DEVELOPER GUIDE

Foxit PDF SDK

For Android

Microsoft® Partner
Gold Independent Software Vendor (ISV)

TABLE OF CONTENTS

1 Introduction to Foxit PDF SDK

1.1 Foxit PDF SDK

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

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

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

1.2 Foxit PDF SDK for Android

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

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

1.2.1 Why Foxit PDF SDK for Android is your choice

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

- **Easy to integrate**

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

- **Perfectly designed**

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

- **Flexible customization**

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

- **Robust performance on mobile platforms**

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

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

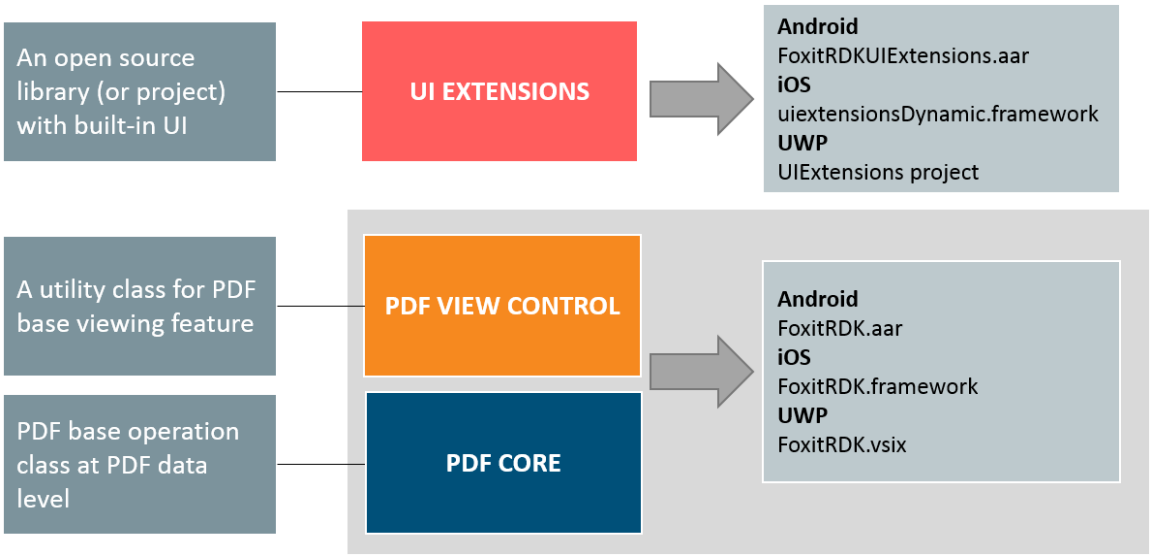
The core technology of the SDK is based on Foxit's PDF engine, which is trusted by a large number of the world's largest and well-known companies. Foxit's powerful engine makes the app fast on parsing, rendering, and makes document viewing consistent on a variety of devices.

- **Premium World-side Support**

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

1.2.2 Main Frame of Foxit PDF SDK for Android

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



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

- **PDF Core API**

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

- **PDF View Control**

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

- **UI Extensions Component**

The UI Extensions **Component** is an open source library that provides a customizable user interface with built-in support for text selection, markup annotation, outline navigation, reading bookmarks, full-text searching, form filling, text reflow, attachment, digital/handwritten signature, document editing and password encryption. These features in the UI Extensions Component are implemented using the PDF Core API and PDF View Control. Developers can utilize these ready-to-use UI implementations to build a

PDF viewer quickly with the added benefit of complete flexibility and control to customize the UI design as desired.

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

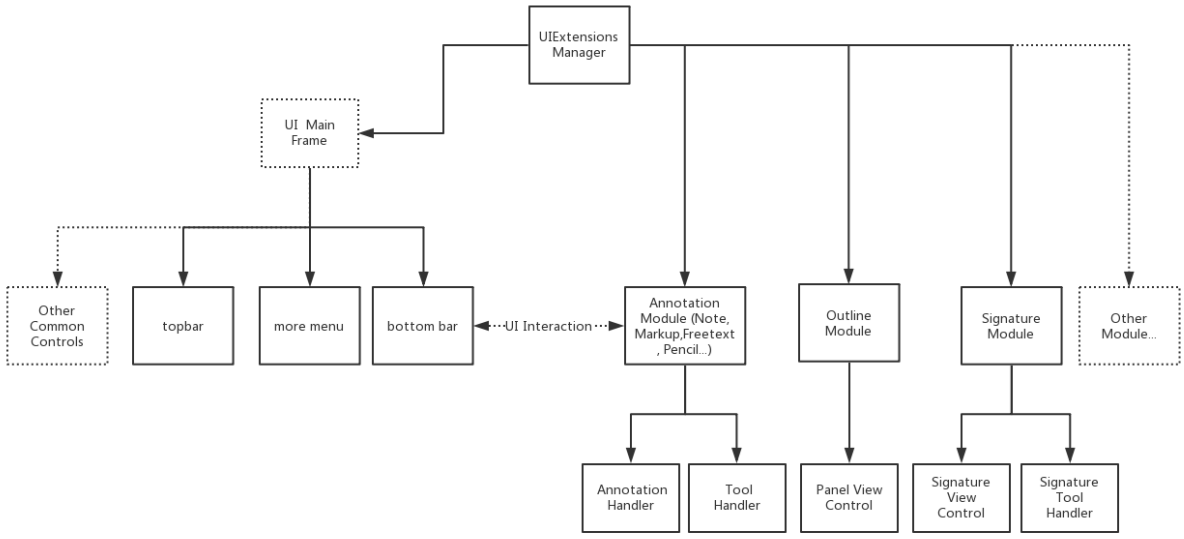
From version 5.0, every element in the built-in UI can be configurable through an API. More advanced APIs and more powerful configuration file are provided for developers to further customize the UI elements, such as adding/removing a button to/from the toolbar, showing/hiding a specific menu/function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

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

1.2.3 UI Extensions Component Overview

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

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



The relationship between UIExtensionsManager and modules

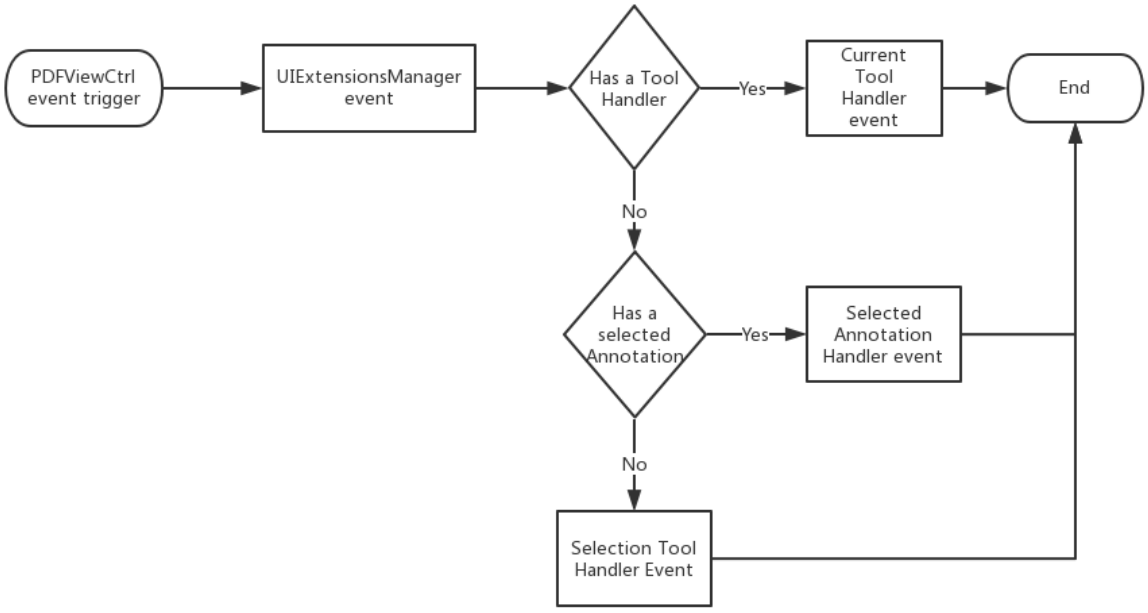
Tool handler and annotation handler will process the events from touch screen or gestures of PDFViewCtrl. When the touch screen and gestures occur, PDFViewCtrl will send the corresponding events to UIExtensionsManager:

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

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

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

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



The event response flow chart between Tool Handler and Annotation Handler

1.2.4 Key Features of Foxit PDF SDK for Android

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

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

Annotation	Create, edit and remove annotations.
Layers	Add, edit, and remove optional content groups.
Attachments	Add, edit, and remove document level attachments.
Form	Fill form with JavaScript support, export and import form data by XFDF/FDF/XML file.
XFA	Support static and dynamic XFA.
Signature	Sign a PDF document, verify a signature, add or delete a signature field. Add and verify third-party digital signature.
Security	Protect PDFs with password or certificate.
Pan and Zoom	Adjust the magnification and position of the view area to match the area in an adjustable rectangle in the Pan & Zoom window's thumbnail view of the page.
Print	Print PDF document.
RMS	Support Microsoft RMS decryption with the standard IRMv1 and IRMv2.
Out of Memory	Recover from an OOM condition.

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

Support robust PDF applications with Foxit PDF SDK for Android

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

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

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

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

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

1.3 Evaluation

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

1.4 License

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

1.5 About this Guide

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

- Section 1: gives an introduction of Foxit PDF SDK, especially for Android platform SDK.
- Section 2: illustrates the package structure and running demos.
- Section 3: describes how to quickly create a full-featured PDF Reader.
- Section 4: introduces how to customize the user interface.
- Section 5: shows how to use Foxit PDF SDK Core API.
- Section 6: shows how to create a custom tool.
- Section 7: shows how to implement Foxit PDF SDK using Cordova
- Section 8: shows how to implement Foxit PDF SDK using React Native
- Section 9: shows how to implement Foxit PDF SDK using Xamarin
- Section 10: lists some frequently asked questions.
- Section 11: provides support information.

2 Getting Started

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

2.1 System Requirements

Android devices' requirements:

- Android 4.1 (API 16) or higher
- 32/64-bit ARM (armeabi-v7a/arm64-v8a) or 32/64-bit Intel x86 CPU

Android Studio 3.0 or newer

The runtime environment for our demos:

- Android Studio 3.1
- JDK 1.8
- Gradle Version 4.4
- Gradle Build Tool 3.1

2.2 What is in the Package

Download the "foxitpdfsdk_6_4_android.zip" package, and extract it to a new directory like "foxitpdfsdk_6_4_android" as shown in Figure 2-1. The package contains:

docs:	A folder containing API references, developer guide, and upgrade warnings.
libs:	A folder containing license files, AAR files, and UI Extensions Component source code.
samples:	A folder containing Android sample projects.
getting_started_android.pdf:	A quick guide for Foxit PDF SDK for Android.
legal.txt:	Legal and copyright information.
release_notes.txt:	Release information.

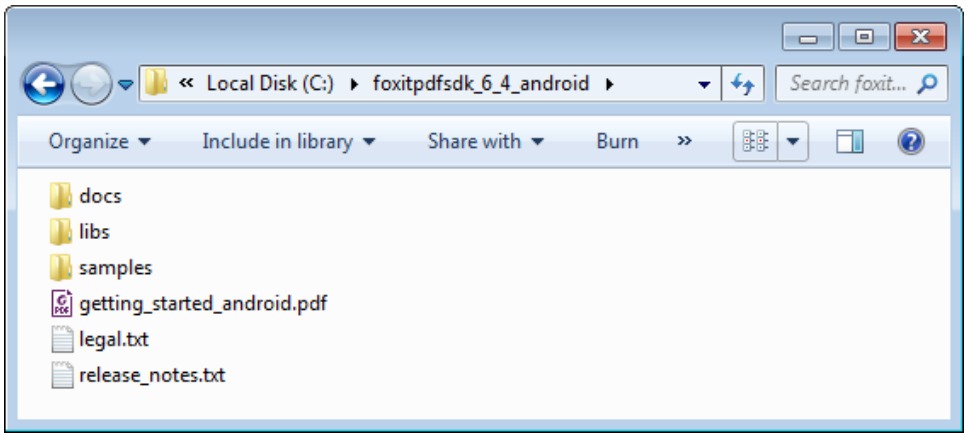


Figure 2-1

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

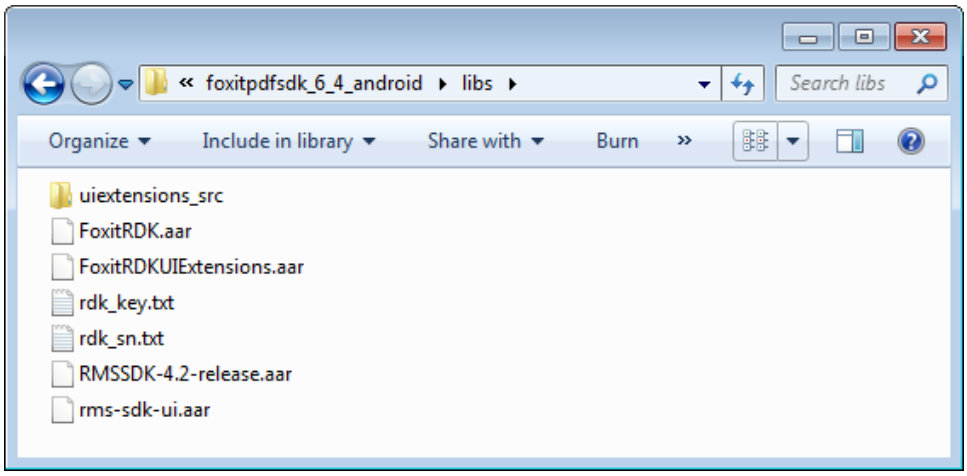


Figure 2-2

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

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

- ***FoxitRDKUIExtensions.aar*** - generated by the "**uiextensions_src**" project found in the "libs" folder. It includes the JAR package, built-in UI implementation, and resource files that are needed for the built-in UI implementations, such as images, strings, color values, layout files, and other Android UI resources.
- ***RMSSDK-4.2-release.aar*** - used for creating rights-enabled applications. For more detailed information, please refer to https://www.microsoft.com/en-ie/download/details.aspx?id=43673.
- ***rms-sdk-ui.aar*** – provides Android Activities that implement the UI required for the RMS SDK functionality. For more detailed information, please refer to https://github.com/AzureAD/rms-sdk-ui-for-android.

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

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

2.3 How to run a demo

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

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

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

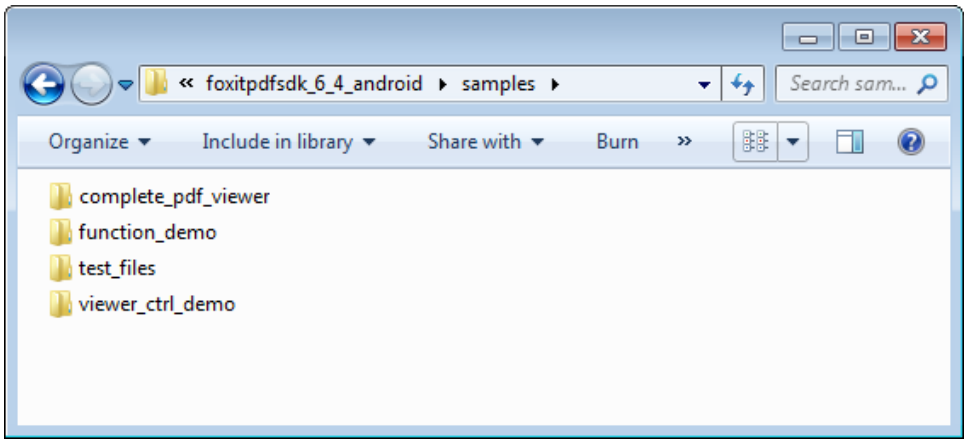


Figure 2-3

2.3.1 Function demo

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

- **pdf2txt**: extract text from a PDF document to a TXT file.
- **outline**: edit outline (aka bookmark) appearances and titles.
- **annotation**: add annotations 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). In this section, an AVD targeting 8.1 will be used as an example. The test files in the "samples/test_files" that are needed for the demos will be copied to the emulator's storage card automatically when running the demos.

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

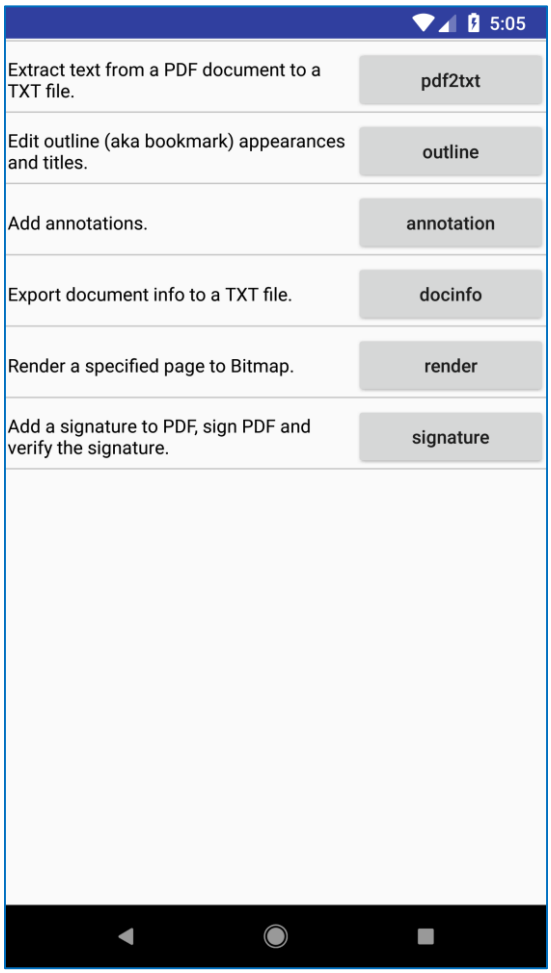


Figure 2-4

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

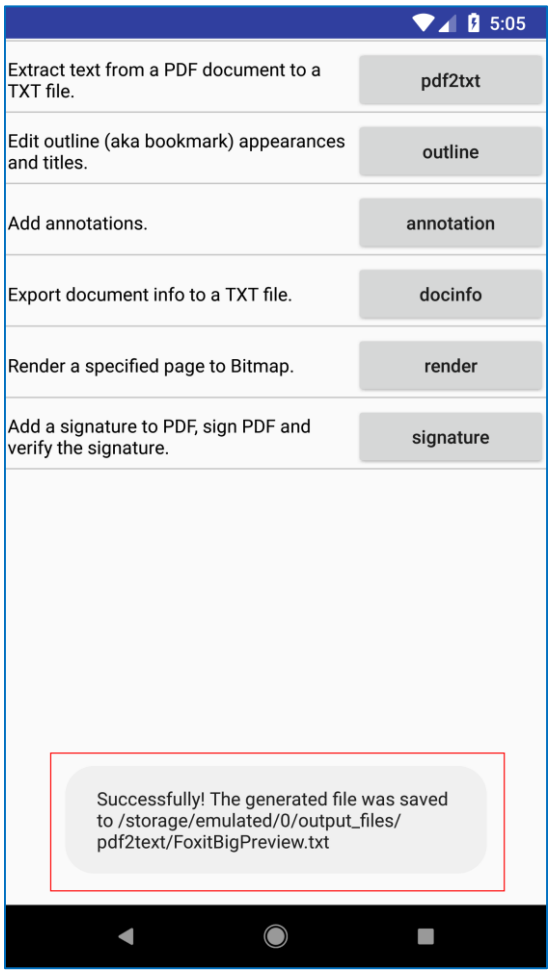


Figure 2-5

2.3.2 Viewer control demo

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

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

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

Figure 2-6 shows what the demo looks like after it was built successfully. Here, an AVD targeting 8.1 will be used as an example to run the demo.

Figure 2-6

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

Figure 2-7

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

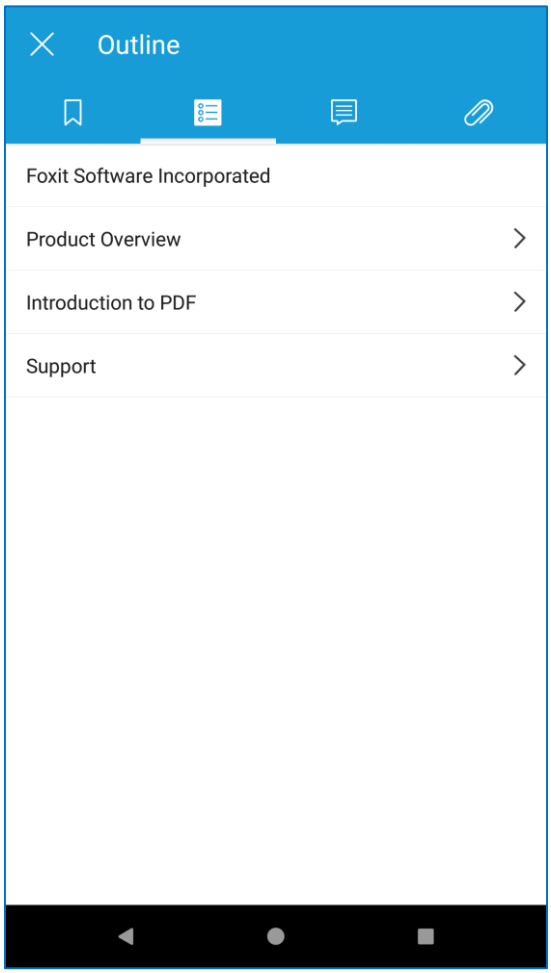


Figure 2-8

2.3.3 Complete PDF viewer demo

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

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

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

Here, an AVD targeting 8.1 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 2-9).

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

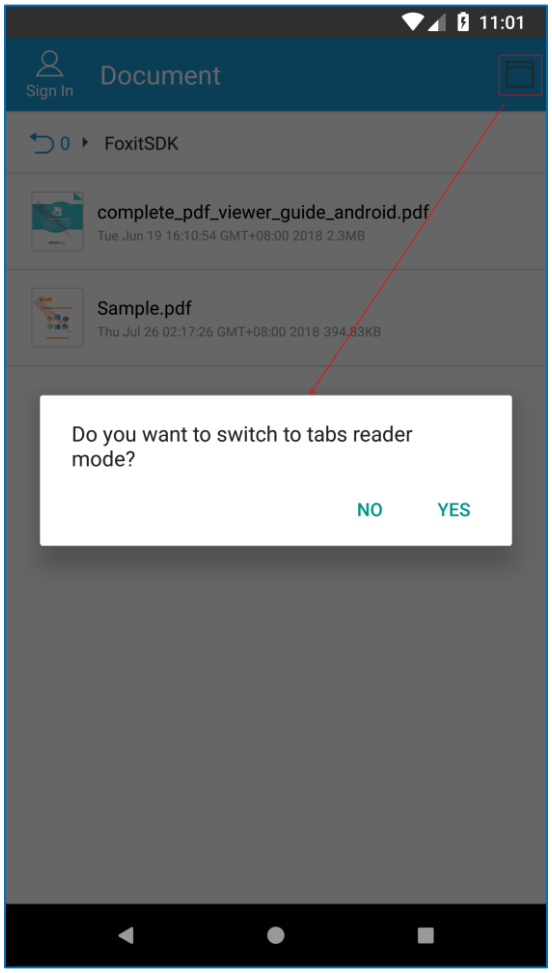


Figure 2-9

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

Figure 2-10

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

For example, it provides the page thumbnail feature. You can click the **View** menu, choose the **Thumbnail** as shown in Figure 2-11, and then the thumbnail of the document will be displayed as shown in Figure 2-12.

