DEVELOPER GUIDE

FOR FOXIT PDF SDK 4.1

2014-8-20

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

TABLE OF CONTENTS

1 INTRODUCTION TO FOXIT SDK

Have you ever thought about building your own application that can do everything you want with PDF files? If your answer is "Yes", congratulations! You just found the best solution in the industry that allows you to build stable, secure, efficient and full-featured PDF applications.

1.1 Why is Foxit your choice

Foxit is an Amazon-invested leading software provider of solutions for reading, editing, creating, organizing, and securing PDF documents. Customers choose Foxit products for the following reasons:

- **High performance** – Very fast on PDF parsing, rendering and conversion.
- **Lightweight footprint** – Do not exhaust system resource and deploys quickly.
- **Cross-platform support** – Support Microsoft Windows, Mac OS, Solaris, Linux, Android, iOS etc.
- **Compatibility** – ISO 32000-1/PDF 1.7 standards compliant and compatible with other PDF products.
- **Great value/affordability** – Right features at right price with email and phone support.
- **Security** - Safeguards confidential information.

In addition, Foxit products are fully supported by our dedicated support engineers if support and maintenance are purchased. Updates are released on a regular basis. Developers may focus more on their solution building rather than spending time on PDF specification. Foxit will be the right choice if you need solutions with excellent features and low cost!

1.2 Foxit PDF SDK

Foxit PDF SDK allows developers to incorporate powerful PDF technology to view, search, and annotate PDF documents and forms. Foxit PDF SDK is easy to integrate and platform independent, it reduces time for release by developing features and porting them to multiple platforms.

Foxit PDF SDK is a powerful multi-platform software. It is compatible with Foxit embedded PDF SDK1.0, Foxit embedded SDK2.0 and Foxit PDF SDK DLL3.1. Foxit PDF SDK enables users to develop their applications with C/C++, object C or Java on multi-platforms such as Window, Linux, Mac, iOS and Android.

1.3 What's new compared with embedded SDK and DLL SDK

1.3.1 Support robust PDF applications on mobile platforms

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

The key of OOM mechanism is that Foxit PDF SDK will monitor the usage of memory and inform applications to do recovery or take recovery operations automatically once OOM is detected. OOM is an evolved feature in Foxit PDF SDK because of its complexity. Currently, the solution to OOM in PDF SDK is that Foxit PDF SDK will initiate a self-recovery of the pdf document to its original states if OOM happens when applications running with limited memory. In this way, part of the data from document editing may be lost and needs to be edited by applications again. About the details how OOM works, please refer to chapter 4.22 "How to utilize OOM provided by PDF SDK to implement robust applications on mobile platforms".

1.3.2 Thread safety

Since most applications use multi-thread programming, it requires PDF SDK to provide thread safe APIs to support this**.** From SDK 4.0, Foxit PDF SDK introduces thread safety mechanism to ensure that all APIs are thread safety. It frees developers from multi-thread supporting details. Based on Foxit PDF SDK, developers can concentrate on multi-thread applications and don't need to worry about thread safety in PDF API level.

1.4 Features

Foxit PDF SDK has a standard package and 7 optional packages, each of which contains several features as listed in Table 1-1. Users can choose the packages and features based on their needs.

Table 1-1

Package name	Features included
Standard	PDF rendering, document navigation, get page information, text extraction and search, access to PDF objects, asynchronous PDF and text reflow, access to layer's information
Edit	Edit document, pages and PDF objects.
Image Conversion	Convert between PDF files and images (bmp, tif, jpx, jpg, gif)
Form	Access form information, import a FDF file into a form and export data to a FDF file.
Annotation	Create, edit and remove annotation. Create watermark.
Security	Support password, certificate, DRM and custom encryption
Sensitive Ink	Generate PSI and convert PSI to annotation.
Barcode	Generate a barcode bitmap from a given string and barcode type.

2 INTRODUCTION TO PDF

2.1 History of PDF

PDF is a file format used to represent documents in a manner independent of application software, hardware, and operating systems. Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, font, graphics, and other information needed to display it.

While Adobe Systems made the PDF specification available for free in 1993, PDF remained a proprietary format controlled by Adobe, until July 1, 2008, when it was officially released as an open standard and published by the International Organization for Standardization as ISO 32000-1:2008. In 2008, Adobe published a Public Patent License to ISO 32000-1 granting royalty-free rights for all patents owned by Adobe that are necessary to make, use, sell and distribute PDF compliant implementations.

