TABLE OF CONTENTS

1 Introduction to Foxit MobilePDF SDK

1.1 Why Foxit MobilePDF SDK is your choice

1.2 Foxit MobilePDF SDK

1.3 Key Features

1.4 Evaluation

1.5 License

1.6 About this Guide

2 Getting Started

2.1 System Requirements

2.2 What is in the Package

2.3 How to run a demo

2.3.1 Function demo

2.3.2 Viewer control demo

2.3.3 Complete PDF viewer demo

3 Rapidly building a full-featured PDF Reader

3.1 Create a new Android project

3.2 Integrate Foxit MobilePDF SDK into your apps

3.3 Initialize Foxit MobilePDF SDK

3.4 Display a PDF document using PDFViewCtrl

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

4 Customizing the UI

4.1 About configuration file

4.1.1 Two formats of the configuration file

4.1.2 Notice items of configuration file

4.1.3 Instantiate a UIExtensionsManager object with the configuration file

4.2 Customize feature modules and annotations through a configuration file
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Customize UI elements through APIs</td>
<td>49</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Customizing top/bottom bar</td>
<td>49</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Customizing to show/hide a specific Panel</td>
<td>57</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Customizing to show/hide the UI elements in the View setting bar</td>
<td>60</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Customizing to show/hide the UI elements in the More Menu view</td>
<td>63</td>
</tr>
<tr>
<td>4.4</td>
<td>Customize UI implementation through source code</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>Creating a custom tool</td>
<td>76</td>
</tr>
<tr>
<td>6</td>
<td>FAQ</td>
<td>88</td>
</tr>
<tr>
<td>6.1</td>
<td>Open a PDF document from a specified PDF file path</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Display a specified page when opening a PDF document</td>
<td>89</td>
</tr>
<tr>
<td>6.3</td>
<td>Add a link annotation to a PDF file</td>
<td>90</td>
</tr>
<tr>
<td>6.4</td>
<td>Insert an image into a PDF file</td>
<td>90</td>
</tr>
<tr>
<td>6.5</td>
<td>License key and serial number cannot work</td>
<td>92</td>
</tr>
<tr>
<td>6.6</td>
<td>Highlight the links in PDF documents and set the highlight color</td>
<td>92</td>
</tr>
<tr>
<td>6.7</td>
<td>Highlight the form fields in PDF form files and set the highlight color</td>
<td>93</td>
</tr>
<tr>
<td>6.8</td>
<td>Indexed Full Text Search support</td>
<td>93</td>
</tr>
<tr>
<td>6.9</td>
<td>Print PDF document</td>
<td>95</td>
</tr>
<tr>
<td>6.10</td>
<td>Night mode color settings</td>
<td>96</td>
</tr>
<tr>
<td>7</td>
<td>Technical Support</td>
<td>97</td>
</tr>
</tbody>
</table>
1 Introduction to Foxit MobilePDF SDK

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.

1.1 Why Foxit MobilePDF SDK is your choice

Foxit is an Amazon-invested 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 MobilePDF SDK is a new SDK product which is developed for providing quick PDF viewing and manipulation support for mobile platforms. Customers choose it for the following reasons:

- Easy to integrate
  Developers can seamlessly integrate Foxit MobilePDF SDK into their own apps with just a few lines of code.

- Perfectly designed
  Foxit MobilePDF SDK is designed with a simple, clean, and friendly style, which provides the best user experience.

- Flexible customization
  Foxit MobilePDF SDK 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 MobilePDF SDK 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 Foxit MobilePDF 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 Foxit MobilePDF SDK

Foxit MobilePDF SDK is a Rapid Development Kit for mobile platforms which focuses on helping developers easily integrate powerful Foxit PDF technology into their own apps. With Foxit MobilePDF SDK, even developers with a limited knowledge of PDF can quickly build a professional PDF viewer with just a few lines of code on iOS, Android platforms and Universal Windows Platform (UWP) for Windows 10.

Foxit MobilePDF SDK consists of three elements as shown in the following picture.

---

**The three elements for Foxit MobilePDF SDK**

---

• **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 (e.g. UIView on iOS and Android.View.ViewGroup on Android) 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, reflow, 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 MobilePDF SDK makes a big change and optimization for the UI Extensions Component. 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 MobilePDF SDK removes the PDFReader class, and move the wrapped APIs in PDFReader class to UI Extensions Component.

**1.3 Key Features**

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

PDF Document Open and close files, set and get metadata.

PDF Page Parse, render, read, and 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 Search text in a PDF document, and provide indexed Full-Text Search

Text Search Search text in a PDF document.

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.

Signature Sign a PDF document, verify a signature, add or delete a signature field.

Security Add and verify third-party digital signature.

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.

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 MobilePDF SDK

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 MobilePDF SDK provides an out-of-memory (OOM) mechanism to support applications.
OOM is an evolved feature in Foxit MobilePDF SDK because of its complexity. The key of OOM mechanism is that Foxit MobilePDF SDK will monitor the usage of memory and take recovery operations automatically once OOM is detected. During the recovery process, Foxit MobilePDF SDK 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 MobilePDF SDK provides a property "shouldRecover" in FSPDFViewCtrl class. By default, the value of "shouldRecover" is "YES". If you do not want to enable the auto-recovery when OOM is detected, you can set "shouldRecover" to "false" as follows:

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

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

1.4 Evaluation

Foxit MobilePDF 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 21-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 MobilePDF SDK.

1.5 License

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

1.6 About this Guide

Foxit MobilePDF SDK is currently available on iOS, Android and UWP for Windows 10 platforms. This guide is intended for the developers who need to integrate Foxit MobilePDF SDK for Android into their own apps. It aims at introducing the following sections:

- Section 1: gives an introduction of Foxit MobilePDF 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 create a custom tool.
• Section 6: lists some frequently asked questions.
• Section 7: provides support information.
2 Getting Started

It is very easy to setup Foxit MobilePDF SDK 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 4.1 (API 16)
- 32/64-bit ARM (armeabi-v7a/arm64-v8a) or 32-bit Intel x86 CPU

Android Studio 3.0 or newer

The runtime environment for our demos:

- Android Studio 3.0
- JDK 1.8
- Gradle Version 3.2
- Gradle Build Tool 2.2.3

2.2 What is in the Package

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

- **docs**: A folder containing API references, developer guide.
- **libs**: A folder containing license files, .so libraries, Jar, AAR files, and UI Extensions Component source code.
- **samples**: A folder containing Android sample projects.
- **getting_started_android.pdf**: A quick guide for Foxit MobilePDF SDK for Android.
- **legal.txt**: Legal and copyright information.
- **release_notes.txt**: Release information.
In the "libs" folder as shown in Figure 2-2, there are items that make up the core components of Foxit MobilePDF SDK for Android.