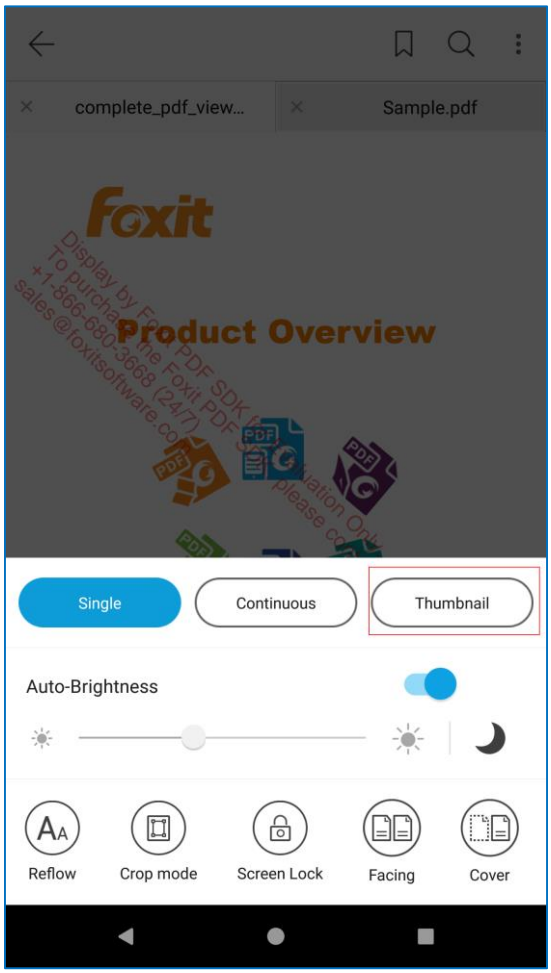

complete_pdf_view...
Sample.pdf
Product Overview
Single
Continuous
Thumbnail
Auto-Brightness
Reflow
Crop mode
Screen Lock
Facing
Cover

Figure 2-11

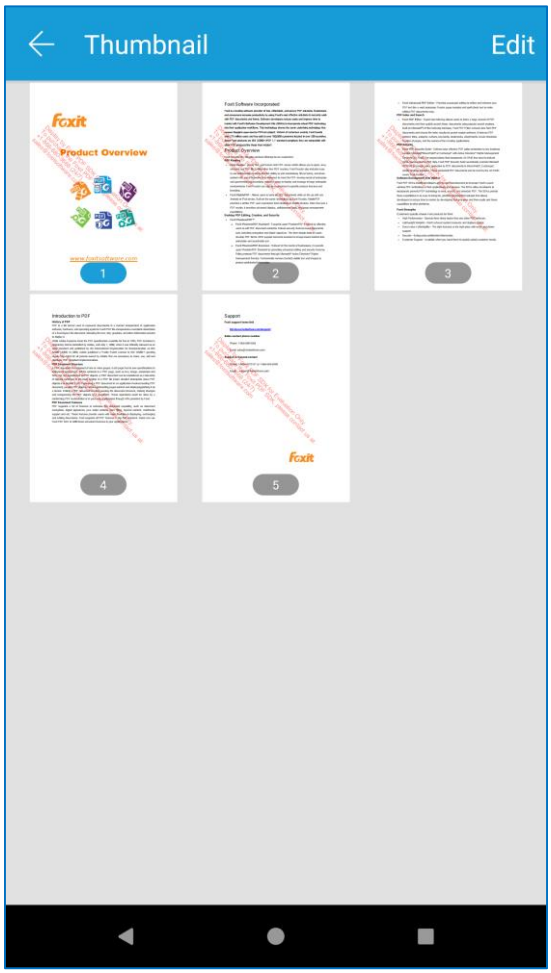


Figure 2-12

3 Rapidly building a full-featured PDF Reader

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

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

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

3.1 Create a new Android project

In this guide, we use Android Studio 3.1.3 along with Android API revision 27.

Open Android Studio, choose **File** -> **New** -> **New Project**… to start the **Create Android Project** wizard, and then fill the **New Project** dialog as shown in Figure 3-1. After filling, click **Next**.

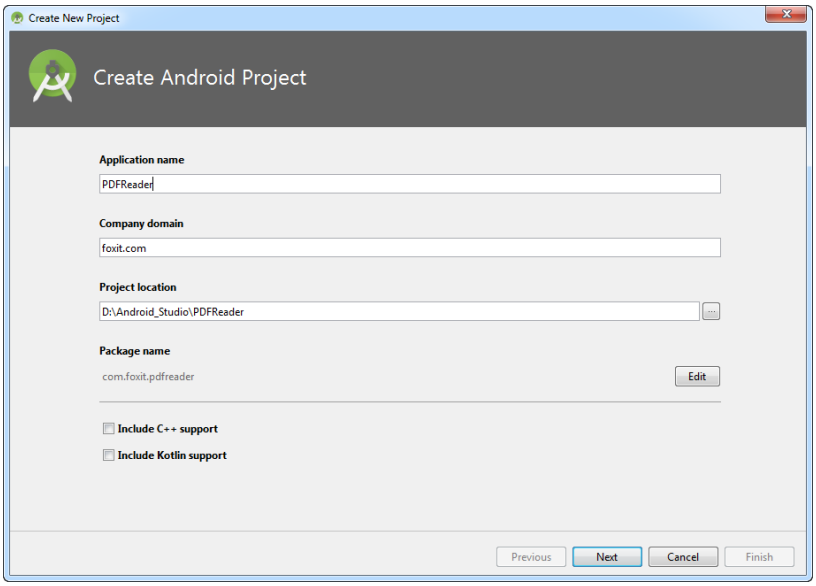


Figure 3-1

In the **Target Android Devices** dialog, set the minimum SDK to API 16. So choose "API 16: Android 4.1 (Jelly Bean)" from the drop-down list as shown in Figure 3-2. Then, click **Next**.

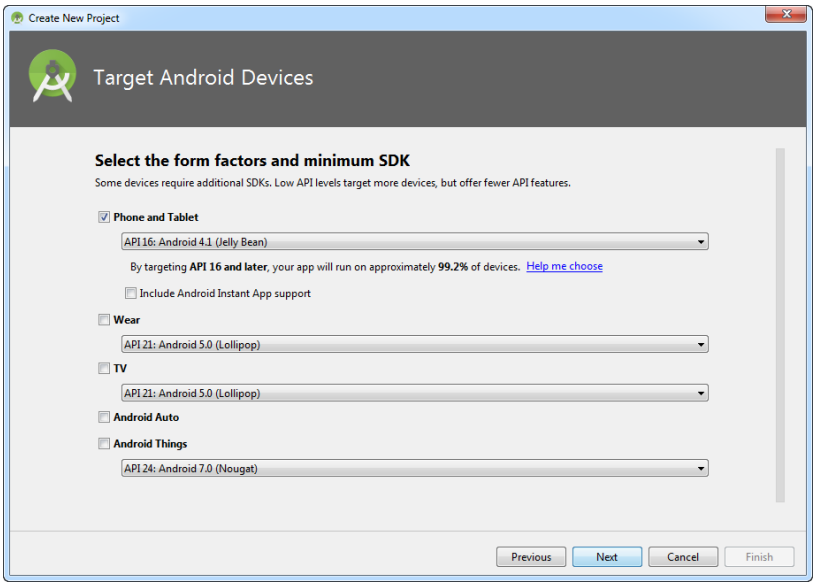


Figure 3-2

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**.

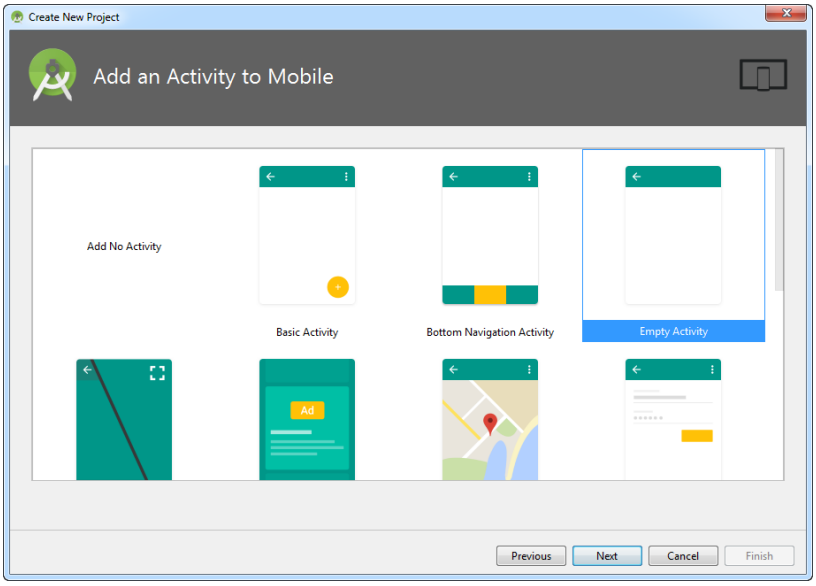


Figure 3-3

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

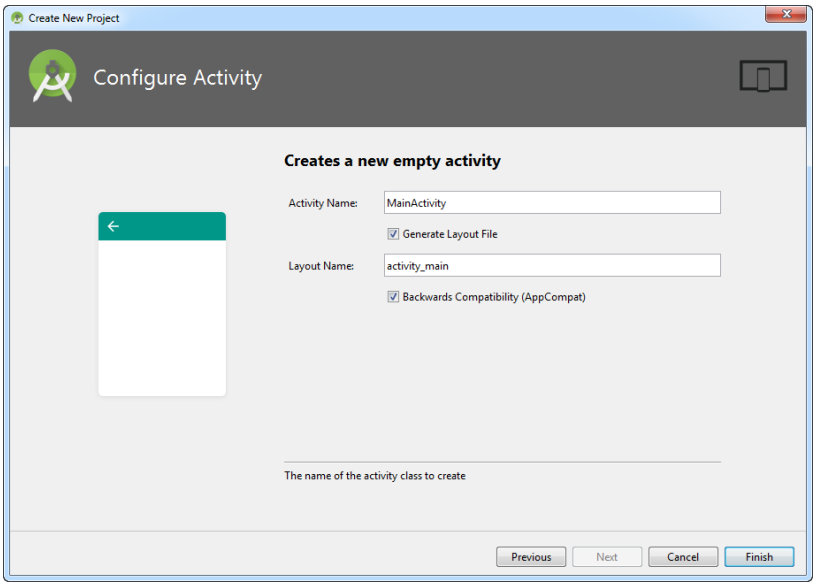


Figure 3-4

3.2 Integrate Foxit PDF SDK for Android into your apps

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

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

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

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

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

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

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

   Synchronize *PDFReader* project, and then it should look like Figure 3-5.

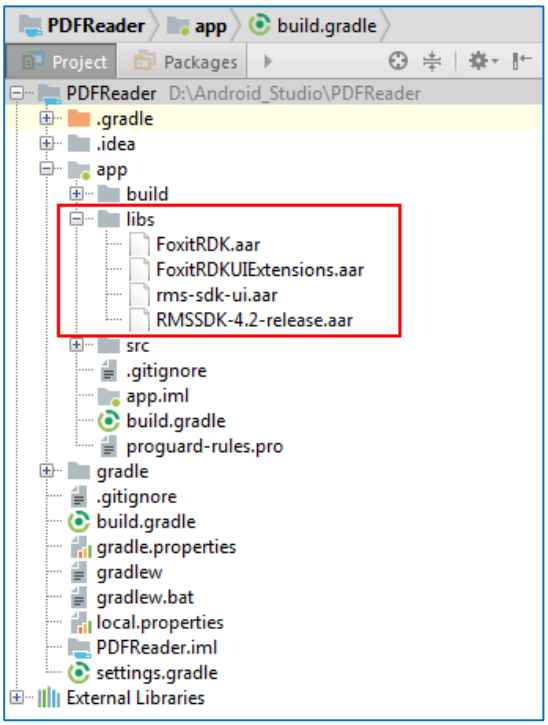


Figure 3-5

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

build.gradle:

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

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

```
dependencies {
    implementation(name: 'FoxitRDK', ext: 'aar')
    implementation(name: 'FoxitRDKUIExtensions', ext: 'aar')
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.edmodo:cropper:1.0.1'
    implementation('com.microsoft.aad:adal:1.1.16') {}
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
```

```
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
    testImplementation 'junit:junit:4.12'
}
```

Note:

- *(**Required**) Foxit PDF SDK for Android has a dependency on **recyclerview** support library, so you should add it to the dependencies. In this project, we add `'com.android.support:design:27.1.1'` which has already included the **recyclerview** package to the dependencies. Or you can add "`implementation 'com.android.support:recyclerview-v7:27.1.1'`" directly.*
- *(**Optional**) If you want to use the Snapshot feature (such as in the Complete PDF viewer demo, click ⋮ at the right top toolbar, then you can see the Snapshot option), you should add "`implementation 'com.edmodo:cropper:1.0.1'`" to the dependencies.*
- *(**Optional**) If you want to open a RMS protected PDF file, you should add the following entries to the dependencies:*

  ```
  implementation('com.microsoft.aad:adal:1.1.16') {}
  implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
  implementation(name: 'rms-sdk-ui', ext: 'aar')
  ```

  ***Note**: You had better use the verson of RMS SDK 4.2 and ADAL 1.1.16, otherwise there may cause compatibility issues.*

Here, we add all the above support libraries to the dependencies, because later we will build a full-featured PDF Reader which contains all the features provided by Foxit PDF SDK for Android. After setting the app's "build.gradle" file, sync it, then the "***FoxitRDK.aar***", "***FoxitRDKUIExtensions.aar***", "***recyclerview***", "***cropper***", and ***RMS*** related pacakges will appear in **External Libraries** as shown in Figure 3-6.

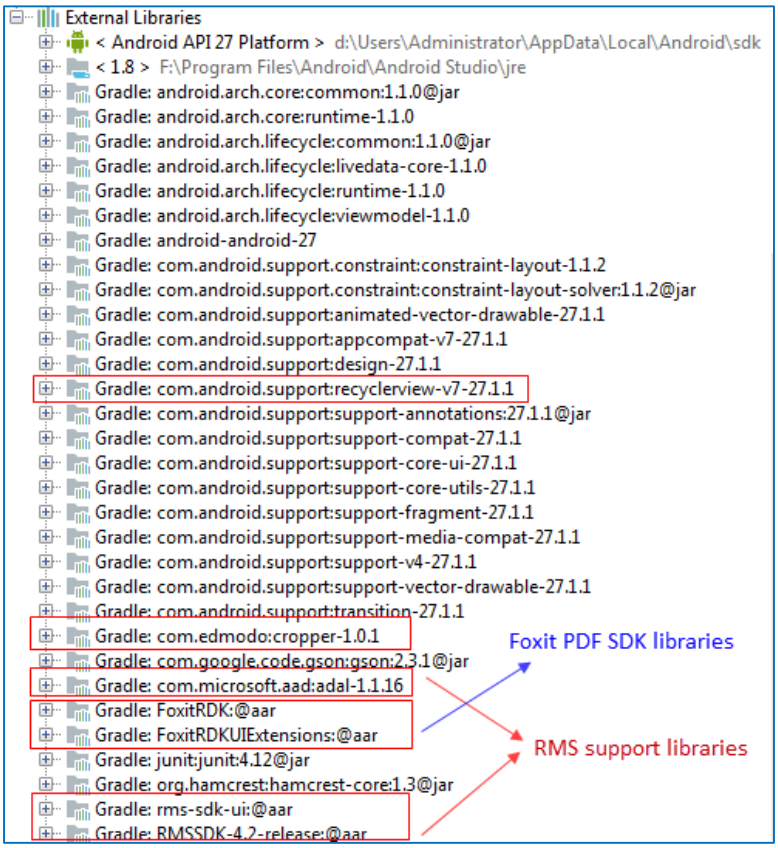


Figure 3-6

The following code shows "build.gradle" in its entirety.

build.gradle:

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

android {
    compileSdkVersion 27
    defaultConfig {
        applicationId "com.foxit.pdfreader"
        minSdkVersion 16
        targetSdkVersion 27
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
```

```
    }
}

repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    implementation(name: 'FoxitRDK', ext: 'aar')
    implementation(name: 'FoxitRDKUIExtensions', ext: 'aar')
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.edmodo:cropper:1.0.1'
    implementation('com.microsoft.aad:adal:1.1.16') {}
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
    testImplementation 'junit:junit:4.12'
}
```

Note *So far, we set the* `compileSdkVersion` *and* `targetSdkVersion` *to API 27. If you also want to use API 27, please make sure you have already installed the SDK Platform Android 8.1, API 27. If you have not already done this, open the Android SDK Manager to download and install it first.*

3.3 Initialize Foxit PDF SDK for Android

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

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

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

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

3.4 Display a PDF document using PDFViewCtrl

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

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

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

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

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

```
import com.foxit.sdk.PDFViewCtrl;
...
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:

```
package com.foxit.pdfreader;

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

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

public class MainActivity extends AppCompatActivity {
```

```
    private PDFViewCtrl pdfViewCtrl = null;

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

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

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

        pdfViewCtrl = new PDFViewCtrl(this);
        String path = "/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.

Note: *If you want to run this project on an Android 6.0 (API 23) or higher devices/emulators, you can do one of the following:*

1. *Change the `targetSdkVersion` in app's "build.gradle" from 27 to the SDK version that is less than 23, such as 21.*
2. *Write additional code to require the authorization of runtime permissions.*

a) Change the *targetSdkVersion* in app's "build.gradle" from 27 to 21.

In this case, 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 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"/>
```

```
    <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">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

b) Write additional code to require the authorization of runtime permissions.

In the **MainActivity.java** file, add the following code to require the authorization of runtime permissions:

```
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
...

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

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

```
    }
}
...

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
    if (requestCode == 1 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        // Open and Reader a PDF document.
        String path = "/mnt/sdcard/input_files/Sample.pdf";
        pdfViewCtrl.openDoc(path, null);
        setContentView(pdfViewCtrl);
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

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

```
package com.foxit.pdfreader;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

public class MainActivity extends AppCompatActivity {

    private PDFViewCtrl pdfViewCtrl = null;

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

    // The value of "sn" can be found in the "rdk_sn.txt".
    // The value of "key" can be found in the "rdk_key.txt".
```

```
    private static String sn = " ";
    private static String key = " ";

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

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

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

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

        // Open and Render a PDF document.
        String path = "/mnt/sdcard/input_files/Sample.pdf";
        pdfViewCtrl.openDoc(path, null);
        setContentView(pdfViewCtrl);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
        if (requestCode == 1 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Open and Render a PDF document.
            String path = "/mnt/sdcard/input_files/Sample.pdf";
            pdfViewCtrl.openDoc(path, null);
            setContentView(pdfViewCtrl);
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

In this section, we build and run the project on an AVD targeting 8.1 (API 27), and use the second method (require authorization of runtime permissions) to get the permissions to write and read the SD card of the emulator.

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

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see that the "Sample.pdf" document is displayed as shown in Figure 3-7. 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 Open a RMS protected document

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

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

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

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

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

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

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

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

```
package com.foxit.pdfreader;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

public class MainActivity extends AppCompatActivity {

    private PDFViewCtrl pdfViewCtrl = null;

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

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

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

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

```
            return;

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

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

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

        // Open and Render a PDF document.
        String path = "/mnt/sdcard/input_files/Sample_RMS.pdf";
        pdfViewCtrl.openDoc(path, null);
        setContentView(pdfViewCtrl);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
        if (requestCode == 1 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Open and Render a PDF document.
            String path = "/mnt/sdcard/input_files/Sample_RMS.pdf";
            pdfViewCtrl.openDoc(path, null);
            setContentView(pdfViewCtrl);
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }

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

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

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see the following window (Figure 3-8) which prompts you to input your organizational email, and later input the password, then you can open the RMS protected document.

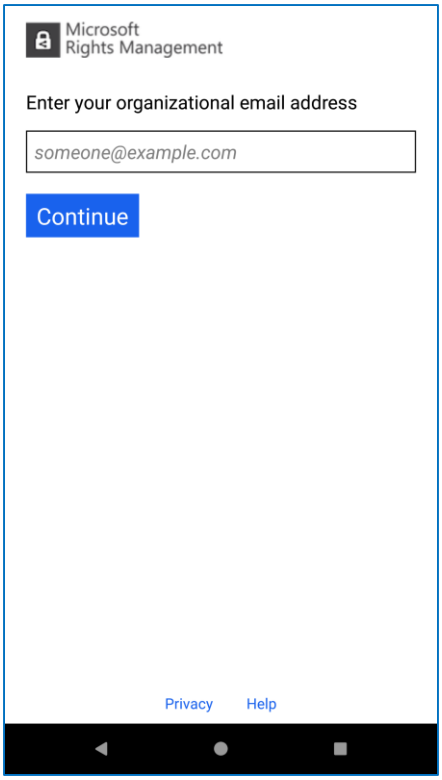


Figure 3-8

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

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

Instantiate a UIExtensionsManager object and set it to PDFViewCtrl

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

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

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

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

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

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

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

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

private UIExtensionsManager uiExtensionsManager = null;
...