2.2 PDF Document Structure

A PDF document is composed of one or more pages. Each page has its own specifications to indicate its appearance. All the contents in a PDF page, such as text, image, annotation and form, etc. are represented as PDF objects. A PDF document can be considered as a hierarchy of objects contained in the body section of a PDF file (more detailed description about PDF objects is in section 4.20). Displaying a PDF document in an application involves loading PDF document, parsing PDF objects, retrieving/decoding pages content and displaying/printing it on a device. Editing a PDF document requires parsing the document structure, making changes and reorganizing the PDF objects in a document. These operations could be done by a conforming PDF reader/editor or in your own applications through APIs provided by Foxit.

2.3 PDF Document Features

PDF supports a lot of features to enhance the document capability, such as document encryption, digital signatures, java script actions, form filling, layered content, multimedia support and etc. These features provide users with more flexibility in displaying, exchanging and editing documents. Foxit supports all PDF features in the ISO standard. Users can use Foxit PDF SDK to fulfill these advanced features in your applications.

3 GETTING STARTED

It's very easy to setup Foxit PDF SDK and see it in action! It takes just a few minutes and we'll show you how. Foxit PDF SDK is a cross platform SDK product. It supports the same interfaces for desktop system of Windows, Linux, Mac, and mobile system of iOS, Android. The following sections introduce the structure of installation package, how to apply a license, how to run a demo and how to create your own project for different platforms.

3.1 System Requirements

Windows:

Windows XP, Vista, 7 and 8 (32-bit, 64-bit)

Windows Server 2003, 2008 and 2012 (32-bit and 64-bit)

The release package includes a 32 bit version and native 64 bit version DLL library for windows 32/64.

Note: it only supports for Windows 8 classic style not for Store App.

Linux:

Linux 32-bit and Linux 64-bit OS

All Linux test cases have been tested on Centos 6.3 32/64 bit

The release package includes both 32-bit and 64-bit version Linux libraries (.so files).

Mac OS X:

Mac OS X 10.6 to 10.9

iOS:

iOS 5 and later

All iOS test cases have been tested on Xcode 4.6 or later

The release package includes 2 types of "*.a" SDK libraries

1. arm v7/v7s Library for applications running on iPhone, iPod and iPad
2. simulator i386 Library for applications running on i386 simulator

Android:

Android 2.2 (API-Level-8) and later

The release package for Android C APIs includes 2 types of "*.a" SDK libraries

1. x86 library for x86 devices
2. armv5/v7 library for arm devices

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

1. x86 library for x86 devices
2. armv5/v7 library for arm devices

3.2 Windows

3.2.1 What's in the package

Download Foxit PDF SDK zip file and unzip it to folder "foxitpdfsdk_4_1_win", which is shown in Figure 3-1. One thing to note that the highlighted rectangle in the figure is the version of the SDK. Here the SDK version is 4.1, so it shows 4_1. Other highlighted rectangles have the same meaning in this guide.

