewCtrl.refresh(pageIndex, new Rect(0, 0, pageWidth, pageHeight));
            }
        }
    }
});

this.mViewCtrl = viewCtrl;
this.mPageNum = pageNum;
mInsertedPages = new ArrayList<>();
}

@Override
protected void execute() {
    try {
        int startIndex = mPageNum * PAGING_SIZE;
        int endIndex = Math.min(mViewCtrl.getPageCount(), (mPageNum + 1) * PAGING_SIZE);

        for (int i = startIndex; i < endIndex; i++) {
            PDFPage page = mViewCtrl.getDoc().getPage(i);
            if (!page.isParsed())
                page.startParse(PDFPage.e_ParsePageNormal, null, true);

            if (!page.hasWatermark()) {
                // Insert a  watermark.
                Watermark watermark = getWatermark(mViewCtrl);
                watermark.insertToPage(page);
                mInsertedPages.add(i);
            }
            page.delete();
        }
    } catch (PDFException e) {
        e.printStackTrace();
    }
}
}
```

Here, mPageNum represents the index of the segment, for example, when preloading pages 0-9, mPageNum is 0, when preloading pages 10-19, mPageNum is 1, and so on.

As for mLoadPageNums, just remember this index. Divide the current page number by the default preloading value (PAGING_SIZE) and round it off, then check whether this value is in mLoadPageNums. This way, you can determine whether the watermark has been loaded on the current page, as follows:

```
int pageNum = (int) Math.floor((float) i / PAGING_SIZE);
if (!mLoadPageNums.contains(pageNum)) {
    // If it's not in the added collection, then it's necessary to add a watermark.
    mLoadPageNums.add(pageNum);
```

```
        pdfView.addTask(new InsertWatermarkTask(pdfView, pageNum));
}
```

3. Initialize the watermark we need, and there's no need to create it every time, only create it when it's empty.

```
private WatermarkSettings settings;
private WatermarkTextProperties text_properties;
private Watermark watermark;


private Watermark getWatermark(PDFViewCtrl pdfView) throws PDFException {
        if (settings == null) {
        settings = new WatermarkSettings();
        settings.setFlags(WatermarkSettings.e_FlagOnTop);
        settings.setOffset_x(0);
        settings.setOffset_y(0);
        settings.setOpacity(80);
        settings.setPosition(Constants.e_PosCenter);
        settings.setRotation(45.f);
        settings.setScale_x(1.f);
        settings.setScale_y(1.f);
        }

        if (text_properties == null) {
        text_properties = new WatermarkTextProperties();
        text_properties.setAlignment(Constants.e_AlignmentCenter);
        text_properties.setColor(0xA4A4A4);
        text_properties.setFont_style(WatermarkTextProperties.e_FontStyleNormal);
        text_properties.setLine_space(1);
        text_properties.setFont_size(30.f);
        text_properties.setFont(new Font(e_StdIDTimesB));
        }

        if (watermark == null) {
        watermark = new Watermark(pdfView.getDoc(), WaterMark, text_properties, settings);
        }
        return watermark;
}
```

4. Listen to the page event callback, add watermark.

   PDFViewCtrl.IPageEventListener:

```
pdfView.registerPageEventListener(new PDFViewCtrl.IPageEventListener() {
@Override
public void onPageVisible(int i) {
        int pageNum = (int) Math.floor((float) i / PAGING_SIZE);
        if (!mLoadPageNums.contains(pageNum)) {
        mLoadPageNums.add(pageNum);
```

```
        pdfView.addTask(new InsertWatermarkTask(pdfView, pageNum));
        }
}