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

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

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

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

String path = "/mnt/sdcard/input_files/Sample.pdf";
uiExtensionsManager.openDocument(path, null);
setContentView(uiExtensionsManager.getContentView());
```

Update MainActivity.java as follows:

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

```
package com.foxit.pdfreader;

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

```
import android.content.res.Configuration;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
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;

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

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

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

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

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

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

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

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

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

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

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

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
        if (requestCode == 1 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Open and Render a PDF document.
            String path = "/mnt/sdcard/input_files/Sample.pdf";
            uiExtensionsManager.openDocument(path, null);
            setContentView(uiExtensionsManager.getContentView());
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }

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

```
    }

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

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

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

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

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

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

```
            uiExtensionsManager.onConfigurationChanged(this, newConfig);
        }
    }

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

Update AndroidManifest.xml

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

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

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

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

Update AndroidManifest.xml as follows:

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

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

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

```
        android:supportsRtl="true"
        android: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>
    </application>

</manifest>
```

Run the project

In this section, we build and run the project on an AVD targeting Android 8.1 (API 27). After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see that the "Sample.pdf" document is displayed as shown in Figure 3-9. Up to now, it is a full-featured PDF Reader which includes all of the features in Complete PDF viewer demo and it also supports to open a RMS protected document. Feel free to try it.

Figure 3-9

4 Customizing User Interface

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

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

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

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

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

4.1 Customize the UI through a configuration file

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

4.1.1 Introduction to JSON file

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

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

```
{
    "modules": {
        "readingbookmark": true,
        "outline": true,
```

```
//        "annotations":true,
        "annotations": {
            "highlight": true,
            "underline": true,
            "squiggly": true,
            "strikeout": true,
            "insert": true,
            "replace": true,
            "line": true,
            "rectangle": true,
            "oval": true,
            "arrow": true,
            "pencil": true,
            "eraser": true,
            "typewriter": true,
            "textbox": true,
            "callout": true,
            "note": true,
            "stamp": true,
            "polygon": true,
            "cloud": true,
            "polyline": true,
            "distance": true,
            "image": true,
            "audio": true,
            "video": true
        },
        "thumbnail": true,
        "attachment": true,
        "signature": true,
        "search": true,
        "navigation": true,
        "form": true,
        "selection": true,
        "encryption": true
        "multipleSelection": true
    },
    "permissions": {
         "runJavaScript": true,
         "copyText": true,
         "disableLink": false
    },
    "uiSettings": {
      "displayMode": "Single",
      "zoomMode": "FitWidth",
      "colorMode": "Normal",
```

```
    "mapForegroundColor": "#5d5b71",
    "mapBackgroundColor": "#00001b",
    "disableFormNavigationBar": false,
    "highlightForm": true,
    "highlightFormColor": "#200066cc",
    "highlightLink": true,
    "highlightLinkColor": "#16007fff",
    "fullscreen": true,
    "annotations": {
            "continuouslyAdd": false,
             "highlight": {
               "color": "#ffff00", "opacity": 1.0                },
             "underline": {
               "color": "#66cc33", "opacity": 1.0
             },
             "squiggly": {
               "color": "#ff6633", "opacity": 1.0
             },
             "strikeout": {
               "color": "#ff0000", "opacity": 1.0
             },
             "insert": {
               "color": "#993399", "opacity": 1.0
             },
             "replace": {
               "color": "#0000ff", "opacity": 1.0
             },
             "line": {
               "color": "#ff0000", "opacity": 1.0, "thickness": 2
             },
             "rectangle": {
               "color": "#ff0000", "opacity": 1.0, "thickness": 2
             },
             "oval": {
               "color": "#ff0000", "opacity": 1.0, "thickness": 2
             },
             "arrow": {
               "color": "#ff0000", "opacity": 1.0, "thickness": 2
             },
             "pencil": {
               "color": "#ff0000", "opacity": 1.0, "thickness": 2
             },
             "polygon": {
               "color": "#ff0000", "opacity": 1.0, "thickness": 2
             },
             "cloud": {
```

```
    "color": "#ff0000", "opacity": 1.0, "thickness": 2
  },
  "polyline": {
    "color": "#ff0000", "opacity": 1.0, "thickness": 2
  },
  "typewriter": {
    "textColor": "#0000ff",
    "opacity": 1.0,
    "textFace": "Courier",
    "textSize": 18
  },
  "textbox": {
    "color": "#ff0000",
    "textColor": "#0000ff",
    "opacity": 1.0,
    "textFace": "Courier",
    "textSize": 18
  },
  "callout": {
    "color": "#ff0000",
    "textColor": "#0000ff",
    "opacity": 1.0,
    "textFace": "Courier",
    "textSize": 18
  },
  "note": {
    "color": "#ff6633",
    "opacity": 1.0,
    "icon": "Comment"
  },
  "attachment": {
    "color": "#ff6633",
    "opacity": 1.0,
    "icon": "PushPin"
  },
  "image": {
     "rotation": 0,
    "opacity": 1.0
  },
  "distance": {
     "color": "#ff0000",
     "opacity": 1.0,
     "thickness": 2,
     "scaleFromUnit": "inch",
     "scaleToUnit": "inch",
     "scaleFromValue": 1,
```

```
                    "scaleToValue": 1
                }
        },
        "signature": {
            "color": "#000000", "thickness": 4
        }
    }
}
```

***Note**:*

- *The values in the above JSON file are the default settings for the configuration items. If some configuration items are not in the JSON file, the default settings will be used. For example, if you comment out "**"highlight": true,**", it is still enabled.*
- *Only the attachment annotation in the Annotation setting bar (See Figure 4-1, to find the Annotation setting bar, just click **Comment** at the bottom toolbar, and then click ) 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 toolbar, you should set both "**annotations**" and "**attachment**" to "**false**".*

Figure 4-1

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.

4.1.2 Configuration Items Description

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

Configure feature module

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

Feature Module	Description
readingbookmark	User-defined bookmark
outline	PDF document bookmark
annotations (highlight, underline, squiggly, strikeout, insert, replace, line, rectangle, oval, arrow, pencil, eraser, typewriter, textbox, callout, note,	Annotation module collection

stamp, polygon, cloud, polyline, distance, image, audio, video)	
thumbnail	PDF page thumbnail display and page management
attachment	PDF document attachments
signature	Digital signatures and handwritten signatures
search	Text search
navigation	PDF page navigation
form	Form Filling and form data importing and exporting
selection	Text selection
encryption	PDF encryption
multipleSelection	Multiple annotations selection

Configure rights management

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

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

Configure UI settings

UI Items	Description/ Property Items	Value Type	Available Value	Default value	Note
displayMode	Page display mode	String	Single/ Continous/ Facing/ CoverRight/ Reflow	Single	For dynamic XFA files, it doesn't support Reflow mode.
zoomMode	Page zoom mode	String	FitWidth/FitPage	FitWidth	
colorMode	Page color display mode	String	Normal/Night/Map	Normal	"Night" is a special "Map" mode.
mapForegroundColor	Foreground color of page display	RGB	---	#5d5b71	It is vaild only when "colorMode" is set to "Map".
mapBackgroundColor	Background color of page dispay	RGB	---	#00001b	It is vaild only when "colorMode" is set to "Map".
disableFormNavigationBar	Whether to disable the	Bool	true/false	false	

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
		supplementary navigation bar of the form				
highlightForm		Whether to highlight form field	Bool	true/false	true	
highlightFormColor		The highlight color of forms	ARGB		#200066cc	It include alpha channel, and it is invalid for dynamic xfa document.
highlightLink		Whether to highlight hyperlink	Bool	true/false	true	
highlightLinkColor		The highlight color of links	ARGB		#16007fff	It include alpha channel.
fullscreen		Whether to display in full screen mode	Bool	true/false	true	It will be in full screen mode immediately when opening a document if "fullscreen" is set to "true". If the user clicks on the page, the toolbar will be displayed. After 5 seconds, if it is in full screen mode, the toolbar and other auxiliary tool buttons will be hidden automatically.
annotations	continuouslyAdd		Bool	true/false	false	Whether to add annotation continuously
	highlight	color	RGB		#ffff00	
		opacity	numeric	[0.0-1.0]	1.0	
	underline	color	RGB		#66cc33	
		opacity	numeric	[0.0-1.0]	1.0	
	squiggly	color	RGB		#ff6633	
		opacity	numeric	[0.0-1.0]	1.0	
	strikeout	color	RGB		#ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
	insert	color	RGB		#993399	
		opacity	numeric	[0.0-1.0]	1.0	

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

UI Items		Description/ Property Items	Value Type	Available Value	Default value	Note
		textSize	Integer	>=1	18	
	callout	color	RGB		#ff0000	
		textColor	RGB		#0000ff	
		opacity	numeric	[0.0-1.0]	1.0	
		textFace	String	Courier/ Helvetica/ Times	Courier	Text font name. If set to an invalid value, the default value will be used.
		textSize	Integer	>=1	18	
	note	color	RGB		#ff6633	
		opacity	numeric	[0.0-1.0]	1.0	
		icon	String	Comment/ Key/ Note/ Help/ NewParagraph/ Paragraph/ Insert	Comment	If set to an invalid value, the default value will be used.
	attachment	color	RGB		#ff6633	
		opacity	numeric	[0.0-1.0]	1.0	
		icon	String	Graph/ PushPin/ Paperclip/ Tag	PushPin	If set to an invalid value, the default value will be used.
	image	rotation	numeric	0/90/180/270	0	
		opacity	numeric	[0.0-1.0]	1.0	
	distance	color	RGB		ff0000	
		opacity	numeric	[0.0-1.0]	1.0	
		thickness	numeric	[1-12]	2	
		scaleFromUnit	String	pt/m/cm/mm/inch/p/ft/yd	inch	The original unit of the scale
		scaleToUnit	String	pt/m/cm/mm/inch/p/ft/yd	inch	The target unit of the scale
		scaleFromValue	numeric		1	The original value of the scale
		scaleToValue	numeric		1	The target value of the scale
signature		color	RGB		#000000	
		thickness	numeric	[1-12]	4	

4.1.3 Instantiate a UIExtensionsManager object with the configuration file

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

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

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

In "MainActivity.java":

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

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

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

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

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

4.1.4 Examples for customizing UI through a configuration file

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

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

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

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

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

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

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

Before: After:

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

Example2: Disable hyperlinks.

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

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

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

Example3: Set the highlight color from yellow to red.

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

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

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

Before:

After:

The highlight color has been changed to red.

4.2 Customize UI elements through APIs

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

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

4.2.1 Customizing top/bottom toolbar

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

1. Show or hide the top/bottom toolbar.
2. Add a custom item at any position.
3. Remove a specific item.
4. Remove all of the items in the toolbar.
5. Show or hide a specific item.
6. Add a custom toolbar.
7. Remove a specific toolbar.
8. Set background color for the toolbar.
9. Get the number of the items in a specific location of the toolbar.

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

Table 4-1

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

int **getItemsCount**(BarName barName, BaseBar.TB_Position gravity);	Get the items count by IBarsHandler.BarName and BaseBar.TB_Position.
boolean **removeItem**(BarName barName, BaseBar.TB_Position gravity, int index);	Remove the item at the specified position in the toolbar.
boolean **removeItem**(BarName barName, BaseBar.TB_Position gravity, BaseItem item);	Remove the item by IBarsHandler.BarName, BaseBar.TB_Position and the specified item.
void **removeAllItems**(BarName barName);	Remove all items from the toolbar.
boolean **addCustomToolBar**(BarName barName, View view);	Add custom toolbar by BarName.
boolean **removeToolBar**(BarName barName);	Remove toolbar by BarName.
void **setBackgroundColor**(BarName barName, int color);	Set background color for the toolbar.
void **setBackgroundResource**(BarName barName, int resid);	Set background to a given resource.
BaseItem **getItem**(BarName barName, BaseBar.TB_Position gravity, int tag);	Get the item by tag, if tag does not exist, it return null.

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

```
enum BarName {
        TOP_BAR,
        BOTTOM_BAR;
    }

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

Note:

1. The top toolbar or bottom toolbar can only add up to 7 items.
2. To add an item to the top toolbar, please set the **BaseBar.TB_Position** to **Position_LT** or **Position_RB.** To add an item to the bottom toolbar, please set the **BaseBar.TB_Position** to **Position_CENTER**. Otherwise, the items may overlap.
3. The bottom toolbar is only one part, and the top toolbar is divided into two parts, so that there are three parts for the toolbar, and each part has a separate index.(See Figure 4-2)
4. For the best UI display, it is recommended to control the character number of text for top toolbar within 15 and for bottom toolbar within 8. If don't, the view layout may be in disorder.

Figure 4-2

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

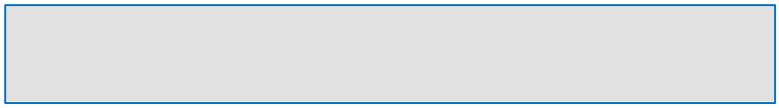
Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "`mUiExtensionsManager = new UIExtensionsManager(getActivity().getApplicationContext(), pdfViewerCtrl, config);`").

Example1: Hide the whole top toolbar.

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

Before:

After:

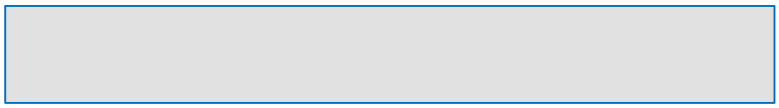


Example2: Hide the whole bottom toolbar.

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

Before:

After:



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

```
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 toolbar at the
second position.");
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.TOP_BAR,
BaseBar.TB_Position.Position_LT, mTopItem1, 1);
```

The result after running the demo:

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

```
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 toolbar at the
first position");
    }
});
```

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

The result after running the demo:

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

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

The result after running the demo:

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

```
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
toolbar at the second position.");
    }
});
mUiExtensionsManager.getBarManager().addItem(IBarsHandler.BarName.BOTTOM_BAR,
BaseBar.TB_Position.Position_CENTER, mSettingBtn, 1);
```

The result after running the demo:

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

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

The result after running the demo:

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

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

Before: (See **Example3**)

After:

Example9: Remove all the items in the bottom toolbar.

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

Before:

After:

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

```
// 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 buttom in the left top toolbar 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 buttom in the left top toolbar 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_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 toolbar will look like as follows:

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

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

Example11: Remove the whole bottom toolbar.

```
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);
```

4.2.2 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 toolbar to find it), you can use the following APIs listed in the Table 4-2.

Table 4-2

public void **setPanelHidden**(boolean isHidden, PanelSpec.PanelType panelType)	To show or hide a panel according to the PanelType.
public boolean **isHiddenPanel**(PanelSpec.PanelType panelType)	Return the current value in **setPanelHidden**.

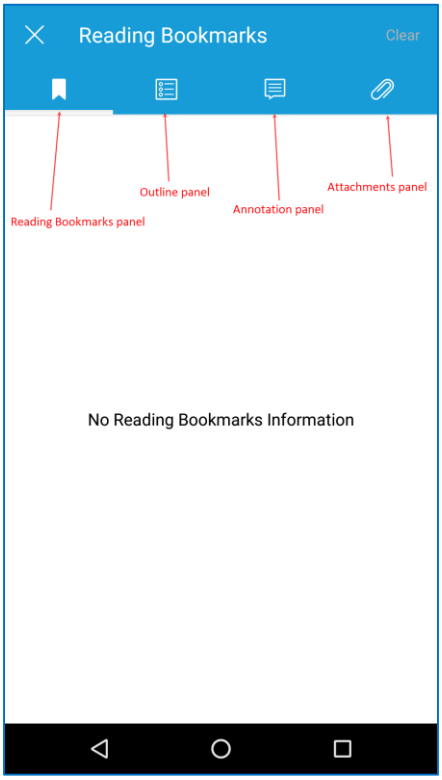


Figure 4-3

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:

Panel	PanelType
Reading Bookmarks	PanelSpec.PanelType.ReadingBookmarks
Outline	PanelSpec.PanelType.Outline
Annotations	PanelSpec.PanelType.Annotations
Attachments	PanelSpec.PanelType.Attachments

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);`").

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

```
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 toolbar 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 toolbar, and you will see the "Outline" panel has been hidden (See Figure 4-4).

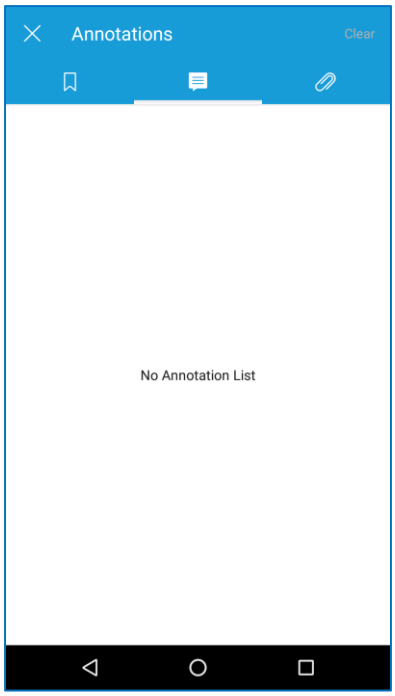


Figure 4-4

For Reading Bookmarks, Annotations, and Attachments panels, you only need to change the **`PanelType`**. Just try it.

4.2.3 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 toolbar to find it), you only need to use the following API:

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

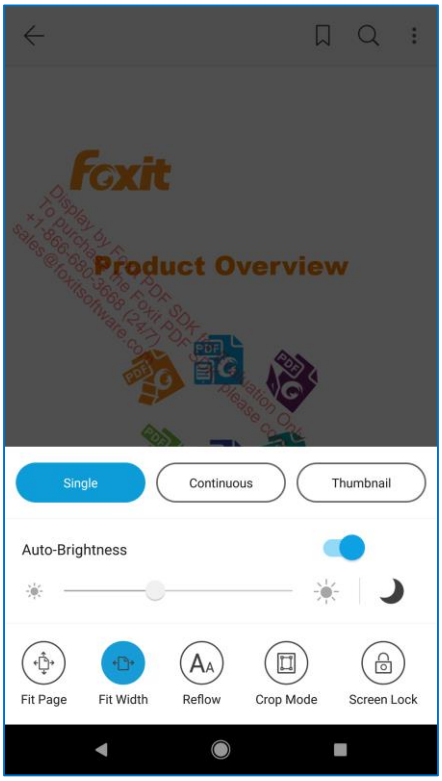


Figure 4-5

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

type	integer
IMultiLineBar.TYPE_LIGHT	1
IMultiLineBar.TYPE_DAYNIGHT	2
IMultiLineBar.TYPE_SYSLIGHT	4
IMultiLineBar.TYPE_SINGLEPAGE	8
IMultiLineBar.TYPE_CONTINUOUSPAGE	16
IMultiLineBar.TYPE_THUMBNAIL	32
IMultiLineBar.TYPE_LOCKSCREEN	64
IMultiLineBar.TYPE_REFLOW	128
IMultiLineBar.TYPE_CROP	256
IMultiLineBar.TYPE_FACING_MODE	288
IMultiLineBar.TYPE_COVER_MODE	320
IMultiLineBar.TYPE_PANZOOM	384

IMultiLineBar.TYPE_TYPE_FITPAGE	512
IMultiLineBar.TYPE_FITWIDTH	544

The value of the parameter "visibility" can be set as follows:

visibility	integer	description
View.VISIBLE	0	The view is visible.
View.INVISIBLE	4	This view is invisible, but it still takes up space for layout purposes.
View.GONE	8	This view is invisible, and it doesn't take any space for layout.

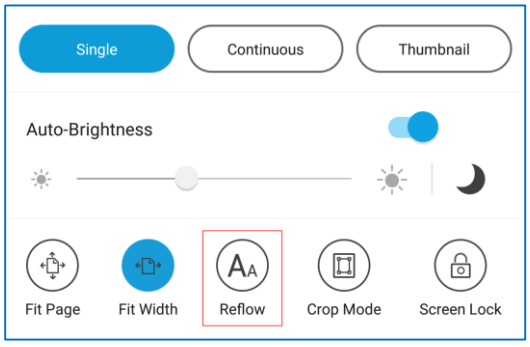
In this section, we only take "Reflow" item as an example to show you how to show or hide the UI elements in the View setting bar through APIs in the "**complete_pdf_viewer**" demo found in the "samples" folder. For other UI elements, you only need to change the "**type**".

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(), pdfViewerCtrl, config);`").

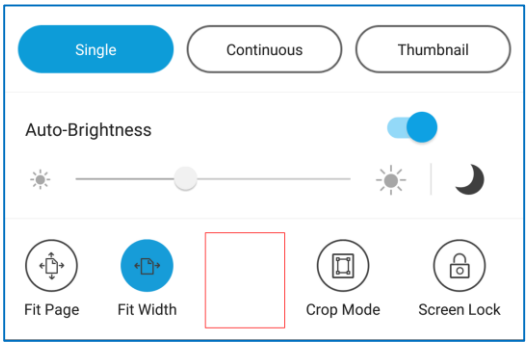
Example1: To hide the Reflow item in the View setting bar without changing the layout.

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

Before:



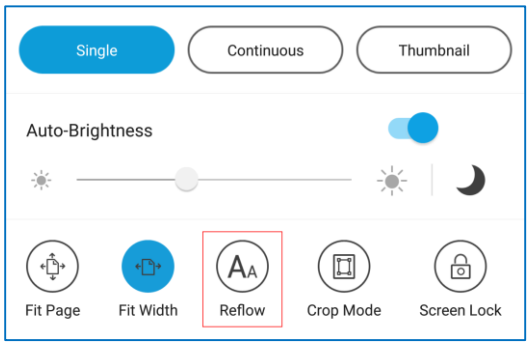
After:



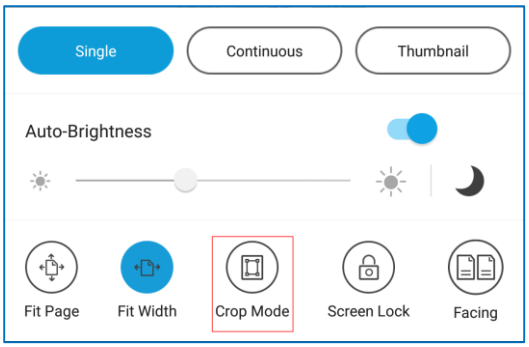
Example2: To hide the Reflow item in the View setting bar with changing the layout.

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

Before:



After:



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"**.

4.2.4 Customizing to show/hide the UI elements in the More Menu view

To show or hide the More Menu item, please see section 4.2.1 "Customizing top/bottom toolbar" (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 toolbar to find it), you can use the following APIs listed in the Table 4-3.

Table 4-3

`void` **`setGroupVisibility`**`(int visibility, int tag);`	`Set the enabled state of group according to "tag".`
`void` **`setItemVisibility`**`(int visibility, int groupTag, int itemTag);`	`Set the enabled state of item according to "groupTag" and "itemTag".`

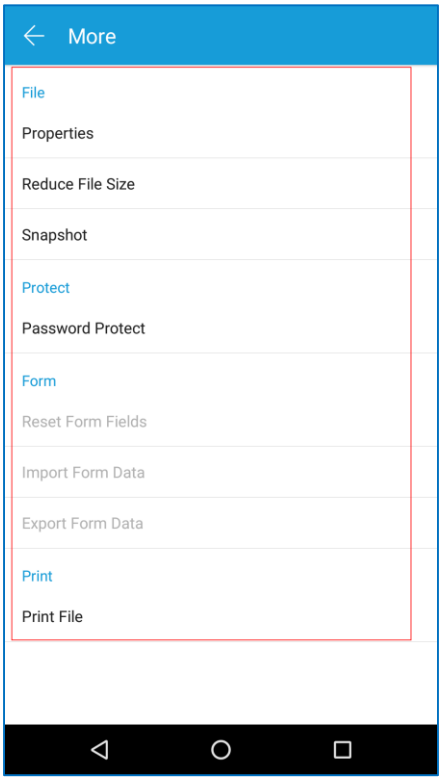


Figure 4-6

The value of the parameter "**tag**" in the **setGroupVisibility** interface or the "**groupTag**" in the **setItemVisibility** interface can be set as follows:

tag	integer
GROUP_FILE	100
GROUP_PROTECT	101
GROUP_FORM	102
GROUP_PRINT	103

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

groupTag	itemTag	integer
GROUP_FILE	ITEM_DOCINFO	0
	ITEM_REDUCE_FILE_SIZE	1
	ITEM_SNAPSHOT	2
GROUP_PROTECT	ITEM_PASSWORD	0
	ITEM_REMOVESECURITY_PASSWORD	4
GROUP_FORM	ITEM_RESET_FORM	0
	ITEM_IMPORT_FORM	1
	ITEM_EXPORT_FORM	2
GROUP_PRINT	ITEM_PRINT_FILE	0

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

visibility	integer	description
View.VISIBLE	0	The view is visible.
View.INVISIBLE	4	This view is invisible, but it still takes up space for layout purposes.
View.GONE	8	This view is invisible, and it doesn't take any space for layout.