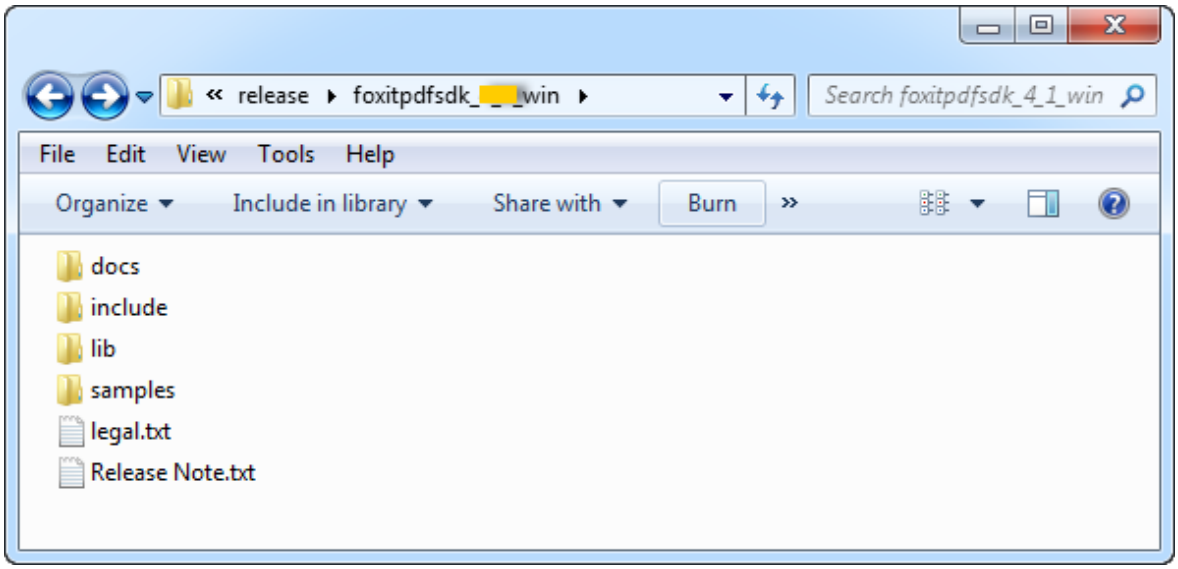

Figure 3-1

docs: API references, demo tutorials, developer guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos

In “samples”, there are two types of demos. “samples/simple_sample” contains more than 20 demos that cover a wide range of PDF applications. “samples/view_demo” contains a demo that realizes a simple PDF viewer.

For the first type of demos under “samples/simple_sample” directory, input files for all demos are put in “input_files”, output files for all demos are put in “output_files” and binary files after building are generated in “samples/simple_sample/bin” folder. A snapshot of “samples/simple_sample” folder is shown in Figure 3-2.