- **librdk.so** (libs/armeabi-v7a, libs/arm64-v8a, libs/x86) - called by the Java APIs in the FoxitRDK.jar. It is the heart of the SDK and contains the core functionalities of Foxit MobilePDF SDK, and it is built separately for each architecture. Currently it is available for armeabi-v7a, arm64-v8a, and x86.

- **FoxitRDK.jar** - used by the Java platform. It includes all the Java APIs of Foxit MobilePDF SDK.

- **FoxitRDKUIExtensions.aar** - generated by the "uiextensions_src" project found in the "libs" folder. It includes the FoxitRDK.jar, 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.
· **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.

**Note:** In order to reduce the size of FoxitDKUIExtensions.aar, Foxit MobilePDF SDK 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](https://developer.android.com/studio/build/shrink-code.html).

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

### 2.3 How to run a demo


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

![Figure 2-3](image)

#### 2.3.1 Function demo

The function demo is provided to show how to use Foxit MobilePDF SDK 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 to a PDF page.
- **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.

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). Open the Android Device Monitor, select the device or emulator that has just been launched and create a new folder named "input_files" under the "mnt/sdcard"(a soft link), then you can find the created folder under the "storage/sdcard". Push the five test files found in the "samples/test_files" folder into this folder. In this section, an AVD targeting 6.0 will be used as an example. Then, the added PDF files will be displayed in "storage/sdcard/input_files" (see Figure 2-4).

**Note** For some Android device, maybe you can only find sdcard0, or sdcard1, such as these. The name of the sdcard is not important, just make sure you have created an "input_files" folder in your device’s storage card and pushed the test files into this folder.

![Figure 2-4](image-url)
c) Click on "Run -> Run 'app'" to run the demo. After building the demo successfully, the features are listed like Figure 2-5.

Note: when running the demo, if you encounter an error like "Error running app: Instant Run requires 'Tools | Android | Enable ADB integration' to be enabled", just follow the prompt to enable the ADB integration in "Tools -> Android -> Enable ADB integration".

![Figure 2-5](image)

Figure 2-5

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 2-6. It shows where the text file was saved to. Just run the demo and try the features.
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 MobilePDF SDK.

To run the demo in Android Studio, please refer to the setup steps outlined in the [Function demo](#).

This demo will use the "Sample.pdf" (found in the "samples/test_files" folder) as the test file, 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 2-7 shows what the demo looks like after it was built successfully. Here, an AVD targeting 6.0 will be used as an example to run the demo.

![Figure 2-7]

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 2-8.
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 2-9. Try using the other features to see it in action.
2.3.3 Complete PDF viewer demo

The complete PDF viewer demo demonstrates how to use Foxit MobilePDF SDK 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 support viewing multiple PDF documents. This demo utilizes all of the features and built-in UI implementations which are provided in Foxit MobilePDF SDK.

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

Here, an AVD targeting 4.4.2 will also 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 2-10).
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.

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 2-11. Now, you can browse the two PDF documents by switching the tabs.
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 View menu, choose the Thumbnail as shown in Figure 2-12, and then the thumbnail of the document will be displayed as shown in Figure 2-13.
Figure 2-12
Figure 2-13
3 Rapidly building a full-featured PDF Reader

Foxit MobilePDF SDK 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 MobilePDF SDK 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 MobilePDF SDK. It includes the following steps:

- Create a new Android project
- Integrate Foxit MobilePDF SDK into your apps
- Initialize Foxit MobilePDF SDK
- Display a PDF document using PDFViewCtrl
- Build a full-featured PDF Reader with UI Extensions Component

3.1 Create a new Android project

In this guide, we use Android Studio 3.0 along with Android API revision 23.

Open Android Studio, choose File -> New -> New Project... to start the Create New Project wizard, and then fill the New Project dialog as shown in Figure 3-1. After filling, click Next.
In the Target Android Devices dialog, use the default settings as shown in Figure 3-2. Then, click Next.
Figure 3-2
In the **Add an activity to Mobile** dialog, select "Empty Activity" (for some other Android Studio versions, it might be "Blank Activity") as shown in Figure 3-3, and then click **Next**.

![Figure 3-3](image)

**Figure 3-3**
In the **Customize the Activity** dialog, customize your activity as desired, here, we use the default settings as shown in Figure 3-4, and then click **Finish**.

![Customize the Activity dialog](image)

Figure 3-4

### 3.2 Integrate Foxit MobilePDF SDK 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.

- **FoxitRDKUIExtensions.aar** - generated by the "uiextensions_src" project found in the "libs" folder. It includes the FoxitRDK.jar, built-in UI implementations, 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.

- **librdk.so** (libs/armeabi-v7a, libs/arm64-v8a, libs/x86) - called by Java APIs in the FoxitRDK.jar. It is the heart of the SDK containing the core functionalities of Foxit MobilePDF SDK, and built separately for each architecture. Currently it is available for armeabi-v7a, arm64-v8a, and x86.

To add the above two files into **PDFReader** project, please switch to the "Project" view panel and then follow the steps below:
a) Copy and paste the "FoxitRDKUIExtensions.aar" file from the "libs" folder of the download package to "PDFReader\app\libs" folder.

b) Copy and paste the x86, arm64-v8a, and armeabi-v7a folder under the "libs" folder of the download package to "PDFReader\app\libs" folder.

Note: For this project, we add all of the "so" libraries, the linker can distinguish the architecture of the device you want to use. If you will run the project on an emulator with x86 architecture, you can only add the "x86/librdk.so" library into the project.

Then, the project should look like Figure 3-5.

![Figure 3-5](image)

Then, the project should look like Figure 3-5.

c) Add a reference to "librdk.so". Inside the app’s "build.gradle" file, add the following configuration:

```
build.gradle:
```
Include Foxit MobilePDF SDK as a dependency in the project by setting up the App module in the "build.gradle" file as follows:

First, define the "libs" directory as a repository in the app’s "build.gradle" repositories section.

build.gradle:

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

And then, add "FoxitRDKUIExtensions.aar" to the dependencies. For simplicity, update the dependencies as follows:

```gradle
dependencies {
    compile ('FoxitRDKUIExtensions', ext:'aar')
    compile 'com.android.support:design:25.3.1'
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.0-alpha7'
    testCompile 'junit:junit:4.12'
}
```

**Note:** Foxit MobilePDF SDK has a dependency on recyclerview support library, so you should add it to the dependencies. In this project, we add 'com.android.support:design:25.3.1' which has already included the recyclerview package to the dependencies. Or you can add "compile 'com.android.support:recyclerview-v7:25.3.1'" directly.

After setting the app’s "build.gradle" file, sync it, then the "FoxitRDKUIExtensions" and "recyclerview" packages will appear in **External Libraries** as shown in Figure 3-6.
The following code shows "build.gradle" in its entirety.

build.gradle:

```gradle
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"
    defaultConfig {
        applicationId "com.foxit.pdfreader"
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    sourceSets {
        main {
            jniLibs.srcDirs = ['libs']
        }
    }
}
```

Figure 3-6
Note So far, we set the compileSdkVersion and targetSdkVersion to API 25. If you also want to use API 25, please make sure you have already installed the SDK Platform Android 7.1.1, API 25. If you have not already done this, open the Android SDK Manager to download and install it first.