***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.

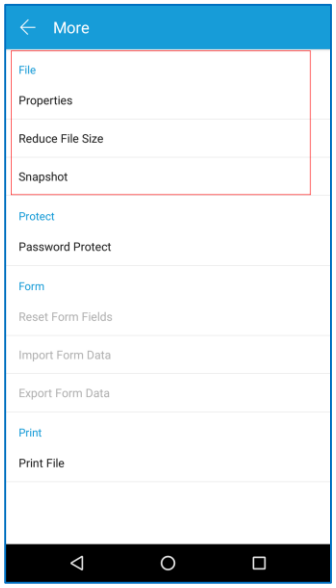
Load the "**complete_pdf_viewer**" demo in Android Studio. Add the sample code to the "PDFReaderFragment.java" file (after the code "`mUiExtensionsManager = new UIExtensionsManager(getActivity().getApplicationContext(), pdfViewerCtrl, config);`").

Example1: Hide the "File" in the More Menu view without changing the layout.

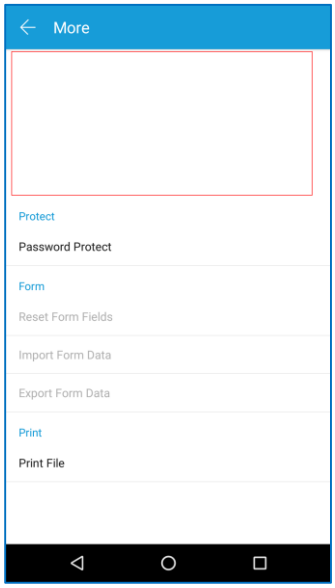
```
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule)
mUiExtensionsManager.getModuleByName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

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

Before:



After:

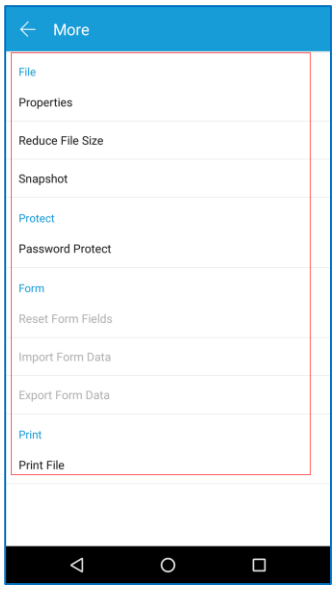


Example2: Hide the "File" in the More Menu view with changing the layout.

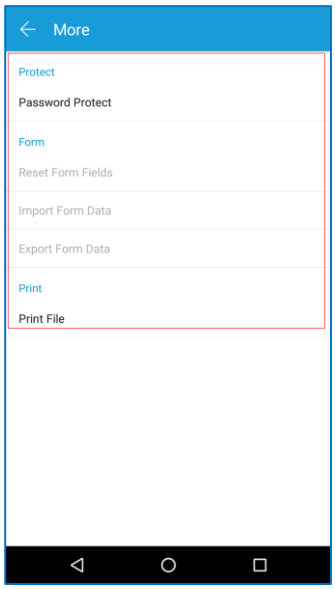
```
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule)
mUiExtensionsManager.getModuleByName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

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

Before:



After:

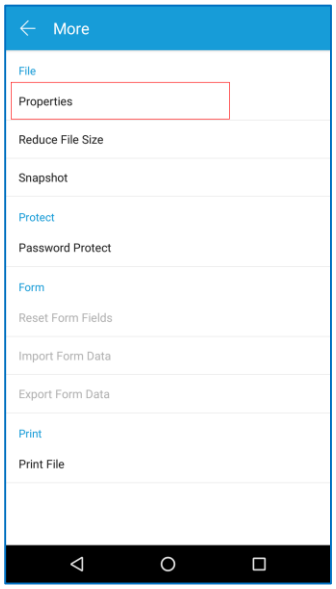


Example3: Hide the "Properties" in the More Menu view without changing the layout.

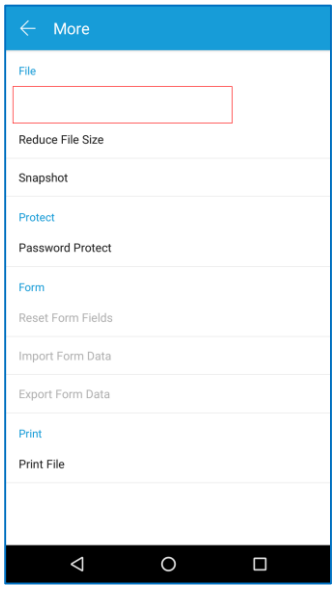
```
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule)
mUiExtensionsManager.getModuleByName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

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

Before:



After:

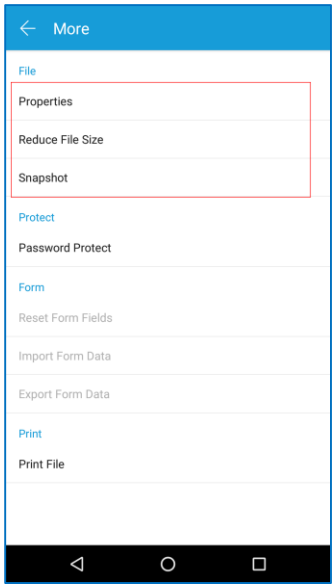


Example4: Hide the "Properties" in the More Menu view with changing the layout.

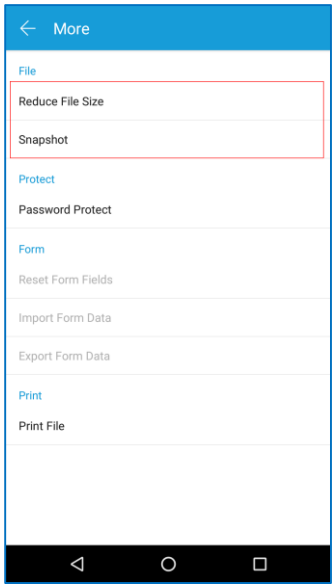
```
// Get MenuViewImpl from MoreMenuModule.
MoreMenuModule moreMenuModule = (MoreMenuModule)
mUiExtensionsManager.getModuleByName(Module.MODULE_MORE_MENU);
MenuViewImpl menuView = (MenuViewImpl) moreMenuModule.getMenuView();

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

Before:



After:



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 **setGroupVisibility** and **setItemVisibility** interfaces to **"View.VISIBLE"**.

4.3 Customize UI implementation through source code

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

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

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

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

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

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

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

UI Customization Example

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

Note *The demo already includes references to the* ***FoxitRDK.aar*** *file, so we just need to add the* ***uiextensions_src*** *project through configuring the "settings.gradle" file. When to include the*

***uiextensions_src** project as a dependency, the reference to the **FoxitRDKUIExtensions.aar** needs to be removed.*

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

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

settings.gradle:

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

Rebuild the gradle, then the ***uiextensions_src*** project will be added as shown in Figure 4-7.

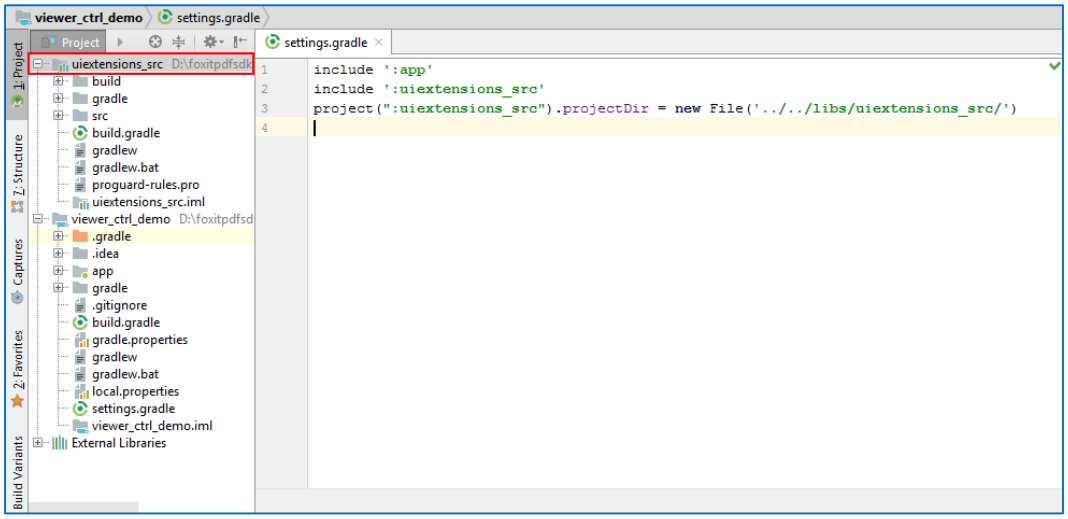


Figure 4-7

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 `implementation(name:'FoxitRDKUIExtensions', ext:'aar')` line as follows:

```
dependencies {
    implementation 'com.google.android:multidex:0.1'
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation (name: 'FoxitRDK', ext: 'aar')
    // implementation(name:'FoxitRDKUIExtensions', ext:'aar')
    implementation 'com.edmodo:cropper:1.0.1'
    implementation('com.microsoft.aad:adal:1.1.16') {}
    implementation(name: 'RMSSDK-4.2-release', ext: 'aar')
    implementation(name: 'rms-sdk-ui', ext: 'aar')
}
```

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.

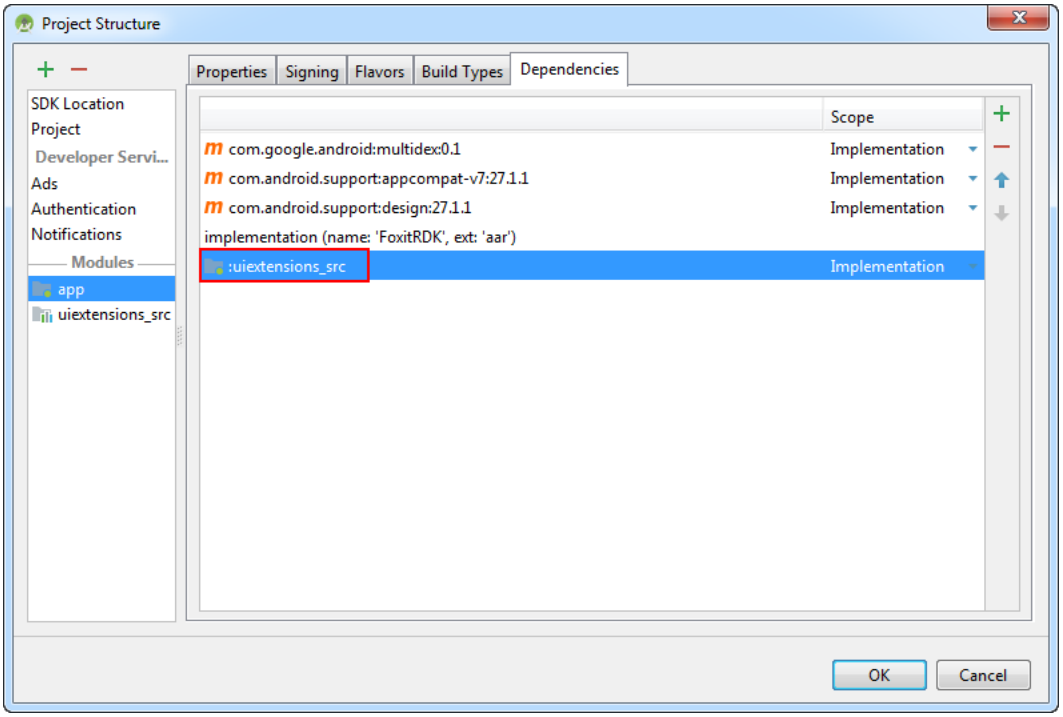


Figure 4-8

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.

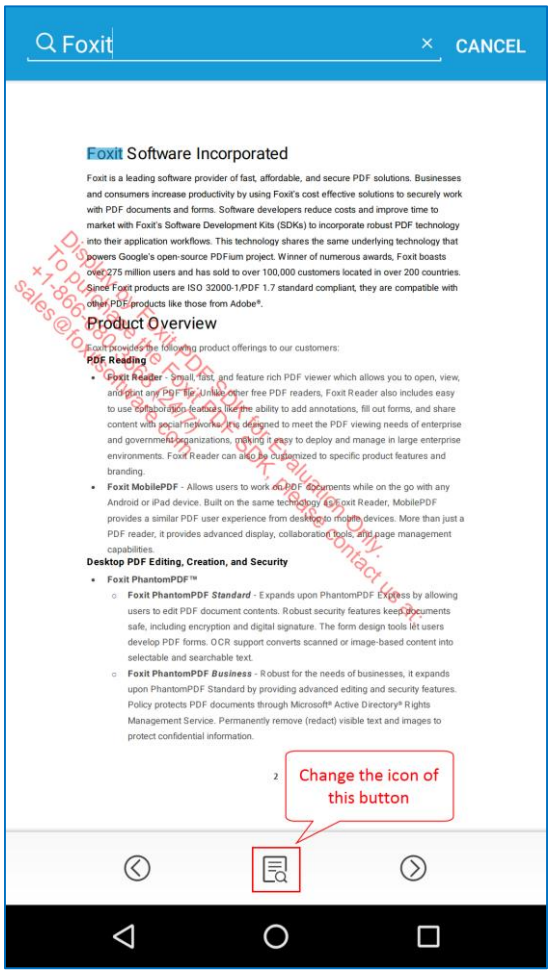


Figure 4-9

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.

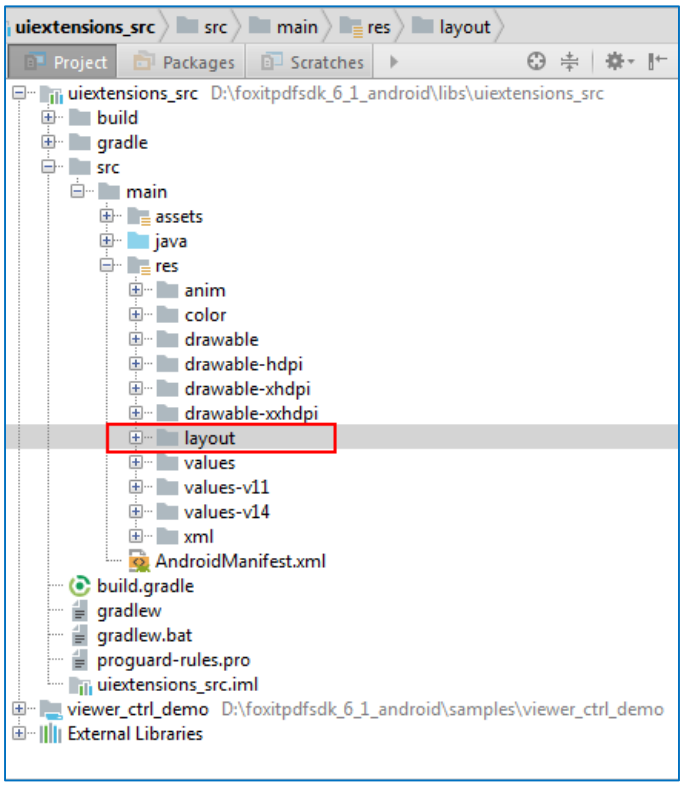


Figure 4-10

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.

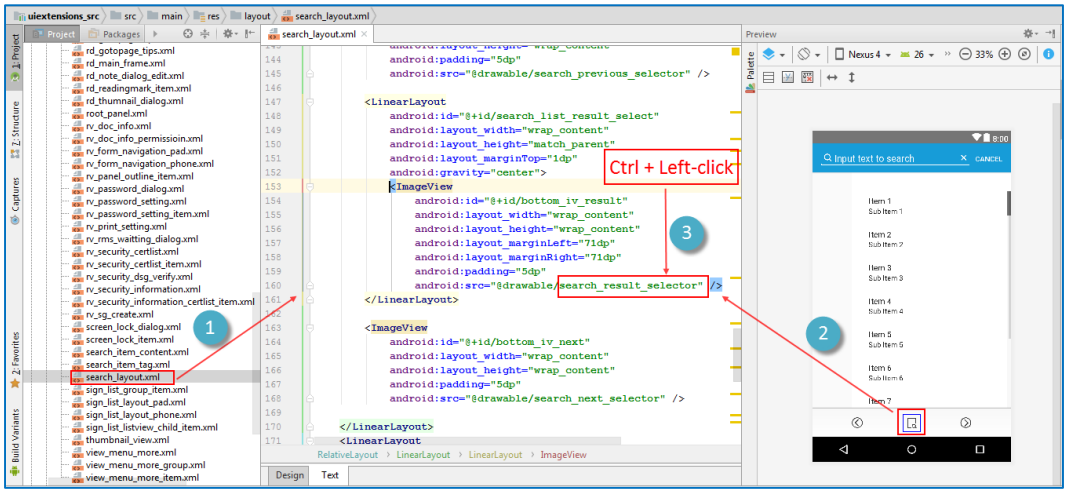


Figure 4-11

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 PDF SDK for Android provides three sets of icons for devices with different DPI requirements to make sure that your apps can run smoothly on every device.*

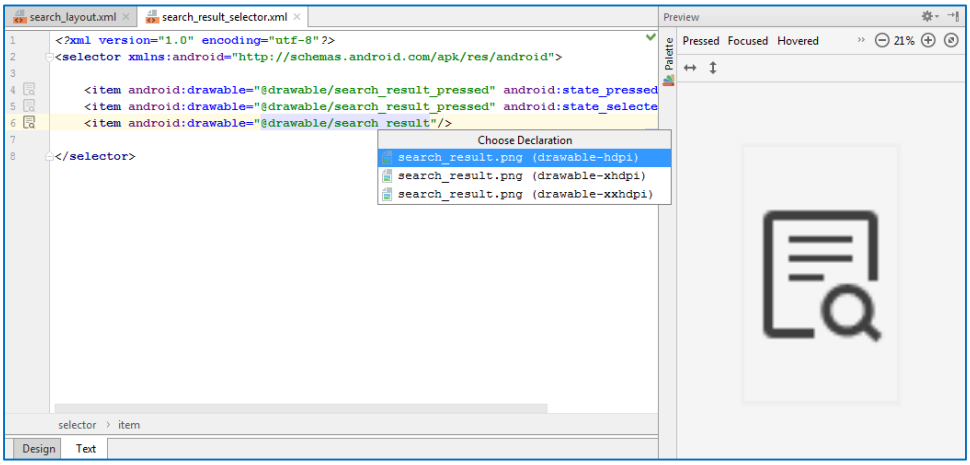


Figure 4-12

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.

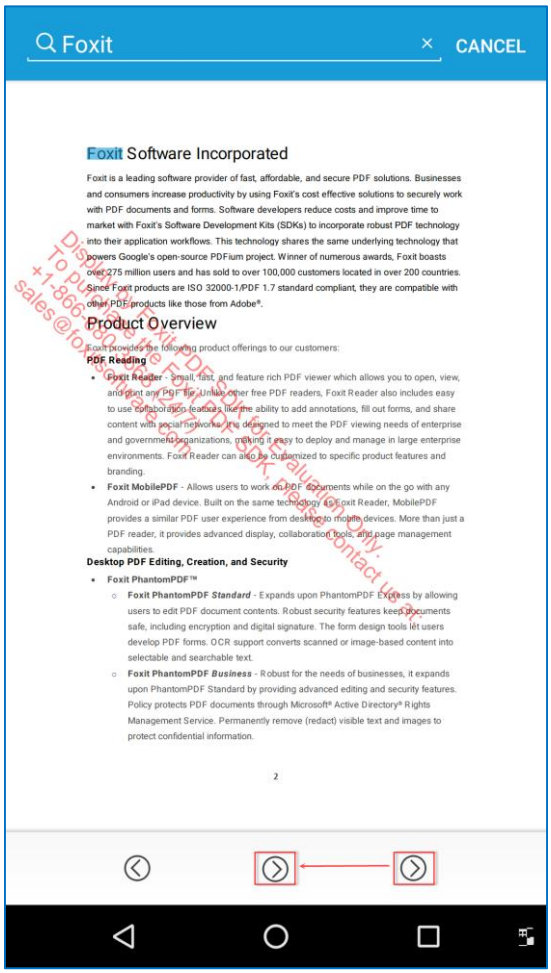


Figure 4-13

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

5 Working with SDK API

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

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

5.1 Render

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

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

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

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

Example:

5.1.1 How to render a specified page to a bitmap

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

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

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

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

5.2 Text Page

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

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

Example:

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

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

// Get the text area on page by selection. The starting selection position and ending
selection position are specified by startPos and endPos.
public ArrayList<RectF> getTextRectsBySelection(PDFPage page, PointF startPos, PointF endPos)
{
    try {
        // If the page hasn't been parsed yet, throw an exception.
        if (page.isParsed()) {
            throw new PDFException(Constants.e_ErrNotParsed, "PDF Page should be parsed
first");
        }

        // Create a text page from the parsed PDF page.
        TextPage textPage = new TextPage(page, TextPage.e_ParseTextNormal);
        if (textPage == null || textPage.isEmpty()) return null;
        int startCharIndex = textPage.getIndexAtPos(startPos.getX(), startPos.getY(), 5);
        int endCharIndex = textPage.getIndexAtPos(endPos.getX(), endPos.getY(), 5);
```

```
        // API getTextRectCount requires that start character index must be lower than or
equal to end character index.
        startCharIndex = startCharIndex < endCharIndex ? startCharIndex : endCharIndex;
        endCharIndex = endCharIndex > startCharIndex ? endCharIndex : startCharIndex;
        int count = textPage.getTextRectCount(startCharIndex, endCharIndex - startCharIndex);

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

5.3 Text Search

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

- To specify the searching pattern and options, use functions TextSearch.setPattern, TextSearch.setStartPage (only useful for a text search in PDF document), TextSearch.setEndPage (only useful for a text search in PDF document) and TextSearch.setSearchFlags.
- To do the searching, use function TextSearch.findNext or TextSearch.findPrev.
- To get the searching result, use function TextSearch.getMatchXXX().