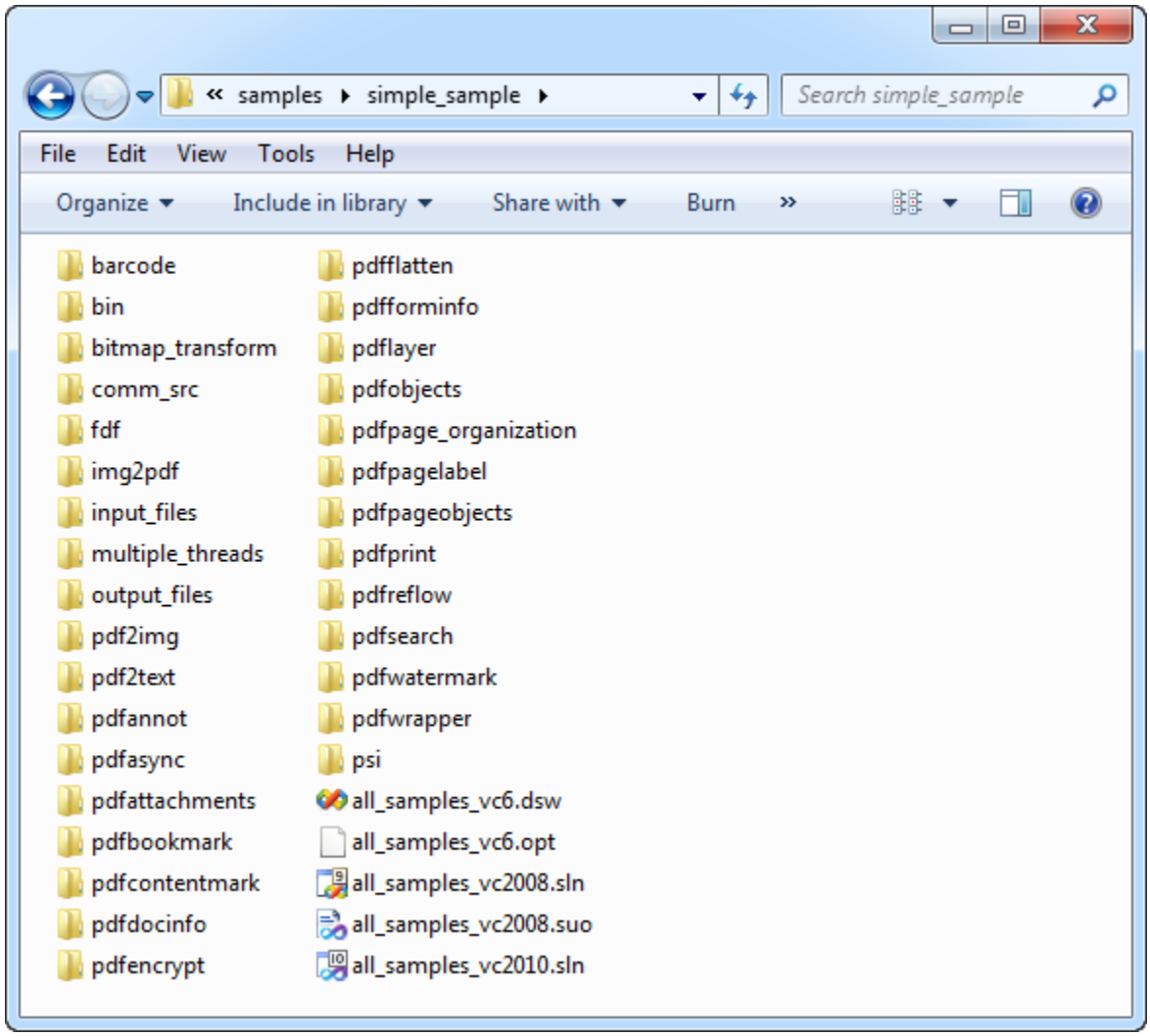

Figure 3-2

For the PDF viewer demo, all resources and files are put under “samples/view_demo/PDFReader_Demo directory as shown in Figure 3-3.

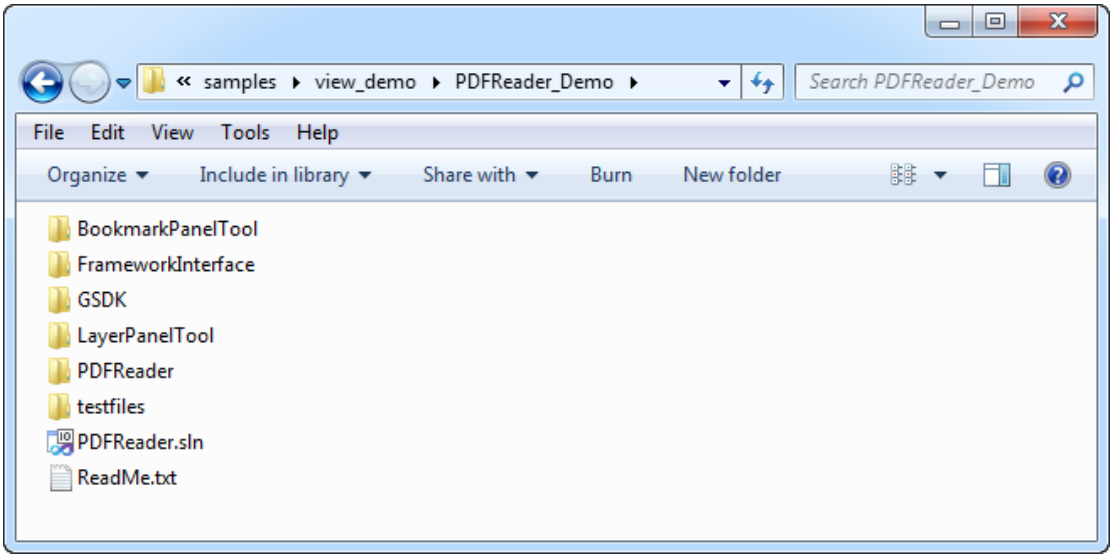

Figure 3-3

3.2.2 How to run a demo

Simple Demo

Simple demo projects provide examples for developers on how to effectively use PDF SDK APIs to complete their applications. To run a demo in Visual Studio, load the visual studio solution files "all_samples_vc6.dsw" or "all_samples_vc2008.sln" or "all_samples_vc2010.sln" depending on your Visual Studio version. Another way is to load the .vxcproj file in the folder of a specific demo project.

For example, to build the "pdf2text" demo, open "pdf2text/pdf2text_vc2010.vcxproj" with Visual Studio 2010 and build it. The executable file "pdf2text.exe" is generated in one of the following four folders as shown in Figure 3-4, which depends on the build configuration.