### Initialize Foxit MobilePDF SDK

It is necessary for apps to initialize and unlock Foxit MobilePDF SDK using a license before calling any APIs. The function `Library.initialize(sn, key)` is provided to initialize Foxit MobilePDF SDK. 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 where to include this code in the PDFReader project.

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

static {
    System.loadLibrary("rdk");
}
...
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 MobilePDF SDK libraries to the PDFReader project, and finished the initialization of the Foxit MobilePDF 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.

```java
import com.foxit.sdk.PDFViewCtrl;
...
private PDFViewCtrl pdfViewCtrl = null;
...
pdfViewCtrl = new PDFViewCtrl(this);
String path = "/mnt/sdcard/input_files/Sample.pdf";
pdfViewCtrl.openDoc(path, null);
setContentView(pdfViewCtrl);
```

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

Update MainActivity.java as follows:

```java
package com.foxit.pdfreader;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import com.foxit.sdk.PDFViewCtrl1;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.Library;

public class MainActivity extends AppCompatActivity {

    private PDFViewCtrl1 pdfViewCtrl1 = null;

    // Load "librdk.so" library.
```
static {
    System.loadLibrary("rdk");
}

// 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 = "/mnt/sdcard/input_files/Sample.pdf";
    pdfViewCtrl.openDoc(path, null);
    setContentView(pdfViewCtrl);
}

b) Set permissions to write and read the SD card of the Android devices or emulators.

Set the "users-permission" in the "AndroidManifest.xml" found in the "app/src/main" to give the project permission to write and read the SD card of the Android devices or emulators.

Update the AndroidManifest.xml as follows:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.foxit.pdfreader">

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```
Note: If you want to run this project on an Android 6.0 (API 23) or higher devices/emulators, you need to write some additional code to require the authorization of runtime permissions, or you can change the targetSdkVersion in app’s "build.gradle" from 25 to the SDK version that is less than 23, such as 21.

The following code snippets can be used as a reference for you to require the authorization of runtime permissions. For more details, you can refer to the MainActivity.java file in the Complete PDF viewer demo.

```java
// Check whether the device or emulator has the permission to access the external storage.
if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
    // If has no permission, then request the permission.
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, REQUEST_CODE);
} else {
    viewDoc();
}

// Define a request code.
private static final int REQUEST_CODE = 1;

// Handle the request after user responds to the request in the dialog.
@override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
    switch (requestCode) {
        case REQUEST_CODE: {
            // 1. If the request is cancelled, the result array is empty.
            // 2. The permission was granted.
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // Do the task you need to do.
            } else {
```
Fantastic! We have now finished building a simple Android app which uses Foxit MobilePDF SDK to display a PDF document with just a few lines of code. The next step is to run the project on a device or emulator.

**In this section**, we build and run the project on an AVD targeting Android 6.0 (API 23), and for simplicity, we just change the `targetSdkVersion` in app’s “build.gradle” from 25 to 21 without needing to write additional code to require the authorization of runtime permissions.

After building, you will see that the "Sample.pdf" document is displayed as shown in Figure 3-7. Now, this sample app has some basic PDF features, such as zooming in/out and page turning. Just have a try!

Figure 3-7
3.5 Build a full-featured PDF Reader with UI Extensions Component

Foxit MobilePDF SDK comes with built-in UI design including the basic UI for app and the feature modules UI, which are implemented using Foxit MobilePDF SDK 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.

   ```java
   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);
   ```

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

   ```java
   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 reader a PDF document, and set the content view.**

   Call UIExtensionsManager.openDocument() function to open and reader a PDF document instead of calling PDFViewCtrl.openDoc() function.
Update MainActivity.java as follows:

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

```java
package com.foxit.pdfreader;

import android.content.res.Configuration;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.Window;
import android.view.WindowManager;
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;

    // Load "librdk.so" library.
    static {
        System.loadLibrary("rdk");
    }

    // 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);

        // 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);

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

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

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

// Open and Reader a PDF document.
String path = "/mnt/sdcard/input_files/Sample.pdf";
uiExtensionsManager.openDocument(path, null);
setContentView(uiExtensionsManager.getContentView());

@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

Update the AndroidManifest.xml file found in "app\src\main" based on the previous section "Display a PDF document using PDFViewCtrl".

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

Add <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" /> and "<service android:name="com.foxit.uiextensions.modules.panzoom.floatwindow.service.FloatWindowService"/>", which grants your phone the floating window permissions. If you do not add this permission and service, the Pan and Zoom feature will not be able to work.

Update AndroidManifest.xml as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.foxit.pdfreader">
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.CAMERA" />
  <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:theme="@style/AppTheme">
    <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>
    <service android:name="com.foxit.uiextensions.modules.panzoom.floatwindow.service.FloatWindowService" />
  </application>
</manifest>
```
Run the project

In this section, we build and run the project on an AVD targeting Android 6.0 (API 23), and you will see that the "Sample.pdf" document is displayed as shown in Figure 3-8. Up to now, it is a full-featured PDF Reader which includes all of the features in Complete PDF viewer demo. Feel free to try it.

![Figure 3-8](image-url)
4 Customizing the UI

Foxit MobilePDF SDK 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 MobilePDF SDK 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 bar, the items in the top bar, and the items in the View setting bar and More Menu view.

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

4.1 About configuration file

From version 4.0, Foxit MobilePDF SDK provides a more convenient way to flexibly control and customize the features through a configuration file. Developers can easily choose the features that they want without needing to write any additional code or redesign the app's UI.

4.1.1 Two formats of the configuration file

The configuration file can be provided as a JSON file or implemented directly in code. It controls which feature modules are enabled. Now, we will introduce the two formats of the configuration file respectively.

The first format, use a JSON file (e.g. name "uiextensions_config.json") which looks like as follows:

```json
{
    "modules": {
        "readingbookmark": true,
        "outline": true,
        "annotations": true,
        "thumbnail": true,
    }
}
```
"attachment": true,
"signature": true,
"search": true,
"pageNavigation": true,
"form": true,
"selection": true,
"encryption": true
}
}

Note: For a full-featured PDF Reader, all of the items in "modules" should be set to "true", which ensures that all of the features are enabled.

Assume that you have put the JSON configuration file to "PDFReader\app\src\main\res\raw" (note that you need to create the "raw" folder by yourself).

Then, use the following code to get the configuration file:

```java
import java.io.InputStream;
...
InputStream stream = this.getApplicationContext().getResources().openRawResource(R.raw.uiextensions_config);
```

The second format, implement it in the code instead of a JSON file as follows:

```java
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.nio.charset.Charset;
...
String UIExtensionsConfig = "\n" \"modules\": {\n"readingbookmark\": true,\n"outline\": true,\n"annotations\": true,\n"thumbnail\": true,\n"attachment\": true,\n"signature\": true,\n"search\": true,\n"pageNavigation\": true,\n"form\": true,\n"selection\": true,\n"encryption\": true\n}\n"\n};
```
InputStream stream = new ByteArrayInputStream(UIExtensionsConfig.getBytes(Charset.forName("UTF-8")));

In this guide, we will use the first format (JSON file) to present the configuration file.

4.1.1 Notice items of configuration file

The JSON configuration file can be written by yourself. There are some situations needed to notice when you are writing the configuration file. Following lists some typical examples:

1) All of the items in "modules" are set to "true", in this case, all of the feature modules will be enabled.

{  
   "modules": {  
      "readingbookmark": true,
      "outline": true,
      "annotations": true,
      "thumbnail": true,
      "attachment": true,
      "signature": true,
      "search": true,
      "pageNavigation": true,
      "form": true,
      "selection": true,
      "encryption": true
   }
}

2) Some items in "modules" are set to "false". For example, "annotations" and "thumbnail" modules are set to "false", in this case, only the "annotations" and "thumbnail" feature modules are disabled, the others are all enabled.

{  
   "modules": {  
      "readingbookmark": true,
      "outline": true,
      "annotations": false,
      "thumbnail": false,
      "attachment": true,
      "signature": true,
      "search": true,
      "pageNavigation": true,
      "form": true,
      "selection": true,
      "encryption": true
   }
}
3) Some items in "modules" are not set. For example, "annotations" and "thumbnail" modules are not set in the configuration file, in this case, the "annotations" and "thumbnail" feature modules are enabled, because the default settings are "true". It means if some features modules are not in the configuration file, they will also be available.

```json
{
  "modules": {
    "readingbookmark": true,
    "outline": true,
    "attachment": true,
    "signature": true,
    "search": true,
    "pageNavigation": true,
    "form": true,
    "selection": true,
    "encryption": true
  }
}
```

From version 5.0, Foxit MobilePDF SDK supports customizing the annotations in the built-in Annotation setting bar through a configuration file, so that developers can choose the annotations that they want without needing to write any additional code. Rewrite the JSON file as follows:

```json
{
  "modules": {
    "readingbookmark": true,
    "outline": true,
    // "annotations":true,
    "annotations": {
      "highlight": true,
      "underline": true,
      "squiggly": true,
      "strikeout": true,
      "inserttext": true,
      "replacetext": true,
      "line": true,
      "rectangle": true,
      "oval": true,
      "arrow": true,
      "pencil": true,
      "eraser": true,
      "typewriter": true,
    }
  }
}
```
Please comment out "annotations" : true,", and set the types of annotations that you want to "true". If you want to remove a specific type of the annotations, just set it to "false".

Note: If some types of annotations are not in the configuration file, they are also enabled, because the default setting is "true". For example, if you comment out "highlight" : true,", it is still available.

4.1.2 Instantiate a UIExtensionsManager object with the configuration file

In section 3.5, we have already introduced how to initialize 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 initialize UIExtensionsManager that uses the configuration file, so that developers can easily customize the feature modules or annotations 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":

```java
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);
 UIExtensionsManager.Config config = new UIExtensionsManager.Config(stream);

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

Note: Here, we use a configuration file to initialize the UIExtensions. If you do not want to use
configuration file, please refer to the section 3.5. If you want to implement the configuration file in code
directly instead of a JSON file, please see section 4.1.1.

4.2 Customize feature modules and annotations through a configuration file

From version 4.0, Foxit MobilePDF SDK provides a more convenient way to flexibly control and
customize the features through a configuration file. Developers can easily choose the features that they
want without needing to write any additional code or redesign the app's UI.

From version 5.0, Foxit MobilePDF SDK supports customizing the Annotations tools (except Attachment
tool) in the built-in Annotation setting bar (See Figure 4-1) through a configuration file. For more details
about the configuration file, you can see section 4.1 "About configuration file".

Note: To find the Annotation setting bar, just click Comment at the bottom bar, and then click 😍.
The above picture does not display all of the annotation tools. You can swipe left on the setting bar to see the other annotation tools.

The configuration file looks like as follows:

```json
{
    "modules": {
        "readingbookmark": true,
        "outline": true,
        // "annotations":true,
        "annotations": {
            "highlight": true,
            "underline": true,
            "squiggly": true,
            "strikeout": true,
            "inserttext": true,
            "replacetext": true,
            "line": true,
            "rectangle": true,
            "oval": true,
            "arrow": true,
            "pencil": true,
```
Please note that only the attachement annotation in the Annotation setting bar is not controlled by the subitems in "annotations". "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 "Comment" in the bottom bar, you should set both "annotations" and "attachment" to false.

To customize the UI elements in the Annotation setting bar, please make sure to comment out "annotation": true,". Then set the types of annotations that you want to "true", if you want to remove a specific type of the annotations, just set it to "false".

Note: If all types of the annotations in the configuration file are set to "false", it is equal to "annotations": false,". If some types of annotations are not in the configuration file, they are also enabled, because the default setting is "true". For example, if you comment out "highlight": true,". it is still available.

In this section, we will show you how to customize feature modules and annotations in your project. You will find it is extremely easy! You only need to modify the configuration file. Below you can see an example 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".

**Example:** Remove "readingbookmark" and "search" feature modules, as well as the "highlight" and "stamp" annotations.

Change the JSON file as follows:

```json
{
   "modules": {
      "readingbookmark": false,
      "outline": true,
      "annotations": {
         "highlight": false,
         "underline": true,
         "squiggly": true,
         "strikeout": true,
         "inserttext": true,
         "replacetext": true,
         "line": true,
         "rectangle": true,
         "oval": true,
         "arrow": true,
         "pencil": true,
         "eraser": true,
         "typewriter": true,
         "textbox": true,
         "callout": true,
         "note": true,
         "stamp": false,
         "polygon": true,
         "cloud": true,
         "polyline": true,
         "distance": true,
         "image": true
      },
      "thumbnail": true,
      "attachment": true,
      "signature": true,
      "search": false,
      "pageNavigation": true,
      "form": true,
      "selection": true,
      "encryption": true
   }
}
```
Then, rebuild and run the demo to see the result. Following lists the comparison diagrams:

Before:

![Before Diagram](image1)

After:

![After Diagram](image2)

The "readingbookmark" and "search" feature modules are removed.

Click **Comment** at the bottom bar, and then click 

![Click Comment](image3)

The "highlight" and "stamp" annotations are removed.
4.3 Customize UI elements through APIs

In version 4.0, Foxit MobilePDF SDK supports customizing to show or hide the whole top or bottom bar, and from version 5.0, it provides APIs to customize to show or hide a specific panel, the items in the top bar, 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.