Example:

5.3.1 How to search a text pattern in a PDF

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

```
import com.foxit.sdk.common.fxcrt.PointF;
import com.foxit.sdk.pdf.TextSearch;
...

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

5.4 Bookmark (Outline)

Foxit PDF SDK provides navigational tools called Bookmarks to allow users to quickly locate and link their point of interest within a PDF document. PDF bookmark is also called outline, and each bookmark contains a destination or actions to describe where it links to. It is a tree-structured hierarchy, so function PDFDoc.getRootBookmark must be called first to get the root of the whole bookmark tree before accessing to the bookmark tree. Here, “root bookmark” is an abstract object which can only have some child bookmarks without next sibling bookmarks and any data (includes bookmark data, destination data and action data). It cannot be shown on the application UI since it has no data. Therefore, a root bookmark can only call function Bookmark.getFirstChild.

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

- To access the parent bookmark, use function Bookmark.getParent.

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

Example:

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

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.Bookmark;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.actions.Destination;
...
private void DepthFistTravelBookmarkTree(Bookmark bookmark, PDFDoc doc) {
    if(bookmark == null || bookmark.isEmpty())
        return;
    try {
        DepthFistTravelBookmarkTree(bookmark.getFirstChild(), doc);
        while (true) {
            // Get bookmark title.
            String title = bookmark.getTitle();
            Destination dest = bookmark.getDestination();
            if (dest != null && !dest.isEmpty())
            {
                float left, right, top, bottom;
                float zoom;
                int pageIndex = dest.getPageIndex(doc);
                // left,right,top,bottom,zoom are only meaningful with some special zoom
modes.
                int mode = dest.getZoomMode();
                switch (mode) {
                    case Destination.e_ZoomXYZ:
                        left = dest.getLeft();
                        top = dest.getTop();
                        zoom = dest.getZoomFactor();
                        break;
                    case Destination.e_ZoomFitPage:
                        break;
                    case Destination.e_ZoomFitHorz:
                        top = dest.getTop();
                        break;
                    case Destination.e_ZoomFitVert:
                        left = dest.getLeft();
```

```
                    break;
                case Destination.e_ZoomFitRect:
                    left = dest.getLeft();
                    bottom = dest.getBottom();
                    right = dest.getRight();
                    top = dest.getTop();
                    break;
                case Destination.e_ZoomFitBBox:
                    break;
                case Destination.e_ZoomFitBHorz:
                    top = dest.getTop();
                    break;
                case Destination.e_ZoomFitBVert:
                    left = dest.getLeft();
                    break;
                default:
                    break;
            }
        }
        bookmark = bookmark.getNextSibling();
        if (bookmark == null || bookmark.isEmpty())
            break;
        DepthFistTravelBookmarkTree(bookmark.getFirstChild(), doc);
    }
}
catch (Exception e) {
}
}
```

5.5 Reading Bookmark

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

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

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

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

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

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

5.6 Attachment

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

Example:

5.6.1 How to embed a specified file to a PDF document

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.Attachments;
import com.foxit.sdk.pdf.FileSpec;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.objects.PDFNameTree;
...
try {
    String pdfpath = "XXX/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
    // Embed the specified file to PDF document.
    String filePath = "/xxx/fileToBeEmbedded.xxx";
    PDFNameTree nameTree = new PDFNameTree(doc, PDFNameTree.e_EmbeddedFiles);
    Attachments attachments = new Attachments(doc, nameTree);
    FileSpec fileSpec = new FileSpec(doc);
    fileSpec.setFileName(filePath);
    if (!fileSpec.embed(filePath)) return;
    attachments.addEmbeddedFile(filePath, fileSpec);
} catch (PDFException e) {
    e.printStackTrace();
}
...
```

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

```
import com.foxit.sdk.PDFException;
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.pdf.Attachments;
import com.foxit.sdk.pdf.FileSpec;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.objects.PDFNameTree;
...
try {
    String pdfpath = "XXX/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
    PDFNameTree nameTree = new PDFNameTree(doc, PDFNameTree.e_EmbeddedFiles);
    Attachments attachments = new Attachments(doc, nameTree);
    // Extract the embedded attachment file.
    int count = attachments.getCount();
    for (int i = 0; i < count; i ++) {
        String key = attachments.getKey(i);
```

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

5.7 Annotation

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

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

Table 5-1

Annotation type	Description	Markup	Supported by SDK
Text(Note)	Text annotation	Yes	Yes
Link	Link Annotation	No	Yes
FreeText (TypeWriter/TextBox/Callout)	Free text annotation	Yes	Yes
Line	Line annotation	Yes	Yes
Square	Square annotation	Yes	Yes
Circle	Circle annotation	Yes	Yes
Polygon	Polygon annotation	Yes	Yes
PolyLine	PolyLine annotation	Yes	Yes
Highlight	Highlight annotation	Yes	Yes
Underline	Underline annotation	Yes	Yes
Squiggly	Squiggly annotation	Yes	Yes
StrikeOut	StrikeOut annotation	Yes	Yes
Stamp	Stamp annotation	Yes	Yes
Caret	Caret annotation	Yes	Yes

Annotation type	Description	Markup	Supported by SDK
Ink(pencil)	Ink annotation	Yes	Yes
Popup	Popup annotation	No	Yes
File Attachment	FileAttachment annotation	Yes	Yes
Sound	Sound annotation	Yes	No
Movie	Movie annotation	No	No
Widget*	Widget annotation	No	Yes
Screen	Screen annotation	No	Yes
PrinterMark	PrinterMark annotation	No	No
TrapNet	Trap network annotation	No	No
Watermark*	Watermark annotation	No	No
3D	3D annotation	No	No
Redact	Redact annotation	Yes	Yes

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

Example:

5.7.1 How to add annotations to a PDF page

```
import com.foxit.sdk.PDFViewCtrl;
import com.foxit.sdk.common.Constants;
import com.foxit.sdk.common.fxcrt.RectF;
import com.foxit.sdk.common.fxcrt.RectFArray;
import com.foxit.sdk.pdf.PDFDoc;
import com.foxit.sdk.pdf.PDFPage;
import com.foxit.sdk.pdf.TextSearch;
import com.foxit.sdk.pdf.annots.Annot;
import com.foxit.sdk.pdf.annots.Note;
import com.foxit.sdk.pdf.annots.QuadPoints;
import com.foxit.sdk.pdf.annots.QuadPointsArray;
import com.foxit.sdk.pdf.annots.TextMarkup;
import com.foxit.sdk.common.fxcrt.PointF;
...
// Add text annot.
try {
    String pdfpath = "xxx/Sample.pdf";
    PDFDoc doc = new PDFDoc(pdfpath);
    PDFPage pdfPage = doc.getPage(1);
    RectF rect = new RectF(100, 100, 120, 120);
    Note note = new Note(pdfPage.addAnnot(Annot.e_Note, rect));
```

```
    if (note == null || note.isEmpty()){
        return;
    }
    note.setIconName("Comment");
    // Set color to blue.
    note.setBorderColor(0xff0000ff);
    note.setContent("This is the note comment, write any content here.");
    note.resetAppearanceStream();

    // The following code demonstrates how to add hightlight annotation on the searched text.
    TextSearch textSearch = new TextSearch(pdfPage.getDocument(),null);
    if (textSearch == null || textSearch.isEmpty()){
        return;
    }
    // Suppose that the text for highlighting is "foxit".
    textSearch.setPattern("foxit");
    boolean bMatched = textSearch.findNext();
    if (bMatched) {
        RectFArray rects = textSearch.getMatchRects();
        int rectCount = rects.getSize();
        // Fill the quadpoints array according to the text rects of matched result.
        QuadPointsArray arrayOfQuadPoints = new QuadPointsArray();
        for (int i = 0; i < rectCount; i++) {
            rect = rects.getAt(i);
            QuadPoints quadPoints = new QuadPoints();
            PointF point = new PointF();
            point.set(rect.getLeft(), rect.getTop());
            quadPoints.setFirst(point);
            point.set(rect.getRight(), rect.getTop());
            quadPoints.setSecond(point);
            point.set(rect.getLeft(), rect.getBottom());
            quadPoints.setThird(point);
            point.set(rect.getRight(), rect.getBottom());
            quadPoints.setFourth(point);
            arrayOfQuadPoints.add(quadPoints);
        }
        // Just set an empty rect to markup annotation, the annotation rect will be calculated
according to the quadpoints that set to it later.
        rect = new RectF(0, 0, 0, 0);
        TextMarkup textMarkup = new TextMarkup(pdfPage.addAnnot(Annot.e_Highlight, rect));
        // Set the quadpoints to this markup annot.
        textMarkup.setQuadPoints(arrayOfQuadPoints);
        // set to red.
        textMarkup.setBorderColor(0xffff0000);
        // set to thirty-percent opacity.
        textMarkup.setOpacity(0.3f);
```

```
        // Generate the appearance.
        textMarkup.resetAppearanceStream();
    }
} catch (Exception e) {}
```

5.7.2 How to delete annotations in a PDF page

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

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

5.8 Form

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

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

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

Example:

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

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

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

5.9 Security

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

Example:

5.9.1 How to encrypt a PDF file with password

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

// Encrypt the source pdf document with specified owner password and user password, the
encrypted PDF will be saved to the path specified by parameter savePath.
public boolean encryPDF(PDFDoc pdfDoc, byte[] ownerPassword, byte[] userPassword, String
savePth) {
```

```
    if (pdfDoc == null || (ownerPassword == null && userPassword == null) || savePth == null)
        return false;
    // The encryption setting data. Whether to encrypt meta data:true, User permission:
modify,assemble,fill form. Cipher algorithm:AES 128.
    StdEncryptData encryptData = new StdEncryptData(true,
            PDFDoc.e_PermModify | PDFDoc.e_PermAssemble | PDFDoc.e_PermFillForm,
            SecurityHandler.e_CipherAES, 16);

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

5.10 Signature

PDF Signature can be used to create and sign digital signatures for PDF documents, which protects the security of documents' contents and avoids it to be tampered maliciously. It can let the receiver make sure that the document is released by the signer and the contents of the document are complete and unchanged. Foxit PDF SDK provides APIs to create digital signature, verify the validity of signature, delete existing digital signature, get and set properties of digital signature, display signature and customize the appearance of the signature form fields.

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

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

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

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

Example:

5.10.1 How to sign a PDF document and verify the signature

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

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

        // The input PKCS#12 format certificate, which contains the public and private keys.
        String certPath = "/somewhere/cert.pfx";
        // Password for that certificate.
        byte[] certPassword = "123".getBytes();
        String signedPDFPath = "/somewhere/signed.pdf";
        // Start to sign the signature, if everything goes well, the signed PDF will be saved
```

```
to the path specified by "save_path".
        Progressive progressive = signature.startSign(certPath, certPassword,
Signature.e_DigestSHA1, signedPDFPath, null, null);
        if (progressive != null) {
            int state = Progressive.e_ToBeContinued;
            while (state == Progressive.e_ToBeContinued) {
                state = progressive.resume();
            }
            if (state != Progressive.e_Finished) return;
        }

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

6 Creating a custom tool

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

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

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

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

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

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

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

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

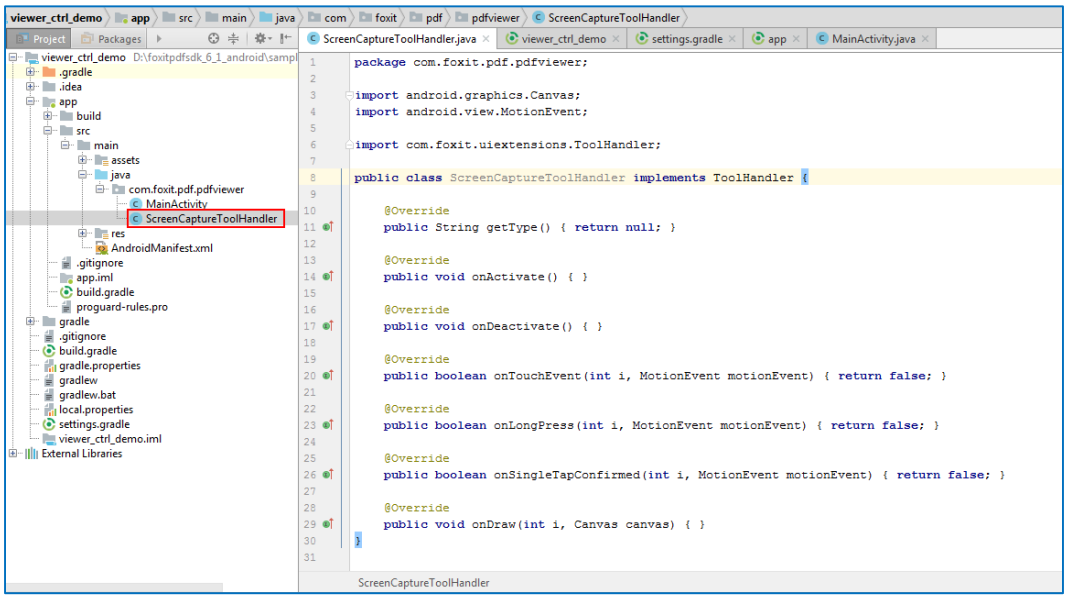


Figure 6-1

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

Update **ScreenCaptureToolHandler.java** as follows:

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

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

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

import java.io.File;
```

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

public class ScreenCaptureToolHandler implements ToolHandler {

    private Context mContext;
    private PDFViewCtrl mPdfViewCtrl;

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

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

    @Override
    public void onActivate() {

    }

    @Override
    public void onDeactivate() {

    }

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

    // Handle OnTouch event
    @Override
    public boolean onTouchEvent(int pageIndex, MotionEvent motionEvent) {

        // Get the display view point in device coordinate system
        PointF devPt = new PointF(motionEvent.getX(), motionEvent.getY());
        PointF point = new PointF();
        // Convert display view point to page view point.
        mPdfViewCtrl.convertDisplayViewPtToPageViewPt(devPt, point, pageIndex);
        float x = point.x;
```

```
        float y = point.y;

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

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

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

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

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

            // Save the selected area as a bitmap.
            case MotionEvent.ACTION_UP:
                if (mLastPageIndex != pageIndex)
                    break;
                if (!mStartPoint.equals(mEndPoint.x, mEndPoint.y)) {
                    renderToBmp(pageIndex, "/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 bimap to a specified path.
    public static void saveBitmap(Bitmap bm, String outPath) throws IOException {
        File file = new File(outPath);
        file.createNewFile();

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

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

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

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

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

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

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

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

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

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

    @Override
    public boolean onLongPress(int 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 8.1. Please make sure you have pushed the "Sample.pdf" document into the correct folder (which matches the path in the demo) of the emulator's SD card.*

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

Figure 6-2

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


Product Overview
www.foxitsoftware.com
The selected area was saved as a bitmap stored in the /mnt/sdcard/ScreenCapture.bmp

Figure 6-3

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

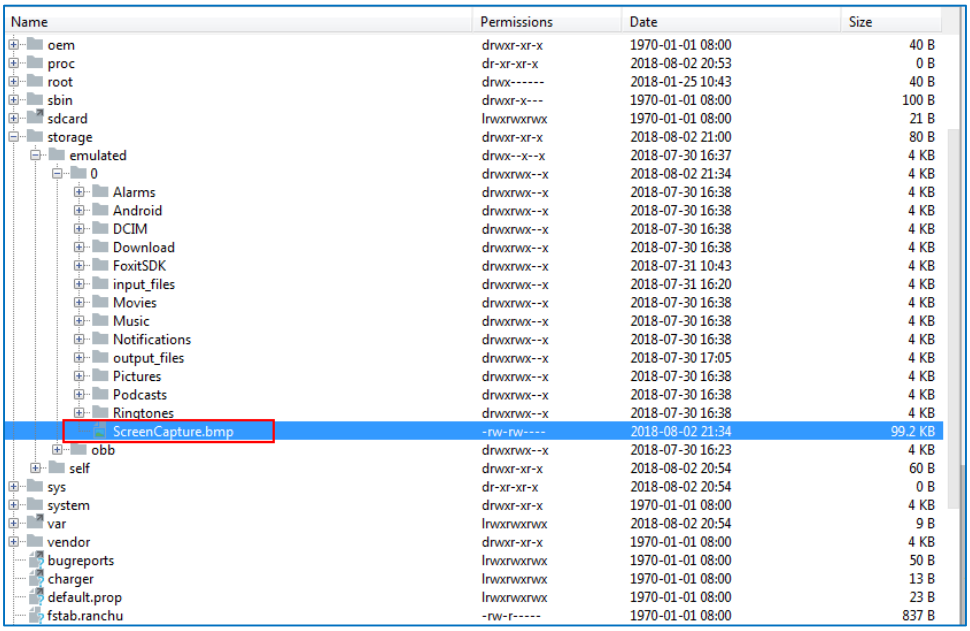


Figure 6-4

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

Figure 6-5

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

7 Implement Foxit PDF SDK for Android using Cordova

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

This section will help you get started with Foxit PDF SDK for Android and the Cordova plugin on Windows. For other operating systems, you can take this tutorial as reference.

7.1 System Requirements

- NPM
- Cordova 8.1.2
- Android SDK
- JDK 1.8
- Foxit PDF SDK For Android

***Note**: The version of Foxit PDF SDK for Android should match the version of the 'cordova-plugin-foxitpdf' plugin. You can specify the plugin version when you install it. Otherwise, the latest version will be installed.*

7.2 Install Cordova Command-line Tool

Please refer to the Apache Cordova website to install the Cordova command-line tool.

- Download Node.js and then install it.
- Open a terminal, and type npm install -g cordova.

7.3 Build a Cordova project using Foxit PDF SDK for Android

7.3.1 Create a Cordova project

Open a command prompt or terminal, navigate to the directory where you wish to create your project and type cordova create <path> command. For example, navigate to "D:\cordova", and type the command below to create a cordova project:

```
cordova create test_cordova com.app Test_Cordova
```

7.3.2 Add Platforms

After creating a Cordova project, navigate to the project directory (D:\cordova\test_cordova), and type the command below to add Android platform to build your app.

```
cd test_cordova
cordova platform add android
```

7.3.3 Install 'cordova-plugin-foxitpdf' plugin

To install 'cordova-plugin-foxitpdf' plugin, choose one of the following ways:

- Download the plugin from npm and install it inside the project folder:

  // Install a specific plugin version, for example the 6.3.0 version:

  ```
  cordova plugin add cordova-plugin-foxitpdf@6.3.0
  ```

  // Install the latest plugin version (just not specify the version):

  ```
  cordova plugin add cordova-plugin-foxitpdf
  ```

- Install it via repo url directly (generally, in this way, the latest plugin version will be installed):

  ```
  cordova plugin add https://github.com/foxitsoftware/cordova-plugin-foxitpdf.git
  ```

7.3.4 Integrate Foxit PDF SDK for Android

Download Foxit PDF SDK for Android package and extract it. If you use the plugin version before 6.3, please follow the Plugin version before 6.3 usage, for plugin version 6.3, follow the Plugin verison 6.3 usage, and for plugin version 6.3.1 or higher, follow the Plugin verison 6.3.1 usage or higher.

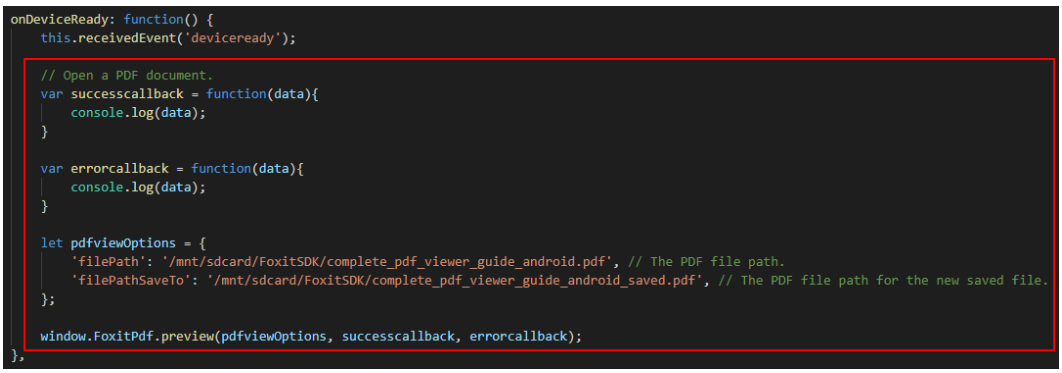
Note: *In this section, we only introduce how to use the Cordova plugin to open a PDF document. For more APIs about the plugin, please refer to the webpage https://github.com/foxitsoftware/cordova-plugin-foxitpdf.*

Plugin version before 6.3 usage

1) Copy "libs" folder from the extracted package to "D:\cordova\test_cordova\platforms\android" directory.

2) Replace the license string ('sn' and 'key') in "FoxitPdf.java" class (under "test_cordova\platforms\android\app\src\main\java\com\foxit\cordova\plugin"). The license files "rdk_sn.txt" and "rdk_key.txt" can be found in the "libs" folder of Foxit PDF SDK for Android package.

3) Add the following code (See Figure 7-1) to "test_cordova\www\js\index.js" file to open a PDF document.



```
onDeviceReady: function() {
    this.receivedEvent('deviceready');

    // Open a PDF document.
    var successcallback = function(data){
        console.log(data);
    }

    var errorcallback = function(data){
        console.log(data);
    }

    let pdfviewOptions = {
        'filePath': '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android.pdf', // The PDF file path.
        'filePathSaveTo': '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android_saved.pdf', // The PDF file path for the new saved file.
    };

    window.FoxitPdf.preview(pdfviewOptions, successcallback, errorcallback);
},
```

Figure 7-1

```
var successcallback = function(data){
  console.log(data);
};

var errorcallback = function(data){
  console.log(data);
};

let pdfviewOptions = {
  'filePath': '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android.pdf', // The PDF
file path.
  'filePathSaveTo': '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android_save.pdf',
// The PDF file path for the new saved file.
};

window.FoxitPdf.preview(pdfviewOptions, successcallback, errorcallback);
```

Note: *Here, we assume that you have pushed the "complete_pdf_viewer_guide_android.pdf" document found in the "samples/test_files" folder of Foxit PDF SDK for Android package into the "FoxitSDK" folder of the Android device or emulator that will be used to run this project. Certainly, you can change the file path with your own files.*

4) Deploy the files in "test_cordova\www" to the "www" folder of the platform (test_cordova\platforms\android\app\src\main\assets\www). In a terminal, go to the project directory, type the command below:

```
cordova prepare android
```

Plugin version 6.3 usage

1) Copy "libs" folder from the extracted package to "D:\cordova\test_cordova\platforms\android" directory.

2) Initialize SDK libraries. The license files "rdk_sn.txt" and "rdk_key.txt" can be found in the "libs" folder of Foxit PDF SDK for Android package. Add the initialization code (See Figure 7-2) to "test_cordova\www\js\index.js" file.

```
onDeviceReady: function() {
    this.receivedEvent('deviceready');

    // Initialize SDK libraries.
    var success = function(data){
        console.log(data);
    }
    var error = function(data){
        console.log(data);
    }
    let initOptions = {
        'foxit_sn': 'xxx', // The value can be found in the "rdk_sn.txt" (the string after "SN=").
        'foxit_key': 'xxx', // The value can be found in the "rdk_key.txt" (the string after "Sign=").
    };
    window.FoxitPdf.initialize(initOptions, success, error);
},
```

Figure 7-2

```
var success = function(data){
  console.log(data);
}
var error = function(data){
  console.log(data);
}
let initOptions = {
  'foxit_sn': 'xxx', // The value can be found in the "rdk_sn.txt" (the string after "SN=").
  'foxit_key': 'xxx', // The value can be found in the "rdk_key.txt" (the string after "Sign=").
};
window.FoxitPdf.initialize(initOptions, success, error);
```

3) Add the following code (See Figure 7-3) to open a PDF document in "test_cordova\www\js\index.js" file.