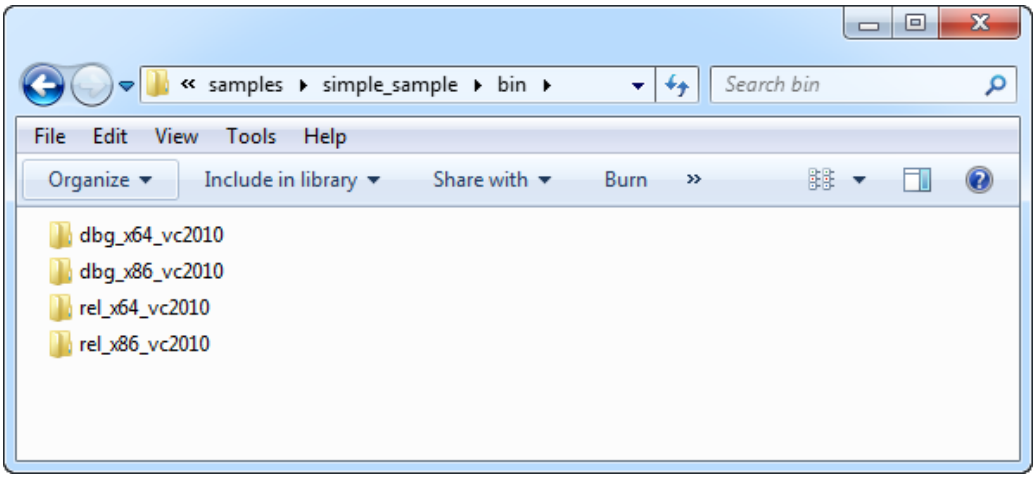

Figure 3-4

To run the executable file, in this example “bin/dbg_x86_vc2010/pdf2text.exe”, there are two options: in command line or in Visual Studio. When running in command line, start “cmd.exe”, go to “bin/dbg_x86_vc2010” and run “pdf2text”. The terminal output is shown in Figure 3-5. When running in Visual Studio, click on “Debug->Start Debugging” or “Debug->Start Without Debugging” on the menu bar to run pdf2text.exe. This is shown in Figure 3-6.