**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.

4.3.1 Customizing top/bottom bar

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

1. Show or hide the top/bottom bar.
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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void enableTopToolbar(boolean isEnabled)</td>
<td>Enable or disable top toolbar.</td>
</tr>
<tr>
<td>void enableBottomToolbar(boolean isEnabled)</td>
<td>Enable or disable bottom toolbar.</td>
</tr>
<tr>
<td>boolean addItem(BarName barName, BaseBar.TB_Position gravity, BaseItem item, int index);</td>
<td>Add a custom item to the toolbar. The item will be inserted before any previous element at the specified location.</td>
</tr>
<tr>
<td>boolean addItem(BarName barName, BaseBar.TB_Position gravity, int textId, int resId, int index, IItemClickListener clickListener);</td>
<td>Add a default item to the toolbar. The item is inserted before any previous element at the specified location.</td>
</tr>
<tr>
<td>int getItemCount(BarName barName, BaseBar.TB_Position gravity);</td>
<td>Get the items count by IBarsHandler.BarName and BaseBar.TB_Position.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>boolean <code>removeItem</code> (BarName barName,</td>
<td>Remove the item by IBarsHandler.BarName,</td>
</tr>
<tr>
<td>BaseBar.TB_Position gravity, int index);</td>
<td>BaseBar.TB_Position and the specified index.</td>
</tr>
<tr>
<td>boolean <code>removeItem</code> (BarName barName,</td>
<td>Removes the item by IBarsHandler.BarName,</td>
</tr>
<tr>
<td>BaseBar.TB_Position gravity, BaseItem item);</td>
<td>BaseBar.TB_Position and the specified item.</td>
</tr>
<tr>
<td>void <code>removeAllItems</code> (BarName barName);</td>
<td>Remove all items from the toolbar.</td>
</tr>
<tr>
<td>boolean <code>addCustomToolBar</code> (BarName barName, View view);</td>
<td>Add custom toolbar by BarName.</td>
</tr>
<tr>
<td>boolean <code>removeToolBar</code> (BarName barName);</td>
<td>Remove toolbar by BarName.</td>
</tr>
<tr>
<td>void <code>setBackgroundColor</code> (BarName barName, int color);</td>
<td>Set background color for the toolbar.</td>
</tr>
<tr>
<td>void <code>setBackgroundResource</code> (BarName barName, int resid);</td>
<td>Set background to a given resource.</td>
</tr>
<tr>
<td>BaseItem <code>getItem</code> (BarName barName, BaseBar.TB_Position</td>
<td>Get the item by tag, but if you remove this item before you get it, it will return null.</td>
</tr>
<tr>
<td>gravity, int tag);</td>
<td></td>
</tr>
</tbody>
</table>

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.

```java
enum BarName {
    TOP_BAR,
    BOTTOM_BAR;
}

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

**Note:**

1. The top bar or bottom bar can only add up to 7 items.

2. To add an item to the top bar, please set the `BaseBar.TB_Position` to `Position_LT` or `Position_RB`. To add an item to the bottom bar, please set the `BaseBar.TB_Position` to `Position_CENTER`. Otherwise, the items may overlap.

3. The bottom bar is only one part, and the top bar is divided into two parts, so that there are three parts for the toolbar, and each part has a separate index. (See Figure 4-2)

4. The top bar can only display a maximum of 15 characters, and the bottom bar can only display a maximum of 8 characters. So, it's better to set the character number of the text within the maximum number. Please note that the character length does not distinguish between Chinese and English.
In the following examples, we will show you how to customize the top/bottom bar 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);").

**Example1:** Hide the whole top bar.

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

Before:

![Before](image1)

After:

![After](image2)
Example2: Hide the whole bottom bar.

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

Before:

![List View Comment Sign]

After:

![List View Comment Sign]

Example3: Add an item in the left top bar at the second position.

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

The result after running the demo:

![Left Item]

Example4: Add an item in the right top bar at the first position.

```java
BaseItemImpl mTopItem2 = new BaseItemImpl(getContext);
mTopItem2.setImageResource(R.drawable.annot_fileattachment_selector);
mTopItem2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        UIToast.getInstance(getActivity()).show("Add an item in the right top bar at the first position");
    }
});
```
The result after running the demo:

Example5: Add an item to the bottom bar at the first position.

```java
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
    BaseBar.TB_Position.Position_RB, mTopItem2, 0);
```

The result after running the demo:

Example6: Add an item with custom style to the bottom bar at the second position.

```java
int circleResId = R.drawable.rd_bar_circle_bg_selector;
int textSize = getResources().getDimensionPixelSize(R.dimen.ux_text_height_toolbar);
int textColorResId = R.color.ux_text_color_button_colour;
int interval =
    getResources().getDimensionPixelSize(R.dimen.ux_toolbar_button_icon_text_vert_interval);
CircleItemImpl mSettingBtn = new CircleItemImpl(this.getContext());
mSettingBtn.setImageResource(R.drawable.rd_annot_create_ok_selector);
mSettingBtn.setText("style");
mSettingBtn.setRelation(BaseItemImpl.RELATION_BELOW);
mSettingBtn.setCircleRes(circleResId);
mSettingBtn.setInterval(interval);
mSettingBtn.setTextSize(AppDisplay.getInstance(getContext()).px2dp(textSize));
mSettingBtn.setTextColorResource(textColorResId);
mSettingBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
```
```
UIToast.getInstance(getActivity()).show("Add an item with custom style to the bottom bar 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 bar).

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 bar that you added before).

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

Before: (See Example3)

After:
Example 9: Remove all the items in the bottom bar.

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

Before:

After:

Example 10: Add two items in the left top bar to control to show and hide the "more menu" item.

```java
// Get and save the item that you want to show or hide.
BaseBarManager baseBarManager = (BaseBarManager) mUiExtensionsManager.getBarManager();
final BaseItemImpl moreItem = (BaseItemImpl) baseBarManager.getItem(IBarsHandler.BarName.TOP_BAR, BaseBar.TB_Position.Position_RB, ToolbarItemConfig.ITEM_TOPBAR_MORE_TAG);

// Add a button in the left top bar to hide the "moreItem" item.
BaseItemImpl mTopItem = new BaseItemImpl(getContext());
mTopItem.setImageResource(R.drawable.rd_annot_item_delete_selector);
mTopItem.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, mTopItem, 1);

// Add a button in the left top bar to show the "moreItem" item.
BaseItemImpl mTopItem2 = new BaseItemImpl(getContext());
mTopItem2.setImageResource(R.drawable.annot_reply_item_add_selector);
mTopItem2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Show the "moreItem" item.
        mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR, BaseBar.TB_Position.Position_LT, mTopItem2, 1);
    }
});
```
BaseBar.TB_Position.Position_RB, moreItem, 2); }

mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, mTopItem2, 2);

The result after running the demo, the top bar 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 bar.