@Override
public void onPageInvisible(int i) {

        }

@Override
public void onPageChanged(int i, int i1) {
        }

@Override
public void onPageJumped() {
        }

@Override
public void onPagesWillRemove(int[] ints) {
        }

@Override
public void onPageWillMove(int i, int i1) {
        }

@Override
public void onPagesWillRotate(int[] ints, int i) {
        }

@Override
public void onPagesRemoved(boolean b, int[] ints) {
        }

@Override
public void onPageMoved(boolean b, int i, int i1) {
        }

@Override
public void onPagesRotated(boolean b, int[] ints, int i) {
        }

@Override
public void onPagesInserted(boolean b, int i, int[] ints) {
        }

@Override
public void onPagesWillInsert(int i, int[] ints) {
        }
});
```

At this point, the watermark addition is complete.

The complete code is as follows:

```
package com.example.pdfdemo;

import android.Manifest;
import android.content.Context;
import android.graphics.Rect;
import android.os.Bundle;
import android.widget.Toast;

import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.Task;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Font;
import com.foxit.sdk.common.Library;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.Watermark;
import com.foxit.sdk.pdf.WatermarkSettings;
import com.foxit.sdk.pdf.WatermarkTextProperties;
import com.hjq.permissions.OnPermissionCallback;
import com.hjq.permissions.XXPermissions;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import androidx.appcompat.app.AppCompatActivity;

import static com.foxit.sdk.common.Font.e_StdIDTimesB;

public class MainActivity extends AppCompatActivity {

    private static final String SN = " ";
    private static final String KEY = " ";
    private static final String WaterMark = "I'm a watermark, I'm a watermark";


    private static final int PAGING_SIZE = 10;
    private final List<Integer> mLoadPageNums = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        int errorCode = Library.initialize(SN, KEY);
        if (errorCode != Constants.e_ErrSuccess) {
```

```
        String errorMsg = (errorCode == Constants.e_ErrInvalidLicense) ?
getString(R.string.fx_the_license_is_invalid) : getString(R.string.fx_failed_to_initialize_the_library);
        Toast.makeText(this, errorMsg, Toast.LENGTH_SHORT).show();
        finish();
        return;
    }
    PDFViewCtrl pdfView = findViewById(R.id.pdfView);
    XXPermissions.with(this).permission(Manifest.permission.READ_EXTERNAL_STORAGE,
Manifest.permission.WRITE_EXTERNAL_STORAGE).request(new OnPermissionCallback() {
        @Override
        public void onGranted(List<String> permissions, boolean all) {
            if (all) {
                openFile(pdfView);
            }
        }

        @Override
        public void onDenied(List<String> permissions, boolean never) {

        }
    });

  }

  private void openFile(PDFViewCtrl pdfView) {
    String assetsCacheFile = getAssetsCacheFile(this, "developer_guide_android_CN.pdf");
    pdfView.openDoc(assetsCacheFile, null);
    pdfView.registerPageEventListener(new PDFViewCtrl.IPageEventListener() {
        @Override
        public void onPageVisible(int i) {
            int pageNum = (int) Math.floor((float) i / PAGING_SIZE);
            if (!mLoadPageNums.contains(pageNum)) {
                mLoadPageNums.add(pageNum);

                pdfView.addTask(new InsertWatermarkTask(pdfView, pageNum));
            }
        }

        @Override
        public void onPageInvisible(int i) {

        }

        @Override
        public void onPageChanged(int i, int i1) {
        }

        @Override
        public void onPageJumped() {
        }
```

```
        @Override
        public void onPagesWillRemove(int[] ints) {
        }

        @Override
        public void onPageWillMove(int i, int i1) {
        }

        @Override
        public void onPagesWillRotate(int[] ints, int i) {
        }

        @Override
        public void onPagesRemoved(boolean b, int[] ints) {
        }

        @Override
        public void onPageMoved(boolean b, int i, int i1) {
        }

        @Override
        public void onPagesRotated(boolean b, int[] ints, int i) {
        }

        @Override
        public void onPagesInserted(boolean b, int i, int[] ints) {
        }