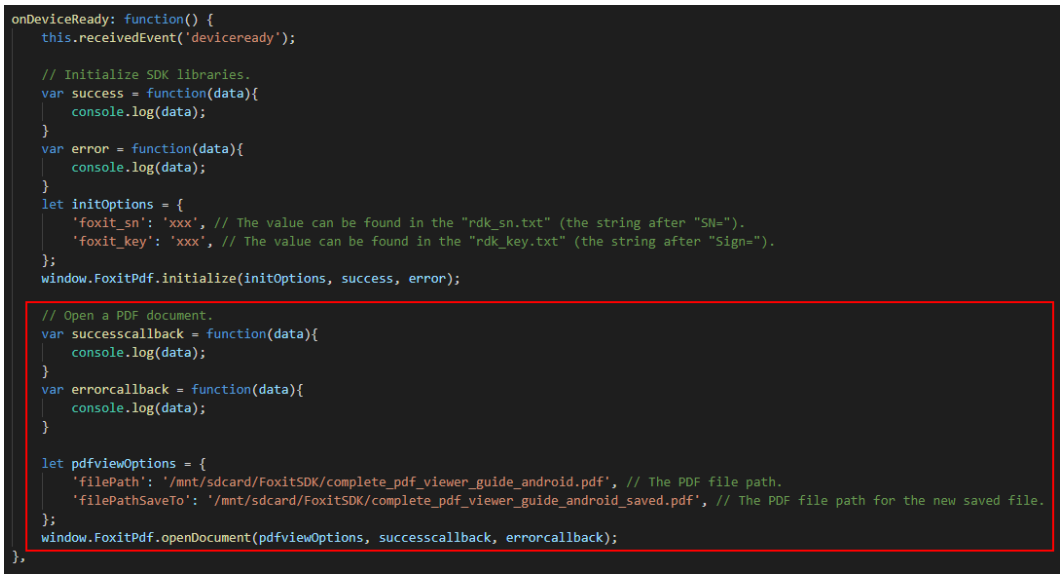


Figure 7-3

```
var successcallback = function(data){
  console.log(data);
};

var errorcallback = function(data){
  console.log(data);
};

let pdfviewOptions = {
  'filePath': '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android.pdf', // The PDF file path.
  'filePathSaveTo': '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android_save.pdf', // The PDF file path for the new saved file.
};

window.FoxitPdf.openDocument(pdfviewOptions, successcallback, errorcallback);
```

Note:

- *In plugin version 6.3 or higher, the usage of the new interface **window.FoxitPdf.openDocument** is the same with the **window.FoxitPdf.preview**, but we recommend using **window.FoxitPdf.openDocument**, because **window.FoxitPdf.preview** will be deprecated in the future. Before calling **window.FoxitPdf.openDocument**, you should call **window.FoxitPdf.initialize** to initialize the SDK libraries at first.*

- *Here, we assume that you have pushed the "complete_pdf_viewer_guide_android.pdf" document found in the "samples/test_files" folder of Foxit PDF SDK for Android package into the "FoxitSDK" folder of the Android device or emulator that will be used to run this project. Certainly, you can change the file path with your own files.*

4) Deploy the files in "test_cordova\www" to the "www" folder of the platform (test_cordova\platforms\android\app\src\main\assets\www). In a terminal, go to the project directory, type the command below:

```
cordova prepare android
```

Plugin version 6.3.1 usage or higher

1) Copy "libs" folder from the extracted package to "D:\cordova\test_cordova\platforms\android" directory.

2) Initialize SDK libraries. The license files "rdk_sn.txt" and "rdk_key.txt" can be found in the "libs" folder of Foxit PDF SDK for Android package. Add the initialization code (See Figure 7-6) to "test_cordova\www\js\index.js" file.

```
onDeviceReady: function() {
    this.receivedEvent('deviceready');

    var foxit_sn = 'xxx'; // The value can be found in the "rdk_sn.txt" (the string after "SN=").
    var foxit_key = 'xxx'; // The value can be found in the "rdk_key.txt" (the string after "Sign=").

    window.FoxitPdf.initialize(foxit_sn, foxit_key);

},
```

Figure 7-4

```
var foxit_sn = 'xxx'; // The value can be found in the "rdk_sn.txt" (the string after "SN=").
var foxit_key = 'xxx'; // The value can be found in the "rdk_key.txt" (the string after "Sign=").

window.FoxitPdf.initialize(foxit_sn, foxit_key);
```

3) Add the following code (See Figure 7-5) to open a PDF document in "test_cordova\www\js\index.js" file.

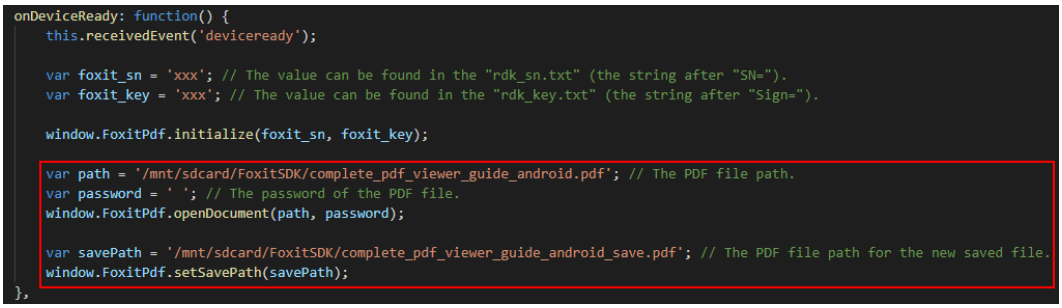


Figure 7-5

```
var path = '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android.pdf'; // The PDF file path.
var password = ' '; // The password of the PDF file.
window.FoxitPdf.openDocument(path, password);

var savePath = '/mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android_save.pdf'; // The PDF file path for the new saved file.
window.FoxitPdf.setSavePath(savePath);
```

Note: *Here, we assume that you have pushed the "complete_pdf_viewer_guide_android.pdf" document found in the "samples/test_files" folder of Foxit PDF SDK for Android package into the "FoxitSDK" folder of the Android device or emulator that will be used to run this project. Certainly, you can change the file path with your own files.*

4) Deploy the files in "test_cordova\www" to the "www" folder of the platform (test_cordova\platforms\android\app\src\main\assets\www). In a terminal, go to the project directory, type the command below:

```
cordova prepare android
```

7.3.5 Run the project

To run the project, first setup your device or emulator to deploy the project to, and then you can use the following command or run it in Android Studio directly.

Navigate to the project directory, type the command as below:

```
cordova run android                 // for device
cordova run android --emulator      // for emulator
```

Note: *If you encounter the problem "AAPT: error: resource android: attr/fontVariationSettings not found", please add the following code to "test_cordova\platforms\android\app\build.gradle" (just copy and paste the code to the end line of the "build.gradle" file), or you can upgrade the Android SDK to 28 or higher.*

```
configurations.all {
    resolutionStrategy {
        force 'com.android.support:support-v4:27.1.0'
    }
}
```

After running the project successfully, the "complete_pdf_viewer_guide_android.pdf" document will be displayed as shown in Figure 7-6.

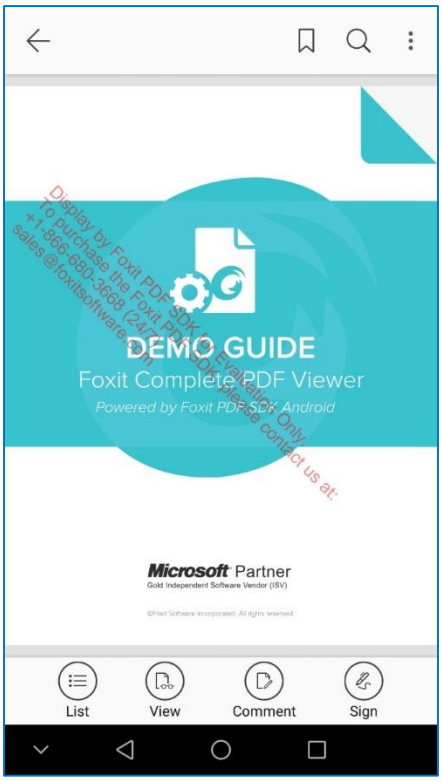


Figure 7-6

7.3.6 Customize the UI

You can customize the UI including the feature modules, rights management and the properties of UI elements through a JSON file called "uiextensions_config.json" which is located in the "test_cordova\platforms\android\app\src\main\assets\www\plugins\cordova-plugin-foxitpdf" folder.

To customize the UI, just modify the JSON file as desired, and then rerun the project. For more detailed information about the JSON file, please refer to section 4.1.2 Configuration Items Description.

***Note**: If you didn't find the "uiextensions_config.json" in the* "test_cordova\platforms\android\app\src\main\assets\www\plugins\cordova-plugin-foxitpdf" folder,

you can copy the JSON file from "test_cordova\node_modules\cordova-plugin-foxitpdf\src\android\assets" to that place manually.

8 Implement Foxit PDF SDK for Android using React Native

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

This section will help you get started with Foxit PDF SDK for Android and the React Native plugin on Windows. For other operating systems, you can take this tutorial as reference.

8.1 System Requirements

- NPM 8.3 or newer
- React-Native
- Android SDK
- Android Gradle plugin 3.1.0 or higher
- Gradle 4.4 or higher
- JDK 1.8
- Foxit PDF SDK for Android

***Note**: The version of Foxit PDF SDK for Android should match the version of the 'react-native-foxitpdf' plugin. You can specify the plugin version when you install it. Otherwise, the latest version will be installed.*

8.2 Install React Native Command Line Interface

To build React Native app for Android, you will need Node, the React Native command line interface, a Java SE Development Kit (JDK) and Android Studio.

Please refer to the official getting started guide on setting up React Native CLI and setting up Android development environment to install React Native command line interface and configure Android development environment.

8.3 Build a React Native project using Foxit PDF SDK for Android

8.3.1 Create a React Native project

Open a command prompt or terminal, navigate to the directory where you wish to create your project and type react-native init <ProjectName> command. For example, navigate to "D:\react-native", and type the command below to create a React Native project called "testRN":

```
react-native init testRN
```

8.3.2 Install 'react-native-foxitpdf' plugin

Download the plugin from npm and install it inside the project folder. You can install a specific plugin version or install the latest plugin version as below:

- // Install a specific plugin version, for example the 6.3.0 version:

```
cd testRN

npm install @foxitsoftware/react-native-foxitpdf@6.3.0 --save
```

- // Install the latest plugin version (by not specifying the version):

```
cd testRN

npm install @foxitsoftware/react-native-foxitpdf --save
```

Link the project to the plugin:

```
react-native link @foxitsoftware/react-native-foxitpdf
```

Note: *All the subsequent commands are executed in the project directory.*

8.3.3 Integrate Foxit PDF SDK for Android

Download Foxit PDF SDK for Android package and extract it. If you use the version before 6.3, please follow the Plugin version before 6.3 usage, otherwise follow the Plugin verison 6.3 or higher usage.

Plugin version before 6.3 usage

1) Copy "libs" folder from the extracted package to "D:\react-native\testRN\android" directory.

2) Add the following code into the project-level build.gradle file (testRN\android\build.gradle):

   Inside the '`allprojects`', add the code:

```
allprojects {
    repositories {
        mavenLocal()
```

```
        google()
        jcenter()
        maven {
            // All of React Native (JS, Obj-C sources, Android binaries) is installed
from npm
            url "$rootDir/../node_modules/react-native/android"
        }
        flatDir {
        dirs project(':@foxitsoftware_react-native-foxitpdf').file("$rootDir/libs")
        }
    }
}
```

3) Add the license information into "testRN\android\gradle.properties" file.

```
FOXIT_LICENSE_SN=RDK_SN
FOXIT_LICENSE_KEY=RDK_KEY
```

The value of "RDK_SN" and "RDK_KEY" can be found in the "rdk_sn.txt" and "rdk_key.txt" files under the "libs" folder of Foxit PDF SDK for Android package.

4) In "testRN\android\app\build.gradle" file, add the following code:

Inside the 'defaultConfig', add the code:

```
defaultConfig {
    applicationId "com.testrn"
    minSdkVersion rootProject.ext.minSdkVersion
    targetSdkVersion rootProject.ext.targetSdkVersion
    versionCode 1
    versionName "1.0"
    ndk {
        abiFilters "armeabi-v7a", "x86"
    }
    manifestPlaceholders = [foxit_license_sn:FOXIT_LICENSE_SN,
foxit_license_key:FOXIT_LICENSE_KEY]
}
```

5) In "testRN\android\app\src\main\AndroidManifest.xml" file, add the required permissions, declare PDFReaderActivity, and add license placeholder.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.testrn">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RUN_INSTRUMENTATION" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />

    <application
        android:name=".MainApplication"
```

```
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:allowBackup="false"
        android:theme="@style/AppTheme"
        tools:replace="android:allowBackup,icon,theme,label,name">

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.facebook.react.devsupport.DevSettingsActivity" />

        <meta-data
            android:name="foxit_sn"
            android:value="${foxit_license_sn}" />
        <meta-data
            android:name="foxit_key"
            android:value="${foxit_license_key}" />

        <activity
            android:name="com.foxitreader.PDFReaderActivity"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
            android:screenOrientation="fullSensor" />

    </application>

</manifest>
```

6) Use the plugin to open a PDF document.

In the "testRN\App.js" file, you can import the plugin using the following code:

```
import FoxitPDF from '@foxitsoftware/react-native-foxitpdf';
```

Then, call the function below to open a PDF document:

```
FoxitPDF.openPDF('a PDF file path');
```

Update the "App.js" file as below, which adds a button to open a PDF document:

```
import React, { Component } from 'react';
import { Platform, StyleSheet, Text, View, TouchableOpacity } from 'react-native';
import FoxitPDF from '@foxitsoftware/react-native-foxitpdf';

const instructions = Platform.select({
  ios: 'Press Cmd+R to reload,\n' + 'Cmd+D or shake for dev menu',
  android:
    'Double tap R on your keyboard to reload,\n' +
    'Shake or press menu button for dev menu',
});

type Props = {};
```

```
export default class App extends Component<Props> {
  onPress() {
    FoxitPDF.openPDF('mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android.pdf');
  }

  render() {
    return (
      <View style={styles.container}>
        <TouchableOpacity
          style={styles.button}
          onPress={this.onPress}
        >
          <Text> Open PDF </Text>
        </TouchableOpacity>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  button: {
    alignItems: 'center',
    backgroundColor: '#DDDDDD',
    padding: 10
  },
});
```

Note: *Here, we assume that you have pushed the "complete_pdf_viewer_guide_android.pdf" document found in the "samples/test_files" folder of Foxit PDF SDK for Android package into the "FoxitSDK" folder of the Android device or emulator that will be used to run this project. Certainly, you can change the file path with your own files.*

Plugin version 6.3 or higher usage

1) Copy "libs" folder from the extracted package to "D:\react-native\testRN\android" directory.

2) Add the following code into the project-level build.gradle file (testRN\android\build.gradle):

   Inside the '`allprojects`', add the code:

```
allprojects {
    repositories {
        mavenLocal()
        google()
        jcenter()
        maven {
            // All of React Native (JS, Obj-C sources, Android binaries) is installed
from npm
            url "$rootDir/../node_modules/react-native/android"
        }
```

```
        flatDir {
        dirs project(':@foxitsoftware_react-native-foxitpdf').file("$rootDir/libs")
        }
    }
}
```

3) In "testRN\android\app\src\main\AndroidManifest.xml" file, add the required permissions and declare PDFReaderActivity.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.testrn">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RUN_INSTRUMENTATION" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />

    <application
        android:name=".MainApplication"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:allowBackup="false"
        android:theme="@style/AppTheme"
        tools:replace="android:allowBackup,icon,theme,label,name">

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.facebook.react.devsupport.DevSettingsActivity" />

        <activity
            android:name="com.foxitreader.PDFReaderActivity"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
            android:screenOrientation="fullSensor" />

    </application>

</manifest>
```

4) Use the plugin to open a PDF document.

In the "testRN\App.js" file, you can import the plugin using the following code:

```
import FoxitPDF from '@foxitsoftware/react-native-foxitpdf';
```

Call the function below to initialize SDK libraries:

```
FoxitPDF.initialize("foxit_sn", "foxit_key");
```

Note: *The "foxit_sn" can be found in the* "rdk_sn.txt" file (the string after "SN="), and the "foxit_key" can be found in the "rdk_key.txt" file (the string after "Sign="). The license files "rdk_sn.txt" and "rdk_key.txt" is in the "libs" folder of Foxit PDF SDK for Android package.

Then, call the following funciton to open a PDF document:

```
FoxitPDF.openDocument('a PDF file path');
```

Update the "App.js" file as below, which adds a button to open a PDF document:

```
import React, { Component } from 'react';
import { Platform, StyleSheet, Text, View, TouchableOpacity } from 'react-native';
import FoxitPDF from '@foxitsoftware/react-native-foxitpdf';

const instructions = Platform.select({
  ios: 'Press Cmd+R to reload,\n' + 'Cmd+D or shake for dev menu',
  android:
    'Double tap R on your keyboard to reload,\n' +
    'Shake or press menu button for dev menu',
});

type Props = {};
export default class App extends Component<Props> {

  constructor(props) {
    super(props);
    // The "foxit_sn" can be found in the "rdk_sn.txt" file (the string after "SN=").
    // The "foxit_key" can be found in the "rdk_key.txt" file (the string after
"Sign=").
    FoxitPDF.initialize("foxit_sn", "foxit_key");

  onPress() {
    FoxitPDF.openDocument('mnt/sdcard/FoxitSDK/complete_pdf_viewer_guide_android.pdf');
  }

  render() {
    return (
      <View style={styles.container}>
        <TouchableOpacity
          style={styles.button}
          onPress={this.onPress}
        >
          <Text> Open PDF </Text>
        </TouchableOpacity>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
```

```
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  button: {
    alignItems: 'center',
    backgroundColor: '#DDDDDD',
    padding: 10
  },
});
```

Note:

- *In version 6.3 or higher, the usage of the new interface **FoxitPDF.openDocument** is the same with the **FoxitPDF.openPDF**, but we recommend using **FoxitPDF.openDocument**, because **FoxitPDF.openPDF** may be deprecated in the future. Before calling **FoxitPDF.openDocument**, you should call **FoxitPDF.initialize** to initialize the SDK libraries at first.*
- *Here, we assume that you have pushed the "complete_pdf_viewer_guide_android.pdf" document found in the "samples/test_files" folder of Foxit PDF SDK for Android package into the "FoxitSDK" folder of the Android device or emulator that will be used to run this project. Certainly, you can change the file path with your own files.*

8.3.4 Run the project

To run the project, first setup your device or emulator to deploy the project to, navigate to the project directory and type the command below:

```
react-native run-android
```

After running the project successfully, click the "Open PDF" button. The "complete_pdf_viewer_guide_android.pdf" document will be displayed as shown in Figure 8-1.

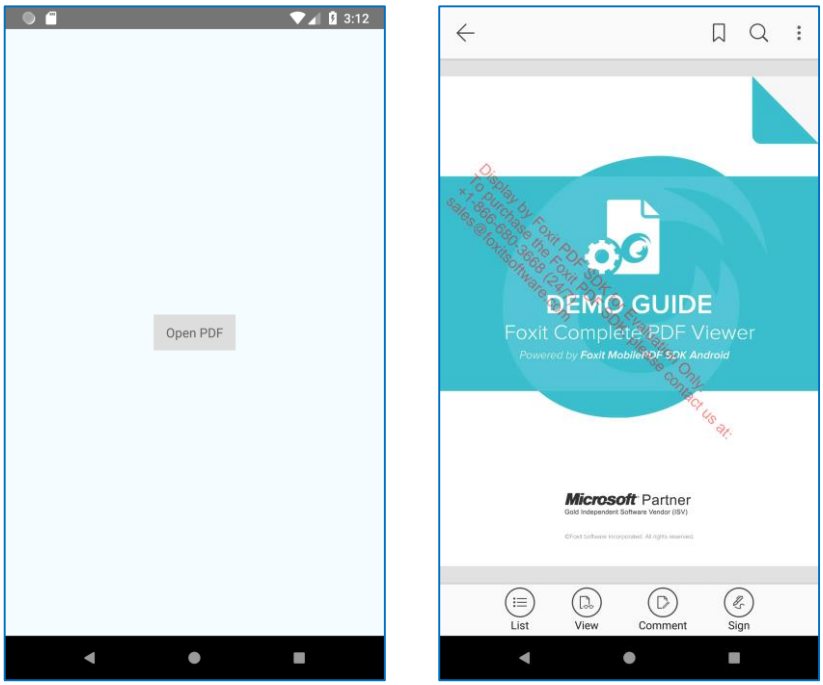


Figure 8-1

Note: *If you encounter the problem "Unable to load script from assets 'index.android.bundle'. Make sure your bundle is packaged correctly or you're running a packager server.'' when running the project, you can do the steps as below:*

1) *Create a new folder called 'assets' in the "testRN\android\app\src\main" folder, or in a command prompt or terminal, navigate to the project directory, do the command below to create a folder:*

```
cd android/app/src/main
mkdir assets
```

2) *Navigate to the project directory, type the following command:*

```
react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index.android.bundle --assets-dest android/app/src/main/res
```

3) *Type the command "react-native run-android" to rerun the project.*

8.3.5 Customize the UI

You can customize the UI including the feature modules, rights management and the properties of UI elements through a JSON file called "uiextensions_config.json" which is located in the "testRN\node_modules\@foxitsoftware\react-native-foxitpdf\lib\android\src\main\res\raw" folder.

To customize the UI, just modify the JSON file as desired, and then rerun the project. For more detailed information about the JSON file, please refer to section 4.1.2 Configuration Items Description.

9 Implement Foxit PDF SDK for Android using Xamarin

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

This section will help you get started with Foxit PDF SDK for Android and the Xamarin framework on Windows. For other operating systems, you can take this tutorial as reference.

9.1 System Requirements

- Visual Studio 2017
- Android SDK
- JDK 1.8
- Foxit PDF SDK for Android

***Note**: The version of Foxit PDF SDK for Android should match the version of the 'foxit_xamarin_android' that you choose and download, or the version that you install from the NuGet.*

9.2 Install Xamarin in Visual Studio 2017

Please refer to the official installation instructions to install Xamarin in Visual Studio 2017.

9.3 Integrate Foxit PDF SDK into your Xamarin project

There are two ways to integrate Foxit PDF SDK into your Xamarin project:

- Integrate with NuGet (supported from version 6.3)
- Integrate manually by building and referencing DLLs

9.3.1 Integrate with NuGet

It is now possible to integrate Foxit PDF SDK into your Xamarin project through NuGet packages, which is much easier than the second way "Integrate manually by building and referencing DLLs", and can save you much time.

To integrate Foxit PDF SDK into your Xamarin project through Nuget packages, please follow the steps below:

1) In the **Solution Explorer**, right-click the **References** node of your project, and click **Manage NuGet Packages…** as shown in Figure 9-1.