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

**Example12:** Add a custom toolbar. (add a custom layout file "test_top_layout")

View topView = View.inflate(getContext(), R.layout.test_top_layout, null);
mUiExtensionsManager.getBarManager().addCustomToolBar(IBarsHandler.BarName.TOP_BAR, topView);
### Customizing to show/hide a specific Panel

To show or hide a specific panel (See Figure 4-3, includes "Reading Bookmarks", "Outline", "Annotations" and "Attachments" panels, just clicks **List** at the bottom bar to find it), you can use the following APIs listed in the Table 4-2.

#### Table 4-2

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void setPanelHidden(boolean isHidden, PanelSpec.PanelType panelType)</code></td>
<td>To show or hide a panel according to the PanelType.</td>
</tr>
<tr>
<td><code>public boolean isHiddenPanel(PanelsSpec.PanelType panelType)</code></td>
<td>Return the current value in <code>setPanelHidden</code>.</td>
</tr>
</tbody>
</table>

**Figure 4-3**

*Note: To show or hide a specific panel through APIs, please mark sure the corresponding features in the configuration file is set to "true". Otherwise, the API settings will not have any effect.*

In this section, we only give an example to show you how to show or hide 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 others panels, you only need to change the `PanelType`. The corresponding relation between panels and `PanelType` are as follows:
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(), pdfViewerCtrl1, config);").

Example: Add an item in the left top bar at the second position to control whether to show or hide the "Outline" panel.

```java
BaseItemImpl mTopItem = new BaseItemImpl(getActivity());
mTopItem.setImageResource(R.drawable.rd_annot_item_delete_selector);
mTopItem.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mUiExtensionsManager.isHiddenPanel(PanelSpec.PanelType.Outline)) {
            mUiExtensionsManager.setPanelHidden(false, PanelSpec.PanelType.Outline);
            UIToast.getInstance(getActivity()).show("show Outline");
        } else {
            mUiExtensionsManager.setPanelHidden(true, PanelSpec.PanelType.Outline);
            UIToast.getInstance(getActivity()).show("hide outline");
        }
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, mTopItem, 1);
```

Here, we add a button in the top bar to try this function. Click the button, if the "Outline" panel exists, then hides it, otherwise shows it.

After running the demo, click the "delete" button, it will pop up "hide Outline" as follows:
Then, tap **List** in the bottom bar, and you will see the "Outline" panel has been hidden (See Figure 4-4).

For Reading Bookmarks, Annotations, and Attachments panels, you only need to change the **PanelType**. Just try it.
4.3.3 Customizing to show/hide the UI elements in the View setting bar

To show or hide the UI elements in the View setting bar (See Figure 4-5, just clicks View at the bottom bar to find it), you only need to use the following API:

```java
public void setVisibility(int type, int visibility)
```

![Figure 4-5](image)

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

<table>
<thead>
<tr>
<th>type</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMultiLineBar.TYPE_LIGHT</td>
<td>1</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_DAYNIGHT</td>
<td>2</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_SYSLIGHT</td>
<td>4</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_SINGLEPAGE</td>
<td>8</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_CONTINUOUSPAGE</td>
<td>16</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_THUMBNNIAL</td>
<td>32</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_LOCKSCREEN</td>
<td>64</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_REFLOW</td>
<td>128</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_CROP</td>
<td>256</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_FACING_MODE</td>
<td>288</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_COVER_MODE</td>
<td>320</td>
</tr>
<tr>
<td>IMultiLineBar.TYPE_PANZOOM</td>
<td>384</td>
</tr>
</tbody>
</table>
The value of the parameter "visibility" can be set as follows:

<table>
<thead>
<tr>
<th>visibility</th>
<th>integer</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View.VISIBLE</td>
<td>0</td>
<td>The view is visible.</td>
</tr>
<tr>
<td>View.INVISIBLE</td>
<td>4</td>
<td>This view is invisible, but it still takes up space for layout purposes.</td>
</tr>
<tr>
<td>View.GONE</td>
<td>8</td>
<td>This view is invisible, and it doesn't take any space for layout.</td>
</tr>
</tbody>
</table>

In this section, we only take "Reflow" item as an example to show you how to show or 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 = new UIExtensionsManager(getActivity().getApplicationContext(), pdfViewerCtrl1, config);").

**Example1: To hide the Reflow item in the View setting bar without changing the layout.**

```java
mUiExtensionsManager.getSettingBar().setVisibility(IMultiLineBar.TYPE_REFLOW, View.INVISIBLE);
```

Before:

![Before](image1)

After:

![After](image2)
Example 2: To hide the Reflow item in the View setting bar with changing the layout.

```java
mUiExtensionsManager.getSettingBar().setVisibility(IMultiLineBar.TYPE_REFLOW, View.GONE);
```

Before:

![Before](image1)

After:

![After](image2)

For other items in the View setting bar, you can refer to the above example, and just need to change the value of the parameter "type" in `setVisibility` interface.

To show one of the UI elements in the View setting bar, just set the value of the parameter "visibility" in `setVisibility` interface to "View.VISIBLE".
4.3.4 Customizing to show/hide the UI elements in the More Menu view

To show or hide the More Menu item, please see section 4.3.1 "Customizing top/bottom bar" (see example 10).

To show or hide the UI elements in the More Menu view (See Figure 4-6, just clicks at the right top bar to find it), you can use the following APIs listed in the Table 4-3.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setGroupVisibility(int visibility, int tag)</td>
<td>Set the enabled state of group according to &quot;tag&quot;.</td>
</tr>
<tr>
<td>void setItemVisibility(int visibility, int groupTag, int itemTag);</td>
<td>Set the enabled state of item according to &quot;groupTag&quot; and &quot;itemTag&quot;.</td>
</tr>
</tbody>
</table>

Figure 4-6
The value of the parameter "tag" in the `setGroupVisibility` interface or the "groupTag" in the `setItemVisibility` interface can be set as follows:

<table>
<thead>
<tr>
<th>tag</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_FILE</td>
<td>100</td>
</tr>
<tr>
<td>GROUP_PROTECT</td>
<td>101</td>
</tr>
<tr>
<td>GROUP_FORM</td>
<td>102</td>
</tr>
<tr>
<td>GROUP_PRINT</td>
<td>103</td>
</tr>
</tbody>
</table>

The value of the parameter "itemTag" in the `setItemVisibility` interface can be set as follows:

<table>
<thead>
<tr>
<th>groupTag</th>
<th>itemTag</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_FILE</td>
<td>ITEM_DOCINFO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ITEM_REDUCE_FILE_SIZE</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ITEM_SNAPSHOT</td>
<td>2</td>
</tr>
<tr>
<td>GROUP_PROTECT</td>
<td>ITEM_PASSWORD</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ITEM_REMOVESECURITY_PASSWORD</td>
<td>4</td>
</tr>
<tr>
<td>GROUP_FORM</td>
<td>ITEM_RESET_FORM</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ITEM_IMPORT_FORM</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ITEM_EXPORT_FORM</td>
<td>2</td>
</tr>
<tr>
<td>GROUP_PRINT</td>
<td>ITEM_PRINT_FILE</td>
<td>0</td>
</tr>
</tbody>
</table>