        @Override
        public void onPagesWillInsert(int i, int[] ints) {
        }
    });
}

private class InsertWatermarkTask extends Task {

    private final PDFViewCtrl mViewCtrl;
    private final int mPageNum;
    private final List<Integer> mInsertedPages;

    private InsertWatermarkTask(PDFViewCtrl viewCtrl, int pageNum) {
        super(new CallBack() {
            @Override
            public void result(Task task) {
                InsertWatermarkTask task1 = (InsertWatermarkTask) task;

                for (int pageIndex : task1.mInsertedPages) {
                    if (task1.mViewCtrl.isPageVisible(pageIndex)) {
                        int pageWidth = task1.mViewCtrl.getPageViewWidth(pageIndex);
                        int pageHeight = task1.mViewCtrl.getPageViewHeight(pageIndex);
                        task1.mViewCtrl.refresh(pageIndex, new Rect(0, 0, pageWidth, pageHeight));
```

```
                }
            }
        }
    });

    this.mViewCtrl = viewCtrl;
    this.mPageNum = pageNum;
    mInsertedPages = new ArrayList<>();
  }

  @Override
  protected void execute() {
    try {
      int startIndex = mPageNum * PAGING_SIZE;
      int endIndex = Math.min(mViewCtrl.getPageCount(), (mPageNum + 1) * PAGING_SIZE);

      for (int i = startIndex; i < endIndex; i++) {
        PDFPage page = mViewCtrl.getDoc().getPage(i);
        if (!page.isParsed())
          page.startParse(PDFPage.e_ParsePageNormal, null, true);

        if (!page.hasWatermark()) {
          // Insert a  watermark.
          Watermark watermark = getWatermark(mViewCtrl);
          watermark.insertToPage(page);
          mInsertedPages.add(i);
        }
        page.delete();
      }
    } catch (PDFException e) {
      e.printStackTrace();
    }
  }
}

public String getAssetsCacheFile(Context context, String fileName) {
  File cacheFile = new File(context.getCacheDir(), fileName);
  try {
    try (InputStream inputStream = context.getAssets().open(fileName)) {
      try (FileOutputStream outputStream = new FileOutputStream(cacheFile)) {
        byte[] buf = new byte[1024];
        int len;
        while ((len = inputStream.read(buf)) > 0) {
          outputStream.write(buf, 0, len);
        }
      }
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
  return cacheFile.getAbsolutePath();
```

```
    }


    private WatermarkSettings settings;
    private WatermarkTextProperties text_properties;
    private Watermark watermark;


    private Watermark getWatermark(PDFViewCtrl pdfView) throws PDFException {
        if (settings == null) {
            settings = new WatermarkSettings();
            settings.setFlags(WatermarkSettings.e_FlagOnTop);
            settings.setOffset_x(0);
            settings.setOffset_y(0);
            settings.setOpacity(80);
            settings.setPosition(Constants.e_PosCenter);
            settings.setRotation(45.f);
            settings.setScale_x(1.f);
            settings.setScale_y(1.f);
        }

        if (text_properties == null) {
            text_properties = new WatermarkTextProperties();
            text_properties.setAlignment(Constants.e_AlignmentCenter);
            text_properties.setColor(0xA4A4A4);
            text_properties.setFont_style(WatermarkTextProperties.e_FontStyleNormal);
            text_properties.setLine_space(1);
            text_properties.setFont_size(30.f);
            text_properties.setFont(new Font(e_StdIDTimesB));
        }

        if (watermark == null) {
            watermark = new Watermark(pdfView.getDoc(), WaterMark, text_properties, settings);
        }
        return watermark;
    }
}
```

11 Technical Support

Foxit Support

In order to provide you with a more personalized support for a resolution, please log in to your Foxit account and submit a ticket so that we can collect details about your issue. We will work to get your problem solved as quickly as we can once your ticket is routed to our support team.

You can also check out our Support Center, choose Foxit PDF SDK which also has a lot of helpful articles that may help with solving your issue.

Phone Support

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