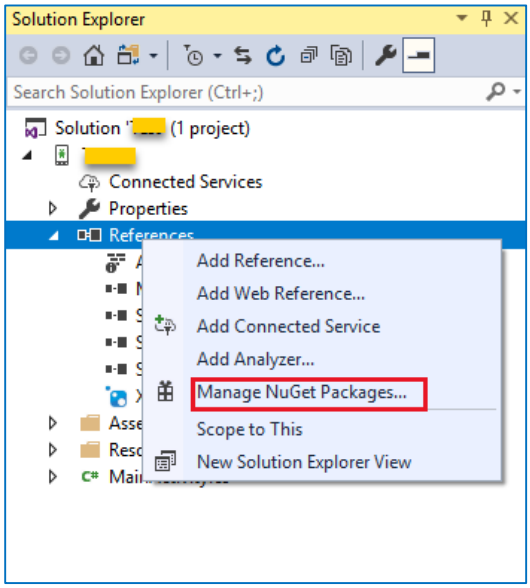


Figure 9-1

2) Then select the **Browse** tab, search for **Foxit.Android**, **Foxit.Android.UIExtensions** and **Foxit.Android.Cropper** (See Figure 9-2), and install the three packages.

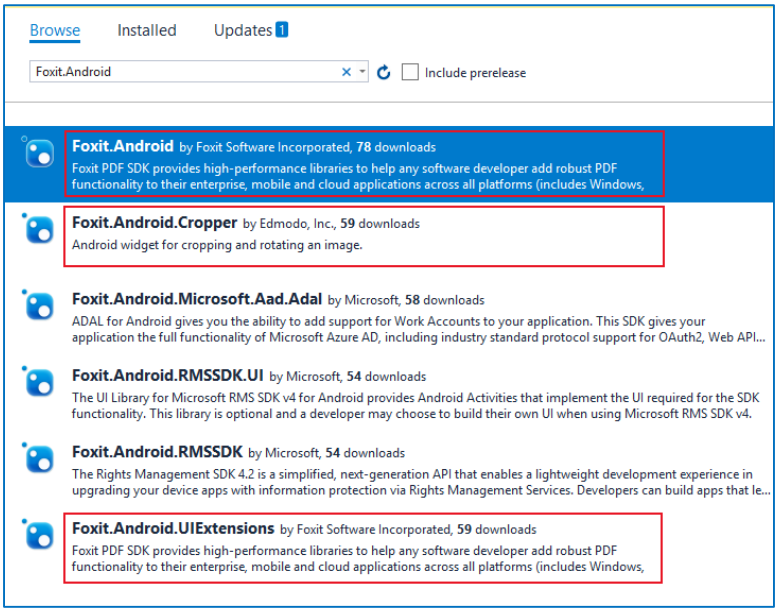


Figure 9-2

3) (**Optional**) If you want to open a RMS protected PDF document, you should install the **Foxit.Android.Microsoft.Aad.Aadl**, **Foxit.Android.RMSSDK**, and **Foxit.Android.RMSSDK.UI** packages as shown in Figure 9-2.

***Note**: The **Foxit.Android.Microsoft.Aad.Aadl**, **Foxit.Android.RMSSDK**, and **Foxit.Android.RMSSDK.UI** packages are required by the section 9.4.5 "Open a RMS protected document", so we also install these three packages. After finishing the above installation, the **References** of your project will look like Figure 9-3.*

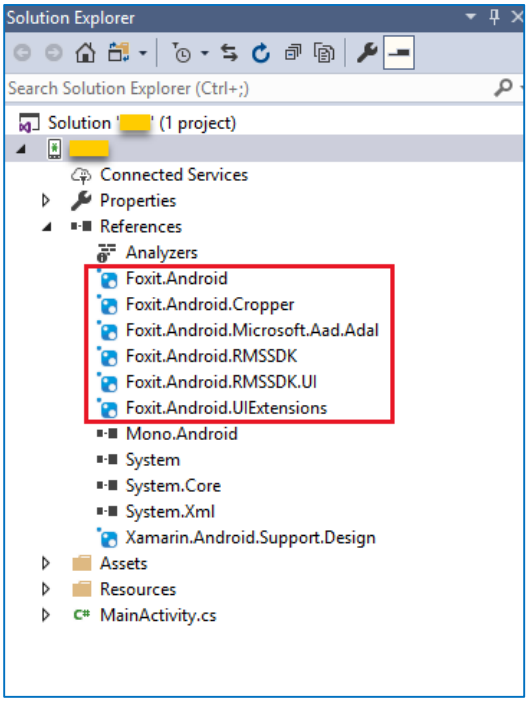


Figure 9-3

9.3.2 Integrate manually by building and referencing DLLs

To integrate manually, you should first build DLLs, and then add it as a reference to your project.

9.3.2.1 Build DLLs

Download the 'foxit_xamarin_android' from GitHub, and then build the related projects and get the DLLs, please follow the steps below:

1) Download Foxit PDF SDK for Android package, extract it, and copy the following files (libraries and licenses) in the "libs" folder of the extracted package to "foxit_xamarin_android\libs" directory:

   FoxitRDK.aar

FoxitRDKUIExtensions.aar
rdk_key.txt
rdk_sn.txt

2) Get **FoxitRDK.dll** which is the heart of the SDK including the core functionalities of Foxit PDF SDK for Android.

   Load **FoxitRDK.sln** in Visual Studio 2017 under the "foxit_xamarin_android\FoxitRDK" directory. Choose **Build** -> **Build Solution** to build the project, then the **FoxitRDK.dll** will be generated in "foxit_xamarin_android\FoxitRDK\FoxitRDK\bin\Debug (or release)" directory.

3) Get **FoxitUIExtensions.dll** which extends more powerful PDF related features including UI and resource files.

   Load **FoxitUIExtensions.sln** in Visual Studio 2017 under "foxit_xamarin_android\FoxitUIExtensions" directory. Choose **Build** -> **Build Solution** to build the project, then the **FoxitUIExtensions.dll** will be generated in "foxit_xamarin_android\FoxitUIExtensions\FoxitUIExtensions\bin\Debug (or release)" directory.

Note:

- *In the "foxit_xamarin_android" folder, the **complete_pdf_viewer** project is a sample for building a full-featured PDF Reader. Before building this project, you should build the **Cropper**, **FoxitRDK**, and **FoxitUIExtensions** projects first.*
- *If you want to open a RMS protected PDF document, you should build **Microsoft_Aad_Aadl**, **RMSSDK**, and **RMSSDK_UI** projects in the "foxit_xamarin_android" folder to get the DLLs. Then, add the DLLs as references to your project.*

9.3.2.2 Add the built DLLs as references to your project

This section takes **FoxitRDK.dll** as an example to show you how to add it as a reference to your project. For other DLLs, do the same steps with **FoxitRDK.dll**. We assume that you have got the **FoxitRDK.dll**, if not, please refer to Build DLLs section to build the **FoxitRDK** project and generate the **FoxitRDK.dll**.

1) In the **Solution Explorer**, right-click the **References** node of your project and select **Add Reference…** as shown in Figure 9-4.

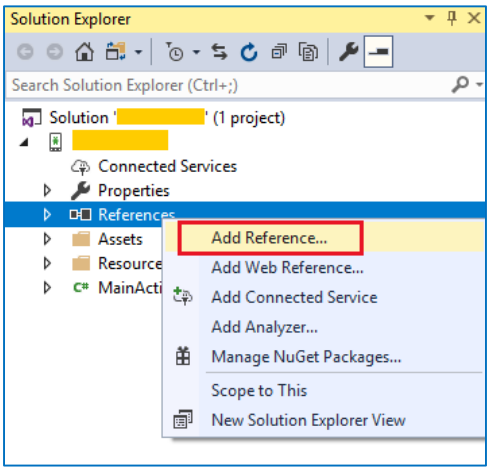


Figure 9-4

2) In the **Reference Manager** dialog, click **Browse…** (See Figure 9-5) to find the **FoxitRDK.dll** (in the "foxit_xamarin_android\FoxitRDK\FoxitRDK\bin\Debug (or release)" folder), select it and then click **Add**, which is shown in Figure 9-6.

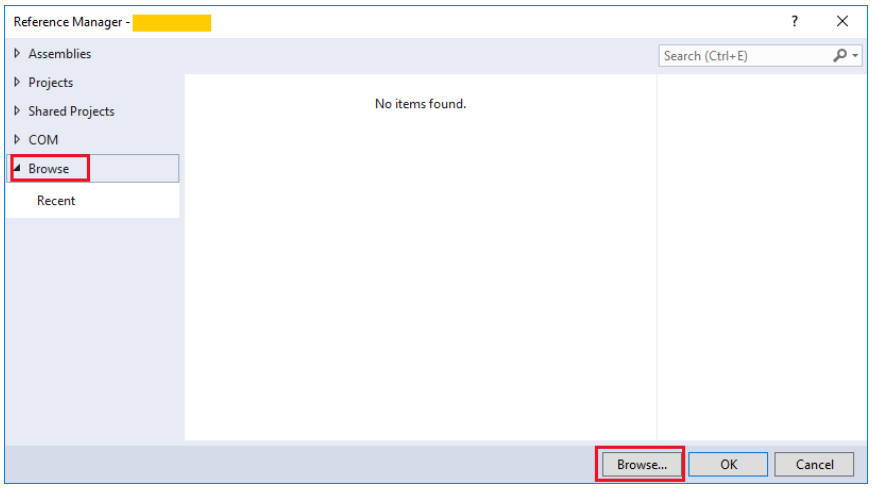


Figure 9-5

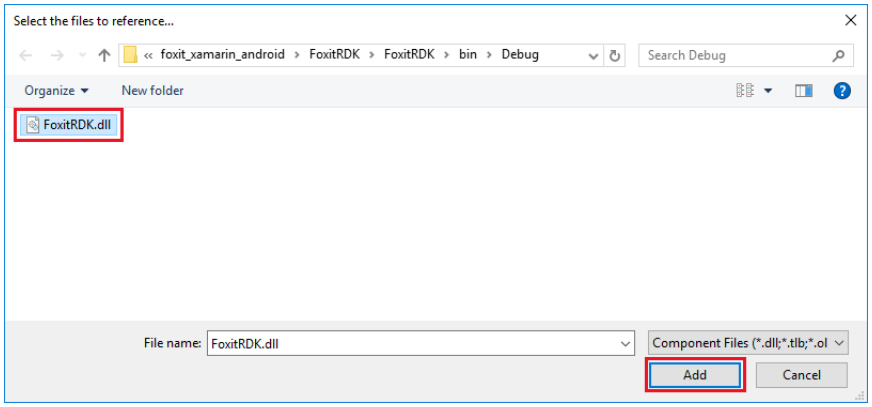


Figure 9-6

Then, the **FoxitRDK.dll** will appear in the list and has already been checked (See Figure 9-7). Click **OK**.

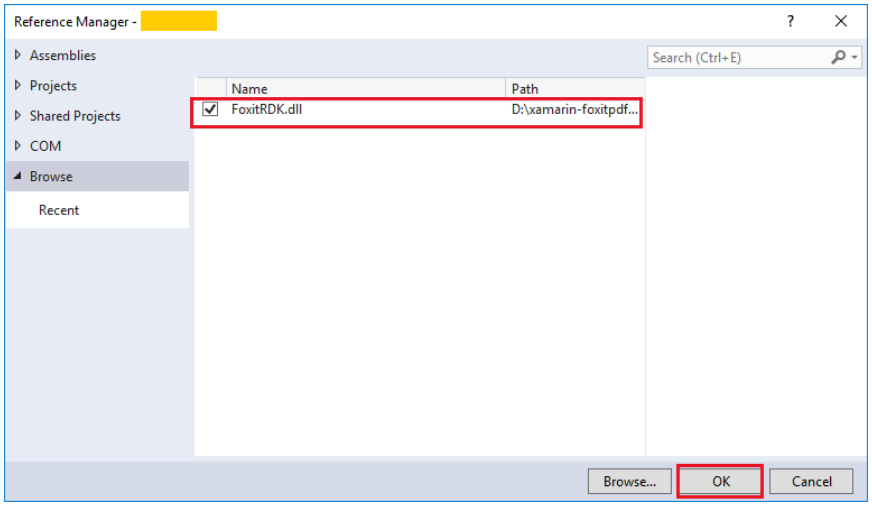


Figure 9-7

3) After finishing the above steps, the **References** of your project will look like Figure 9-8.

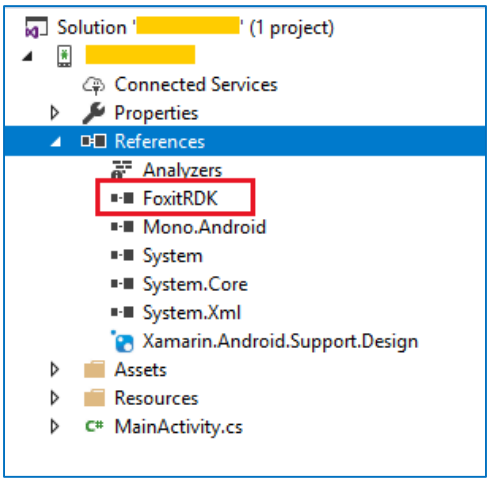


Figure 9-8

Note:

- *The section 9.4.5 "Open a RMS protected document" requires the libraries related with RMS, so please refer to Build DLLs section to build the **Microsoft_Aad_Aadl**, **RMSSDK**, and **RMSSDK_UI** projects in the "foxit_xamarin_android" folder to get the following three DLLs:*

  *Microsoft_Aad_Aadl\Microsoft_Aad_Aadl\bin\Debug**Microsoft_Aad_Aadl.dll***
  *RMSSDK\RMSSDK\bin\Debug**RMSSDK.dll***
  *RMSSDK_UI\RMSSDK_UI\bin\Debug**RMSSDK_UI.dll***

*Add the above three DLLs libraries as references to the project (refer to the steps of adding **FoxitRDK.dll**).*

- *For section 9.4.6 "Build a full-featured PDF Reader", you should refer to Build DLLs section to build the **FoxitUIExtensions** and **Cropper** projects in the "foxit_xamarin_android" folder to get the following two DLLs:*

  *FoxitUIExtensions\FoxitUIExtensions\bin\Debug**FoxitUIExtensions.dll***
  *Cropper\Cropper\bin\Debug**Cropper.dll***

  *Add the above two DLLs libraries as references to the project (refer to the steps of adding **FoxitRDK.dll**).*

*After finishing the above steps, the **References** of your project will look like Figure 9-9.*

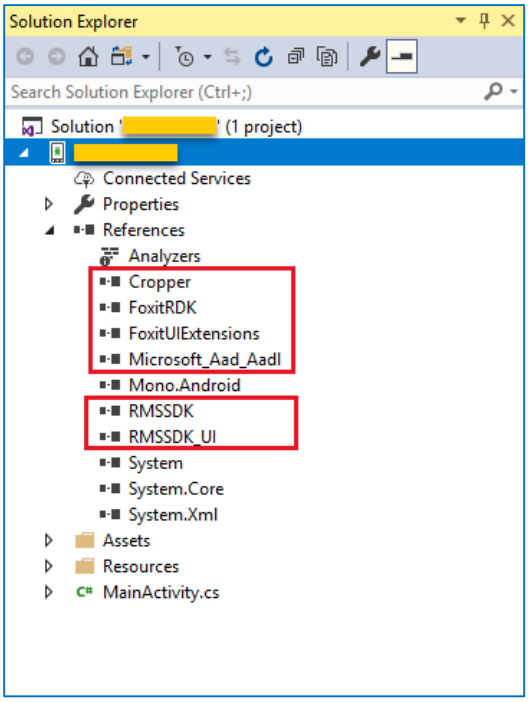


Figure 9-9

9.4 Build a Xamarin Android project using Foxit PDF SDK for Android

This section will help you to quickly build a full-featured PDF Reader in Xamarin Android platform with step-by-step instructions provided.

9.4.1 Create a new Xamarin Android project

Open Visual Studio 2017, choose **File** -> **New** -> **Project**... to start the **New Project** wizard. Create a new Xamarin Android project called "**TestXamarin**" which is shown in Figure 9-10. Then click **OK**.

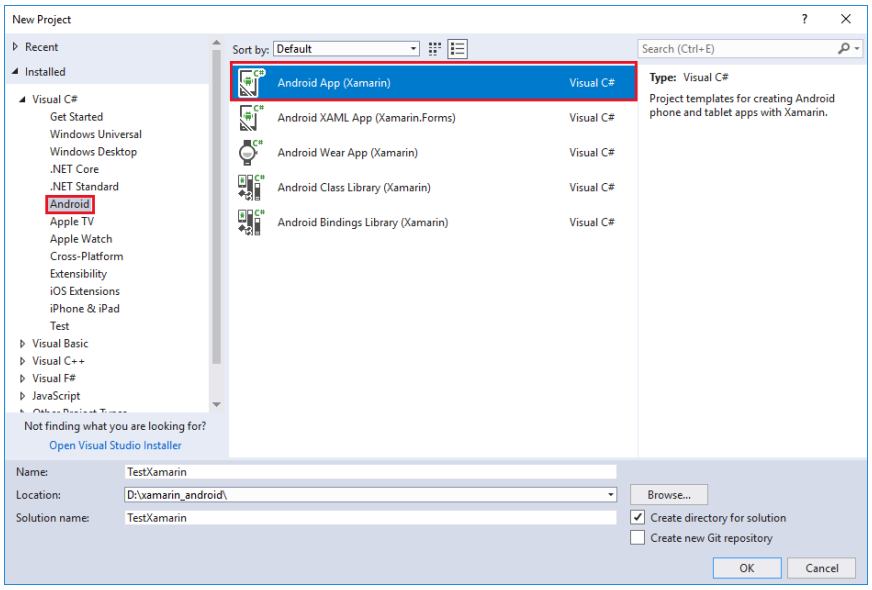


Figure 9-10

In the **New Android App** dialog, select **Single View App** as shown in Figure 9-11. Then, click **OK**.

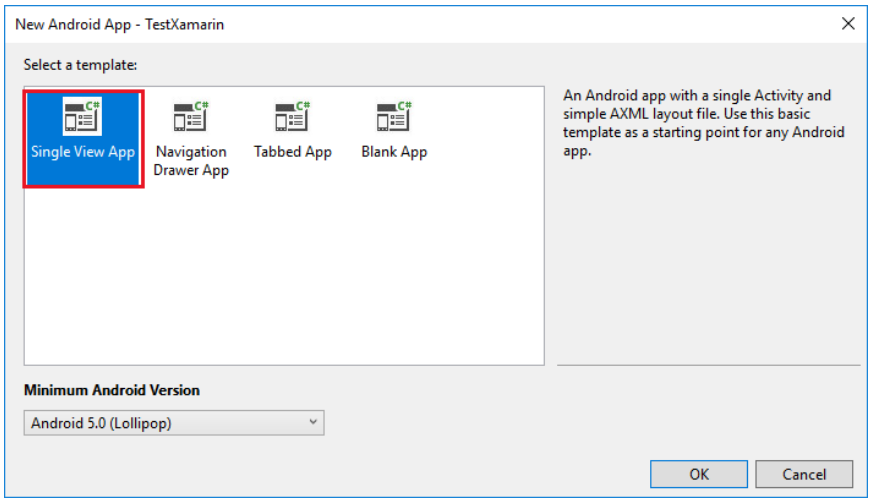


Figure 9-11

9.4.2 Integrate Foxit PDF SDK into the project

Please refer to section 9.3 "Integrate Foxit PDF SDK into your Xamarin project" to integrate the Foxit PDF SDK into the created project. We recommend using the first way "Integrate with NuGet", which is more easy and convenient.

9.4.3 Initialize Foxit PDF SDK Library

Before calling any APIs, you must first initialize Foxit PDF SDK library by using the function ***Library.Initialize****(sn, key)*. Below you can see an example of how to initialize the SDK library. The next section will show you where to include this code in "MainActivity.cs" file.

```
using Com.Foxit.Sdk;
using Com.Foxit.Sdk.Common;
...
String sn = "xxx";
String key = "xxx";
int errCode = Library.Initialize(sn, key);
if (errCode != Constants.EErrSuccess)
  return;
```

Note*: The value of "sn" can be found in the "**rdk_sn.txt**" (the string after "SN=") and the value of "key" can be found in the "**rdk_key.txt**" (the string after "Sign="). The trial license files (**rdk_sn.txt** and **rdk_key.txt**) can be found in the "foxit_xamarin_android\libs" folder or in the "libs" folder of Foxit PDF SDK for Android package.*

9.4.4 Display a PDF document using PDFViewCtrl

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

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

   In "MainActivity.cs" file, instantiate a PDFViewCtrl object, and call **PDFViewCtrl.OpenDoc** function to open and render the PDF document.

```
using Com.Foxit.Sdk;
...

private String path = "/mnt/sdcard/complete_pdf_viewer_guide_android.pdf";
private PDFViewCtrl pdfViewCtrl;
...

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

// Open and Render a PDF document.
pdfViewCtrl.OpenDoc(path, null);
SetContentView(pdfViewCtrl);
```

   Note*: Please make sure you have pushed the "complete_pdf_viewer_guide_android.pdf" document found in the "samples/test_files" folder of Foxit PDF SDK for Android package into the SD card of the Android device or emulator that will be used to run this project. Certainly, you can change the file path with your own files.*

Update MainActivity.cs as follows:

```
using System;
using Android.App;
using Android.OS;
using Android.Support.V7.App;

using Com.Foxit.Sdk;
using Com.Foxit.Sdk.Common;

namespace TestXamarin
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme.NoActionBar",
MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {

        // 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 String sn = "xxx";
        private String key = "xxx";

        private String path = "/mnt/sdcard/complete_pdf_viewer_guide_android.pdf";

        private PDFViewCtrl pdfViewCtrl;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            // Initialize Foxit SDK Library.
            int errCode = Library.Initialize(sn, key);
            if (errCode != Constants.EErrSuccess)
                return;

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

            // Open and Render a PDF document.
            pdfViewCtrl.OpenDoc(path, null);
            SetContentView(pdfViewCtrl);
        }
    }
}
```

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

Note: *If you want to run this project on an Android 6.0 (API 23) or higher devices/emulators, you can do one of the following:*

a) Change the `android:targetSdkVersion` *in "AndroidManifest.xml" to a lower SDK version that is less than 23, such as 21.*

b) Write additional code to require the authorization of runtime permissions.

a) Change the android:targetSdkVersion in "AndroidManifest.xml"

In this case, set the "users-permission" in the "AndroidManifest.xml" found in the "TestXamarin\Properties" to give the project permission to write and read the SD card of the Android devices or emulators (See Figure 9-12).

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

Figure 9-12

b) Write additional code to require the authorization of runtime permissions.