The value of the parameter "visibility" in the `setGroupVisibility` and `setItemVisibility` interfaces can be set as follows:

<table>
<thead>
<tr>
<th>visibility</th>
<th>integer</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View.VISIBLE</td>
<td>0</td>
<td>The view is visible.</td>
</tr>
<tr>
<td>View.INVISIBLE</td>
<td>4</td>
<td>This view is invisible, but it still takes up space for layout purposes.</td>
</tr>
<tr>
<td>View.GONE</td>
<td>8</td>
<td>This view is invisible, and it doesn’t take any space for layout.</td>
</tr>
</tbody>
</table>

*Note: For `setItemVisibility` interface, if you want to show or hide an `itemTag`, please make sure the corresponding `groupTag` has been set to "View.VISIBLE". Otherwise, the settings will not have any effect.*

In this section, we only take "GROUP_FILE" (File in the view) and "ITEM_DOCINFO" (Properties in the view) as an example to show you how to show or hide the UI elements in the More Menu view through APIs in the "complete_pdf_viewer" demo found in the "samples" folder. For other UI elements, you can refer to the following examples and only need to change the parameter value in the `setGroupVisibility` and `setItemVisibility` interfaces.
Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "m(UIExtensionsManager = new UIExtensionsManager(getActivity().getApplicationContext(), pdfViewerCtrl, config);”).

**Example 1:** Hide the "File" in the More Menu view without changing the layout.

```java
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule)
    m(UIExtensionsManager.getModuleByName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

menuView.setGroupVisibility(View.INVISIBLE, MoreMenuConfig.GROUP_FILE);
```

**Before:**

![Before screenshot]

**After:**

![After screenshot]
Example2: Hide the "File" in the More Menu view with changing the layout.

```java
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule) mUiExtensionsManager.getModuleByName(Module.MODE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

menuView.setGroupVisibility(View.GONE, MoreMenuConfig.GROUP_FILE);
```

Before:
```
More

- File
- Properties
- Reduce File Size
- Snapshot
- Protect
- Password Protect
- Form
- Insert Form Fields
- Import Form Data
- Export Form Data
- Print
- Print File
```

After:
```
More

- Project
- Password Protect
- Note
- Reset Form Fields
- Import Form Data
- Export Form Data
- Page
- Print File
```
**Example 3:** Hide the "Properties" in the More Menu view without changing the layout.

```java
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule)
mUiExtensionsManager.getModuleByModuleName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

menuView.setItemVisibility(View.INVISIBLE, MoreMenuConfig.GROUP_FILE, MoreMenuConfig.ITEM_DOCINFO);
```

Before:

![Before Image]

After:

![After Image]
Example 4: Hide the "Properties" in the More Menu view with changing the layout.

```java
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule) 
mUiExtensionsManager.getModuleByName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule getMenuView();

menuView.setItemVisibility(View.GONE, MoreMenuConfig.GROUP_FILE, MoreMenuConfig.ITEM_DOCINFO);

Before:

<table>
<thead>
<tr>
<th>File</th>
<th>Properties</th>
<th>Reduce File Size</th>
<th>Snapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protect</td>
<td>Password Protect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Form</td>
<td>Reset Form Fields</td>
<td>Import Form Data</td>
<td>Export Form Data</td>
</tr>
<tr>
<td>Print</td>
<td>Print File</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After:

<table>
<thead>
<tr>
<th>File</th>
<th>Reduce File Size</th>
<th>Snapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>Password Protect</td>
<td>Form</td>
</tr>
<tr>
<td>Export Form Data</td>
<td>Import Form Data</td>
<td>Print</td>
</tr>
</tbody>
</table>

For other items in the More Menu view, you can refer to the above examples, and just need to change the parameter value in the setGroupVisibility and setItemVisibility interfaces.

To show one of the UI elements in the More Menu view, just set the value of the parameter "visibility" in setVisibility and setItemVisibility interfaces to "View.VISIBLE".
4.4 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 MobilePDF SDK. 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:

**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.jar** - used by the Java platform. It includes all the Java APIs of Foxit MobilePDF SDK.

- **librdk.so** (libs/armeabi-v7a, libs/arm64-v8a, libs/x86) - called by the Java APIs in the FoxitRDK.jar. It is the heart of the SDK and contains the core functionalities of Foxit MobilePDF SDK, and it is built separately for each architecture. Currently it is available for armeabi-v7a, arm64-v8a, and x86.

  **Note** The **uiextensions_src project has a dependency on FoxitRDK.jar. 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.jar 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.jar and librdk.so files, 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.

```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 4-7.

![Figure 4-7](image)

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

```gradle
dependencies {
    compile 'com.google.android:multidex:0.1'
    compile 'com.android.support:appcompat-v7:23.4.0'
    compile 'com.android.support:design:23.4.0'
    compile project(':uiextensions_src')
    //compile(name:'FoxitRDKUIExtensions', ext:'aar')
}
```

After making the change, rebuild this gradle. Then, select "File -> Project Structure..." to open the Project Structure dialog. In the dialog click "Modules -> app", and select the Dependencies option, then you can see that the demo has a dependency on the uiextensions_src project as shown in Figure 4-8.
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 4-9.
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 4-10.
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 4-11.

After finishing the three steps described in the above picture, go to "search_result_selector.xml" as shown in Figure 4-12. We can see that the icon is stored in the "drawable-xxx" folder with the name of "search_result.png" by holding Ctrl and left-clicking on "search_result". Just replace it with your own icon.
Note Foxit MobilePDF SDK provides three sets of icons for devices with different DPI requirements to make sure that your apps can run smoothly on every device.

For example, we use the icon of the search next button ("search_next.png" stored in the same folder with "search_result.png") 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.
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 Creating a custom tool

With Foxit MobilePDF SDK, 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 MobilePDF SDK. 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 tool manager.
- Set the ScreenCaptureToolHandler object as the current tool handler.

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

```java
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 5-1.
```
Step 2: Handle the `onTouchEvent` and `onDraw` events.

Update `ScreenCaptureToolHandler.java` as follows:

```java
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.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;
```

Figure 5-1
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 (!mStartPoint.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, "/mnt/sdcard/ScreenCapture.bmp");
            Toast.makeText(mContext, "The selected area was saved as a bitmap stored in the /mnt/sdcard/ScreenCapture.bmp", Toast.LENGTH_LONG).show();
        }
}
mDownPoint.set(0, 0);
mLastPageIndex = -1;
return true;
default:
    return true;
}
return true;
}

// Save a bitmap 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 i, MotionEvent motionEvent) {
    return false;
}

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

// Handle the drawing event.
@Override
public void onDraw(int pageIndex, Canvas canvas) {

}
if (((UIExtensionsManager) mPdfViewCtrl.getUIExtensionsManager()).getCurrentToolHandler() != this)
    return;

if (mLastPageIndex != pageIndex) {
    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 6.0. 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 successfully, click anywhere to display the Contextual Action bar, and click ☰
(overflow button) to find the ScreenCapture action button as shown in Figure 5-2.
Figure 5-2
Click **ScreenCapture**, long press and select a rectangular area, and then a message box will be popped up as shown in Figure 5-3. It shows where the bitmap (selected area) was saved to.

![Figure 5-3](image-url)
In order to verify whether the tool captures the selected area successfully, we need to find the screenshot. Open the **Android Device Monitor**, we can see the screenshot named "ScreenCapture.bmp" in the SD card as shown in Figure 5-4.
Pull the ScreenCapture.bmp from the emulator, we can see the image as shown in Figure 5-5.

![Product Overview](www.foxitsoftware.com)

**Figure 5-5**

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 MobilePDF SDK. You can refer to it or our demos to develop the tools you want.
6 FAQ

6.1 Open a PDF document from a specified PDF file path

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

Foxit MobilePDF SDK 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:

```
// Assuming A PDFViewCtrl has been created.
// Open an unencrypted PDF document from a specified PDF file path.
String path = "/mnt/sdcard/input_files/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 = "/mnt/sdcard/input_files/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.
}
```
6.2 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)` is supposed to use. Foxit MobilePDF SDK utilizes multi-thread to improve rendering speed, so you need to 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:

```java
// Assuming A PDFViewCtrl has been created.
// Register the PDF document event listener.
pdfViewCtrl1.registerDocEventListener(docListener);

// Open an unencrypted PDF document from a specified PDF file path.
String path = "/mnt/sdcard/input_files/Sample.pdf";
pdfViewCtrl1.openDoc(path, null);
...

PDFViewCtrl1.IDocEventListener docListener = new PDFViewCtrl1.IDocEventListener() {
    @Override
    public void onDocWillOpen() {}

    @Override
    public void onDocOpened(PDFDoc pdfDoc, int errCode) {
        pdfViewCtrl1.gotoPage(2);
    }

    @Override
    public void onDocWillClose(PDFDoc pdfDoc) {} 

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

    @Override
    public void onDocWillSave(PDFDoc pdfDoc) {} 
};
```
6.3 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 an 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:

```java
private Link linkAnnot = null;
...
String path = "mnt/sdcard/input_files/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 = (Link) page.addAnnot(Annot.e_Link, new RectF(250, 750, 500, 650));

    // Create a URI action and set the URI.
    URIAction uriAction = (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("mnt/sdcard/input_files/sample_annot.pdf", PDFDoc.e_SaveFlagNormal);
} catch (Exception e) {
    e.printStackTrace();
}
```

6.4 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.

```java
String path = "mnt/sdcard/input_files/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("mnt/sdcard/input_files/2.png", new PointF(20, 30), 60, 50, true);
    // Save the document that has added the image.
    document.saveAs("mnt/sdcard/input_files/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:

```java
String path = "mnt/sdcard/input_files/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 = (Stamp) page.addAnnot(Annot.e_annotStamp, new RectF(100, 350, 250, 150));

    // Load a Local image and convert it to a Bitmap.
    Bitmap bitmap = BitmapFactory.decodeFile("mnt/sdcard/input_files/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("mnt/sdcard/input_files/sample_image.pdf", PDFDoc.e_saveFlagNormal);

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

6.5 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 time of your device. If the device's time 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 time of your device.

6.6 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?

UIExtensionsManager.enableLinkHighlight() interface is provided to set whether to enable highlighting the links in PDF documents. By default, the links in PDF documents will be highlighted. 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. Please make sure you have not disabled highlighting links. Following is a sample for calling this API:

// Assume you have already Initialized a UIExtensionsManager object
uiExtensionsManager.setLinkHighlightColor(0x4b0000ff);
6.7 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?

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

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

UIExtensionsManager.setLinkHighlightColor() interface is used to set the highlight color. Please make sure you have not disabled highlighting form fields. Following is a sample for calling this API:

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

6.8 Indexed Full Text Search support

Does Foxit MobilePDF SDK 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 MobilePDF SDK for Android supports Indexed Full Text Search from version 5.0.

To use this feature, follow the steps below:

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

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

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

```java
public FullTextSearch()
public void setDatabasePath(String pathOfDataBase)
```

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

```java
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:

```java
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 database 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:

```java
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 
Page index is:%d Start text
End text index:%d 
Match is:%s 
\n", filePath, pageIndex, 
matchStartTextIndex, matchEndTextIndex, matchResult);
        Log.v(TAG, "retrieveSearchResult: " + s);
        return 0;
    }
}
```

**Note:**

- The indexed full text search provided by Foxit MobilePDF SDK 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 should 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.

### 6.9 Print PDF document

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

Yes. Foxit MobilePDF SDK supports 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:

```java
// 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:

```java
// 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("/mnt/sdcard/input_files/Sample.pdf");
    doc.load(null);
} catch (PDFException e) {
    Assert.fail("unexpected a PDF Exception!! errCode = " + e.getLastError());
}
uExtensionsManager.startPrintJob(getActivity(), doc, "print with name", "print_withAPI",
    print_callback);
```

### 6.10 Night mode color settings

**How can I set the night mode color?**

From version 5.1, Foxit MobilePDF SDK support to set the night mode color as desired.

To set the night mode color, you should call the `PDFViewCtrl.setMappingModeBackgroundColor(int)` and `PDFViewCtrl.setMappingModeForegroundColor(int)` APIs to set the color as you wish, then set the color mode by using `PDFViewCtrl setColorMode(int)`.

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

Following is a sample to set the night mode color:

```java
private UIExtensionsManager uiExtensionsManager = null;
...
PDFViewCtrl pdfViewCtrl = uiExtensionsManager.getPDFViewCtrl();
pdfViewCtrl.setMappingModeBackgroundColor(0xff87cefa);
pdfViewCtrl.setMappingModeForegroundColor(0xff7cfc00);
pdfViewCtrl.setColorMode(Renderer.e_ColorModeMapping);
```
7 Technical Support

Reporting Problems

Foxit offers 24/7 support for its products and are fully supported by the PDF industry’s largest development team of support engineers. If you encounter any technical questions or bug issues when using Foxit MobilePDF SDK, please submit the problem report to the Foxit support team at http://tickets.foxitsoftware.com/create.php. In order to better help you solve the problem, please provide the following information:

- Contact details
- Foxit MobilePDF SDK product and version
- Your Operating System and IDE version
- Detailed description of the problem
- Any other related information, such as log file or error screenshot

Contact Information

You can contact Foxit directly, please use the contact information as follows:

Foxit Support:

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

Sales Contact:

- Phone: 1-866-680-3668
- Email: sales@foxitsoftware.com

Support & General Contact:

- Phone: 1-866-MYFOXIT or 1-866-693-6948
- Email: support@foxitsoftware.com