In the **MainActivity.cs** file, add code to require the authorization of runtime permissions. So, **update the whole MainActivity.cs as follows:**

```
using System;
using Android.App;
using Android.OS;
using Android.Runtime;
using Android.Support.V7.App;

using Com.Foxit.Sdk;
using Com.Foxit.Sdk.Common;
using Android;
using Android.Content.PM;
using Android.Support.V4.Content;
using Android.Support.V4.App;

namespace TestXamarin
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme.NoActionBar",
MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        public static int REQUEST_EXTERNAL_STORAGE = 1;
        private static string[] PERMISSIONS_STORAGE = {
            Manifest.Permission.ReadExternalStorage,
            Manifest.Permission.WriteExternalStorage
        };

        // 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 String sn = "xxx";
        private String key = "xxx";
```

```
        private String path = "/mnt/sdcard/complete_pdf_viewer_guide_android.pdf";

        private PDFViewCtrl pdfViewCtrl;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            // Initialize Foxit SDK Library.
            int errCode = Library.Initialize(sn, key);
            if (errCode != Constants.EErrSuccess)
                return;

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

            // Require the authorization of runtime permissions.
            if (Build.VERSION.SdkInt > BuildVersionCodes.M)
            {
                Permission permission =
ContextCompat.CheckSelfPermission(this.ApplicationContext,
Manifest.Permission.WriteExternalStorage);
                if (permission != Permission.Granted)
                {
                    ActivityCompat.RequestPermissions(this, PERMISSIONS_STORAGE,
REQUEST_EXTERNAL_STORAGE);
                    return;
                }
            }

            // Open and Render a PDF document.
            pdfViewCtrl.OpenDoc(path, null);
            SetContentView(pdfViewCtrl);
        }

        public override void OnRequestPermissionsResult(int requestCode, string[]
permissions, [GeneratedEnum] Permission[] grantResults)
        {
            if (requestCode == REQUEST_EXTERNAL_STORAGE
             && grantResults[0] == Permission.Granted)
            {
                if (pdfViewCtrl != null)
                {
                    pdfViewCtrl.OpenDoc(path, null);
                }
            }
            else
            {
                base.OnRequestPermissionsResult(requestCode, permissions,
grantResults);
            }
        }
    }
}
```

In this chapter, we build and run the project on an AVD targeting 9.0 (API 28), and use the second method (require authorization of runtime permissions) to get the permissions to write and read the SD card of the emulator.

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see that the "complete_pdf_viewer_guide_android.pdf" document is displayed as shown in Figure 9-13. Now, this sample app has some basic PDF features, such as zooming in/out and page turning.

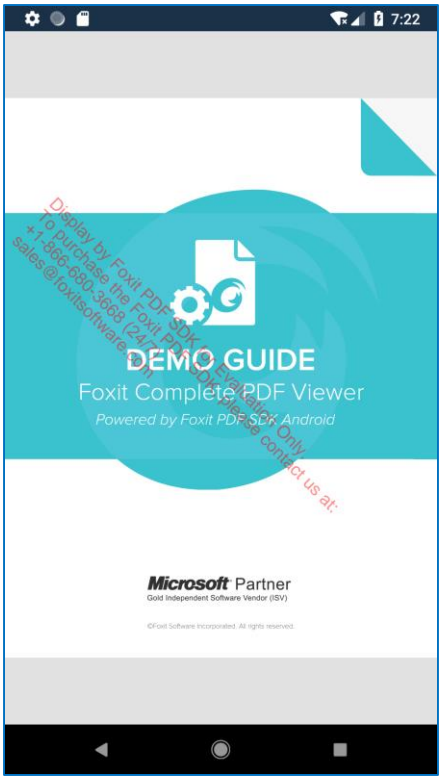


Figure 9-13

9.4.5 Open a RMS protected document

To open a RMS protected document, please make sure you have added the references of the libraries related with RMS. If not, please refer to the section 9.3 "Integrate Foxit PDF SDK into your Xamarin project". Then, follow the steps below:

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

```
pdfViewCtrl.AttachedActivity = this;
```

2) Handle the activity result from the UI operations. Call the API "`pdfViewCtrl.HandleActivityResult()`" on the related **`OnActivityResult`** function.

```
protected override void OnActivityResult(int requestCode, [GeneratedEnum] Result
resultCode, Intent data)
{
    pdfViewCtrl.HandleActivityResult(requestCode, (int)resultCode, data);
```

```
}
```

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

```
using System;
using Android.App;
using Android.OS;
using Android.Runtime;
using Android.Support.V7.App;

using Com.Foxit.Sdk;
using Com.Foxit.Sdk.Common;
using Android;
using Android.Content.PM;
using Android.Support.V4.Content;
using Android.Support.V4.App;
using Android.Content;

namespace TestXamarin
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme.NoActionBar",
MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        public static int REQUEST_EXTERNAL_STORAGE = 1;
        private static string[] PERMISSIONS_STORAGE = {
            Manifest.Permission.ReadExternalStorage,
            Manifest.Permission.WriteExternalStorage
        };

        // 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 String sn = "xxx";
        private String key = "xxx";

        private String path = "/mnt/sdcard/Sample_RMS.pdf";

        private PDFViewCtrl pdfViewCtrl;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            // Initialize Foxit SDK Library.
            int errCode = Library.Initialize(sn, key);
            if (errCode != Constants.EErrSuccess)
                return;

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

            // Set the associated activity for RMS UI operations.
            pdfViewCtrl.AttachedActivity = this;

            // Require the authorization of runtime permissions.
            if (Build.VERSION.SdkInt > BuildVersionCodes.M)
            {
```

```
                Permission permission =
ContextCompat.CheckSelfPermission(this.ApplicationContext,
Manifest.Permission.WriteExternalStorage);
                if (permission != Permission.Granted)
                {
                    ActivityCompat.RequestPermissions(this, PERMISSIONS_STORAGE,
REQUEST_EXTERNAL_STORAGE);
                    return;
                }
            }

            // Open and Render a PDF document.
            pdfViewCtrl.OpenDoc(path, null);
            SetContentView(pdfViewCtrl);
        }

        public override void OnRequestPermissionsResult(int requestCode, string[]
permissions, [GeneratedEnum] Permission[] grantResults)
        {
            if (requestCode == REQUEST_EXTERNAL_STORAGE
             && grantResults[0] == Permission.Granted)
            {
                if (pdfViewCtrl != null)
                {
                    pdfViewCtrl.OpenDoc(path, null);
                }
            }
            else
            {
                base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
            }
        }

        // Used for opening a RMS document.
        protected override void OnActivityResult(int requestCode, [GeneratedEnum] Result
resultCode, Intent data)
        {
            pdfViewCtrl.HandleActivityResult(requestCode, (int)resultCode, data);
        }
    }
}
```

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

Run the project

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window to allow the project to access files on the device, and then you will see the following window (Figure 9-14) which prompts you to input your organizational email, and later input the password, then you can open the RMS protected document.

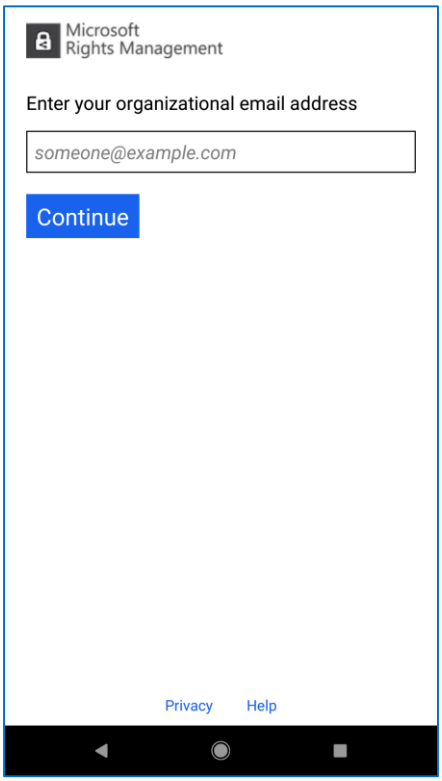


Figure 9-14

9.4.6 Build a full-featured PDF Reader

If you want to build a full-featured PDF reader (such like including commenting, editing features, and so on), please make sure you have added the references of the **UIExtensions** and **Cropper** libraries. If not, please refer to the section 9.3 "Integrate Foxit PDF SDK into your Xamarin project". Then, follow the steps below:

1) Set the required permissions in the "AndroidManifest.xml" found in the "TestXamarin\Properties" (See Figure 9-15).

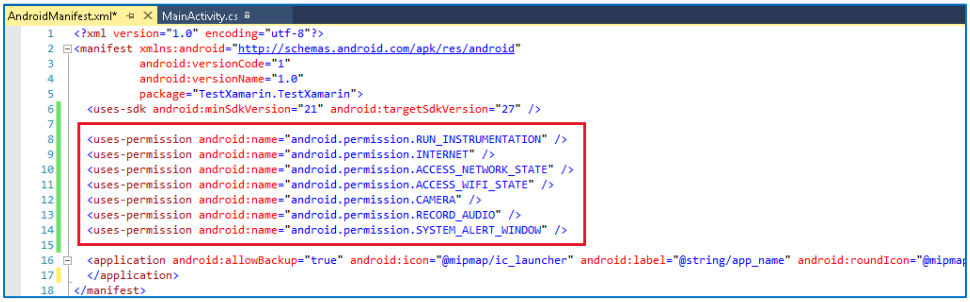


Figure 9-15

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

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

2) Add the core code below to "MainActivity.cs" file.

a) Instantiate a UIExtensionsManager object and set it to PDFViewCtrl.

```
using Com.Foxit.Uiextensions;
...

private UIExtensionsManager uiExtensionsManager;
...

uiExtensionsManager = new UIExtensionsManager(this.ApplicationContext,
pdfViewCtrl);
uiExtensionsManager.AttachedActivity = this;
uiExtensionsManager.OnCreate(this, pdfViewCtrl, savedInstanceState);
pdfViewCtrl.UIExtensionsManager = uiExtensionsManager;
```

b) Open and render a PDF document, and set the content view. Call `uiExtensionsManager.OpenDocument()` function to open and render a PDF document instead of calling `pdfViewCtrl.OpenDoc()` function.

```
uiExtensionsManager.OpenDocument(path, null);
SetContentView(uiExtensionsManager.ContentView);
```

c) Add `ConfigurationChanges = ConfigChanges.KeyboardHidden|ConfigChanges.Orientation|ConfigChanges.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.

```
[Activity(Label = "@string/app_name", ConfigurationChanges =
ConfigChanges.KeyboardHidden|ConfigChanges.Orientation|ConfigChanges.ScreenSize,
Theme = "@style/AppTheme.NoActionBar", MainLauncher = true)]
```

Update the whole content of MainActivity.cs file as follows:

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

```
using System;
using Android.App;
using Android.OS;
using Android.Runtime;
using Android.Support.V7.App;

using Com.Foxit.Sdk;
using Com.Foxit.Sdk.Common;
using Android;
```

```
using Android.Content.PM;
using Android.Support.V4.Content;
using Android.Support.V4.App;
using Android.Content;
using Com.Foxit.Uiextensions;
using Android.Views;
using Android.Content.Res;

namespace TestXamarin

{

    [Activity(Label = "@string/app_name", ConfigurationChanges =
ConfigChanges.KeyboardHidden|ConfigChanges.Orientation|ConfigChanges.ScreenSize, Theme
= "@style/AppTheme.NoActionBar", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        public static int REQUEST_EXTERNAL_STORAGE = 1;
        private static string[] PERMISSIONS_STORAGE = {
            Manifest.Permission.ReadExternalStorage,
            Manifest.Permission.WriteExternalStorage
        };

        // 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 String sn = "xxx";
        private String key = "xxx";

        private String path = "/mnt/sdcard/complete_pdf_viewer_guide_android.pdf";

        private PDFViewCtrl pdfViewCtrl;
        private UIExtensionsManager uiExtensionsManager;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            // Initialize Foxit SDK Library.
            int errCode = Library.Initialize(sn, key);
            if (errCode != Constants.EErrSuccess)
                return;

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

            // Set the associated activity for RMS UI operations.
            pdfViewCtrl.AttachedActivity = this;

            // Initialize a UIExtensionManager object and set it to PDFViewCtrl.
            uiExtensionsManager = new UIExtensionsManager(this.ApplicationContext,
pdfViewCtrl);
            uiExtensionsManager.AttachedActivity = this;
            uiExtensionsManager.OnCreate(this, pdfViewCtrl, savedInstanceState);
            pdfViewCtrl.UIExtensionsManager = uiExtensionsManager;

            // Require the authorization of runtime permissions.
            if (Build.VERSION.SdkInt > BuildVersionCodes.M)
            {
```

```
                Permission permission =
ContextCompat.CheckSelfPermission(this.ApplicationContext,
Manifest.Permission.WriteExternalStorage);
                if (permission != Permission.Granted)
                {
                    ActivityCompat.RequestPermissions(this, PERMISSIONS_STORAGE,
REQUEST_EXTERNAL_STORAGE);
                    return;
                }
            }

            // Open and Render a PDF document.
            uiExtensionsManager.OpenDocument(path, null);
            SetContentView(uiExtensionsManager.ContentView);
        }

        public override void OnRequestPermissionsResult(int requestCode, string[]
permissions, [GeneratedEnum] Permission[] grantResults)
        {
            if (requestCode == REQUEST_EXTERNAL_STORAGE
             && grantResults[0] == Permission.Granted)
            {
                if (uiExtensionsManager != null)
                {
                    uiExtensionsManager.OpenDocument(path, null);
                    SetContentView(uiExtensionsManager.ContentView);
                }
            }
            else
            {
                base.OnRequestPermissionsResult(requestCode, permissions,
grantResults);
            }
        }

        // Used for opening a RMS document.
        protected override void OnActivityResult(int requestCode, [GeneratedEnum]
Result resultCode, Intent data)
        {
            pdfViewCtrl.HandleActivityResult(requestCode, (int)resultCode, data);
        }

        protected override void OnStart()
        {
            if (uiExtensionsManager != null)
            {
                uiExtensionsManager.OnStart(this);
            }
            base.OnStart();
        }

        protected override void OnStop()
        {
            if (uiExtensionsManager != null)
            {
                uiExtensionsManager.OnStop(this);
            }
            base.OnStop();
        }
```

```
        protected override void OnPause()
        {
            if (uiExtensionsManager != null)
            {
                uiExtensionsManager.OnPause(this);
            }
            base.OnPause();
        }

        protected override void OnResume()
        {
            if (uiExtensionsManager != null)
            {
                uiExtensionsManager.OnResume(this);
            }
            base.OnResume();
        }

        protected override void OnDestroy()
        {
            if (uiExtensionsManager != null)
            {
                uiExtensionsManager.OnDestroy(this);
            }
            base.OnDestroy();
        }

        public override void OnConfigurationChanged(Configuration newConfig)
        {
            base.OnConfigurationChanged(newConfig);
            if (uiExtensionsManager != null)
            {
                uiExtensionsManager.OnConfigurationChanged(this, newConfig);
            }
        }

        public override bool OnKeyDown([GeneratedEnum] Keycode keyCode, KeyEvent e)
        {
            if (uiExtensionsManager != null && uiExtensionsManager.OnKeyDown(this,
(int)keyCode, e)) return true;
            return base.OnKeyDown(keyCode, e);
        }
    }
}
```

3) Enable Multi-Dex. If you don't, you may encounter the problem "error MSB6006: "java.exe" exited with code 2."

   Right-click the *TestXamarin* project, choose Properties, check **Enable Multi-Dex** under Android Options -> Packaging properties (See Figure 9-16).

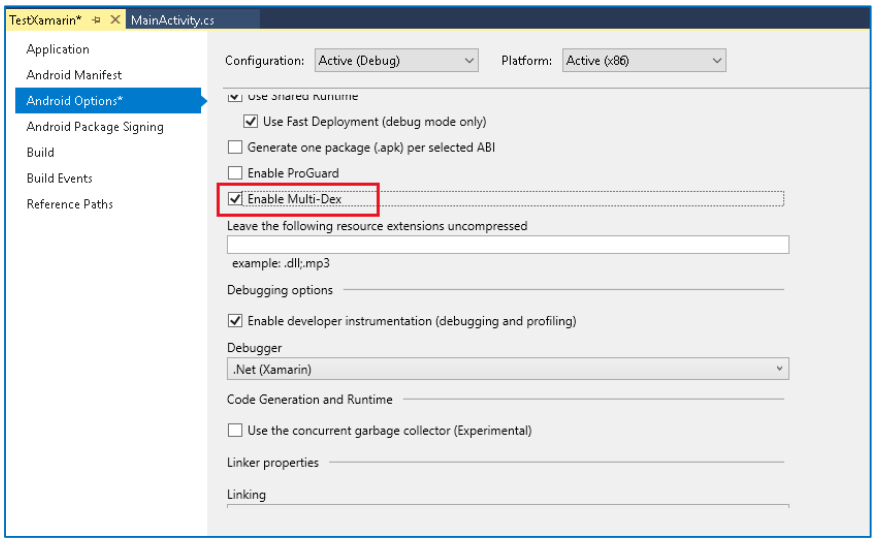


Figure 9-16

Run the project

After building the project and installing APK on the emulator, tap **Allow** on the pop-up window, and then you will see that the "complete_pdf_viewer_guide_android.pdf" document is displayed as shown in Figure 9-17. Now, it is a full-featured PDF Reader which includes all the features in Foxit PDF SDK for Android and it also supports to open a RMS protected document.

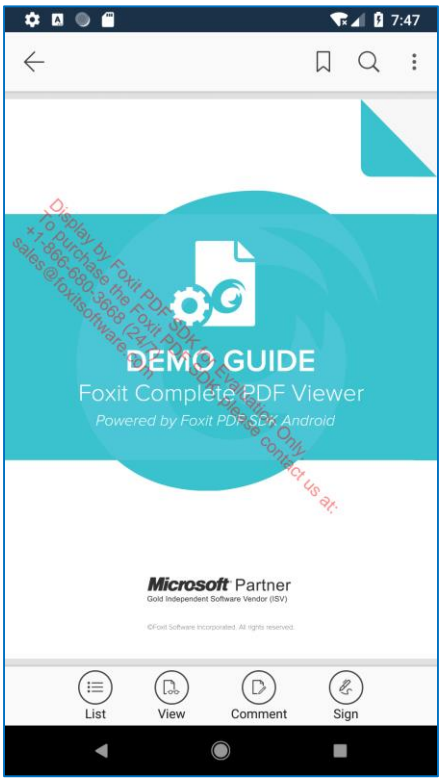


Figure 9-17

9.4.7 Customize the UI

For how to customize the UI in your Xamarin Android project, you can refer to the section 4 "Customizing User Interface".

9.5 FAQ for Xamarin Android

How to fix the "Java.Lang.IllegalStateException: This app has been built with an incorrect configuration. Please configure your build for VectorDrawableCompat" exception?

This exception happens when the project can't find the Android.Xamarin.Vector.Drawable NuGet package. Normally this would be set up in "build.gradle" in Android Studio, but when using Xamarin you need to install the package in Visual Studio. Right-click your project and choose 'Manage NuGet Packages…'. In the NuGet Package Manager window, search for 'Xamarin.Android.Support.Vector.Drawable' and then install it, which is shown in Figure 9-18.

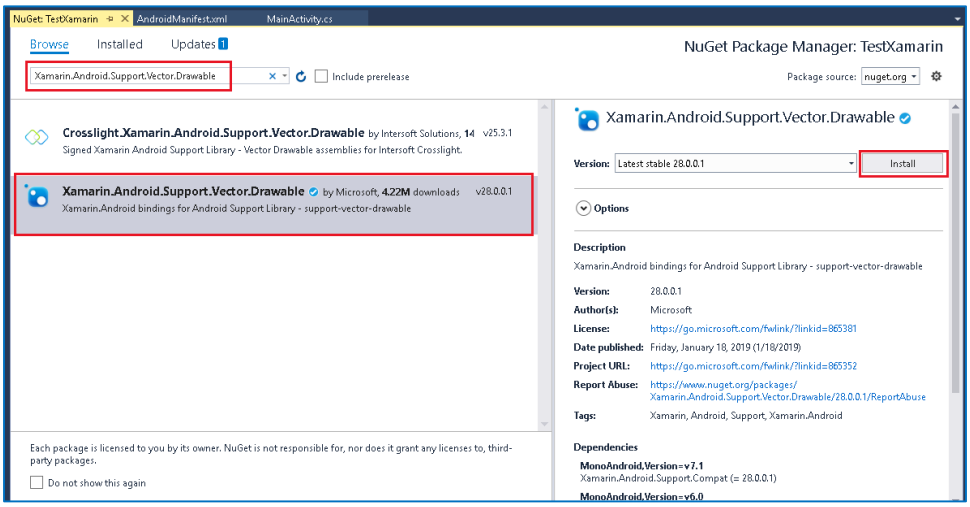


Figure 9-18

***Note**: If your project target API is below 28, you need to download a version of the NuGet package that is compatible with your project (i.e API 27 project should use version 27.0.1.2 of 'Xamarin.Android.Support.Vector.Drawable').*

10 FAQ

10.1 Open a PDF document from a specified PDF file path

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

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

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

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

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

// Open an unencrypted PDF document from a specified PDF file path.
String path = "/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.
	pdfViewCtrl.setDoc(document);
} catch (Exception e) {
```

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

10.2 Display a specified page when opening a PDF document

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

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

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

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

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

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

...

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

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

        @Override
        public void onDocWillClose(PDFDoc pdfDoc) { }

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

        @Override
        public void onDocWillSave(PDFDoc pdfDoc) { }

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

```
    };
```

10.3 License key and serial number cannot work

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

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

10.4 Add a link annotation to a PDF file

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

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

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

String path = "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();
}
```

10.5 Insert an image into a PDF file

How do I insert an image into a PDF file?

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

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

```
String path = "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:

```
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 = new Stamp(page.addAnnot(Annot.e_Stamp, new RectF(100, 350, 250, 150)));

    // Load a local image and convert it to a Bitmap.
    Bitmap bitmap = BitmapFactory.decodeFile("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();
}
```

10.6 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?

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

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

Through JSON file

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

Through calling API

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

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

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

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

10.7 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?

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

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

Through JSON file

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

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

Through calling API

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

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

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

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

10.8 Indexed Full Text Search support

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

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

To use this feature, follows the steps below:

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

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

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

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

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

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

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

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

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

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

Following is a sample for how to use it:

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

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

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

```
        state = progressive.resume();
    }

    // Create a callback object which will be invoked when a matched one is found.
    MySearchCallback searchCallback = new MySearchCallback();

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

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

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

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

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

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

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

    @Override
    public void release() {
    }

    @Override
    public int retrieveSearchResult(String filePath, int pageIndex, String matchResult,
int matchStartTextIndex, int matchEndTextIndex) {
        String s = String.format("Found file is :%s \n Page index is :%d Start text
index :%d End text index :%d \n Match is :%s \n\n", filePath, pageIndex,
matchStartTextIndex, matchEndTextIndex, matchResult);
        Log.v(TAG, "retrieveSearchResult: " + s);
        return 0;
    }
}
```

Note:

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

10.9 Print PDF document

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

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

// for iPhone and iTouch

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

Following is a sample for how to use it:

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

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

    @Override
    public void printFailed() {
    }

    @Override
    public void printCancelled() {
    }
};

try {
```

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

10.10 Night mode color settings

How can I set the night mode color?

From version 6.3, you can set the night mode color easily in the configuration JSON file, just set the following two items:

```
"mapForegroundColor": "#000333",
"mapBackgroundColor": "#fff666",
```

From version 5.1, if you wan to set the night mode color, please first call the **PDFViewCtrl.setMappingModeBackgroundColor(int)** and **PDFViewCtrl.setMappingModeForegroundColor(int)** APIs to set the color as you wish, and then set the color mode by using **PDFViewCtrl.setColorMode(int)**.

Note: *If the color mode is already set to Renderer.e_ColorModeMapping, 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:

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

11 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 PDF SDK for Android, 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 PDF 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