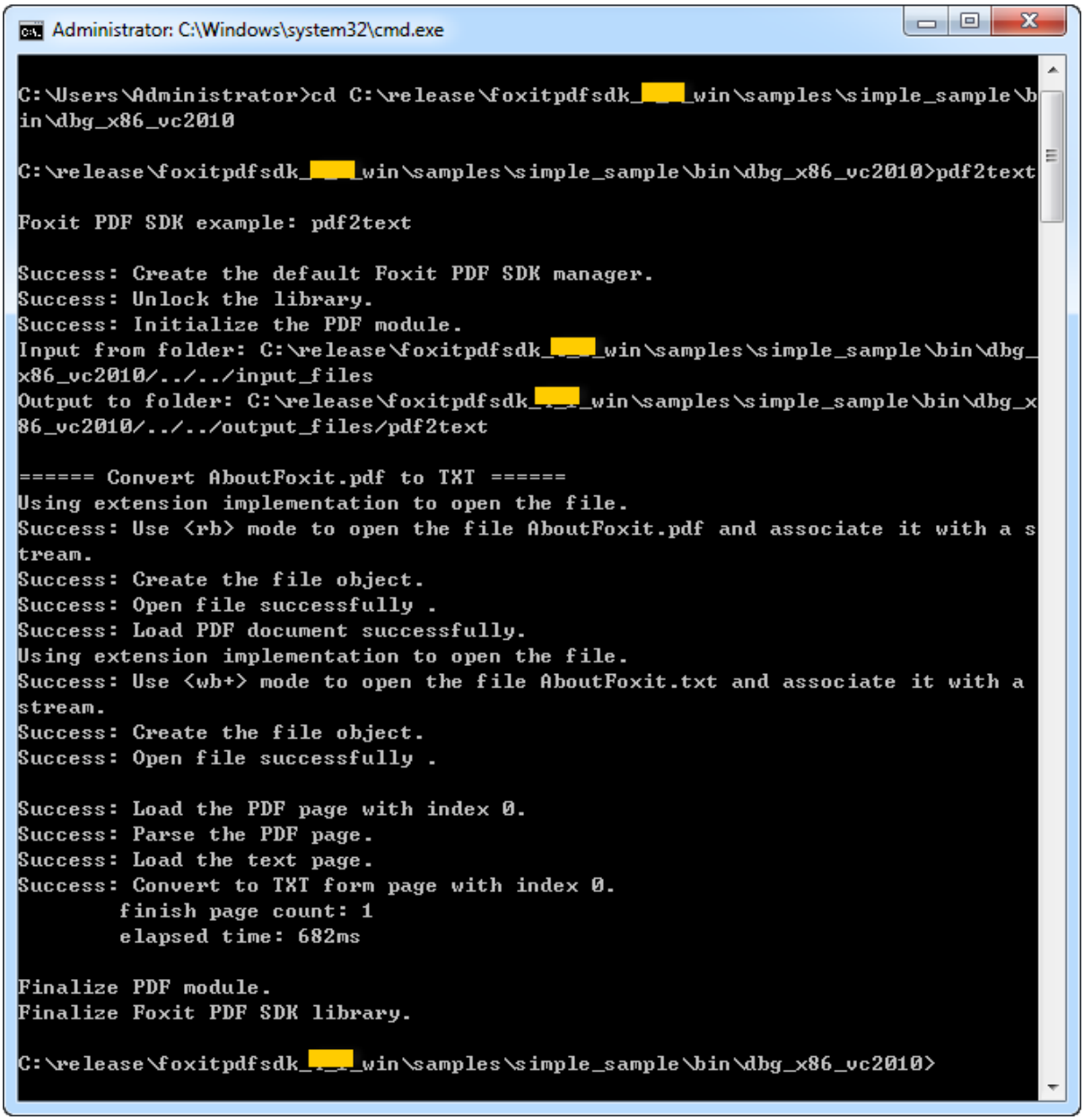

Figure 3-5

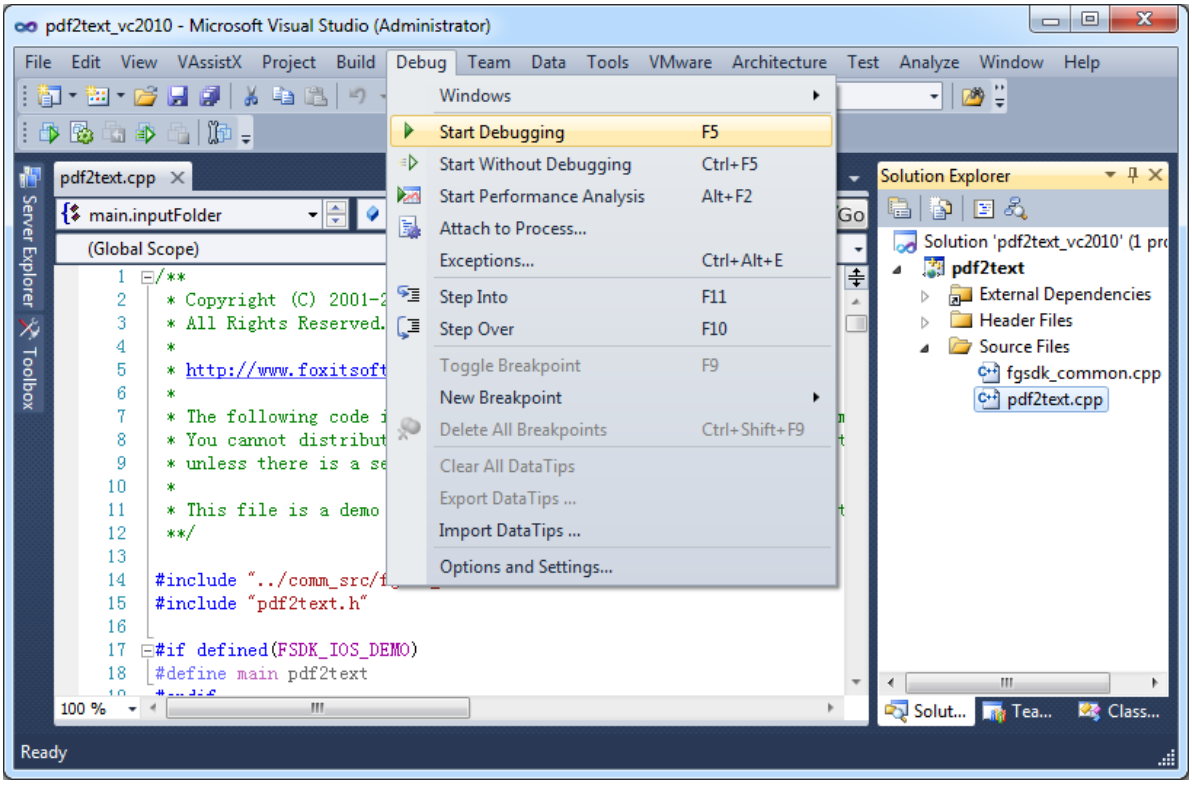

Figure 3-6

PDF Reader Demo

View demo provides an example for developing a PDF reader using PDF SDK APIs. To run this demo in Visual Studio,

a) Open “samples/view_demo/PDFReader_Demo/PDFReader.sln” in Visual Studio 2010 and build the solution.

b) Set up PDFReader as the startup project by right clicking the project “PDFReader” and select “Set as the Startup Project”. Press F5 to run the project.

c) After the demo starts, choose “File->Open” or click the directory icon to open a PDF file. Browse the content by scrolling down or move the PDF page by holding the left mouse button. Click the arrows or bookmark icon on the left toolbar, bookmark will show up. Click any page as you like. A screenshot of the demo is shown in Figure 3-7.

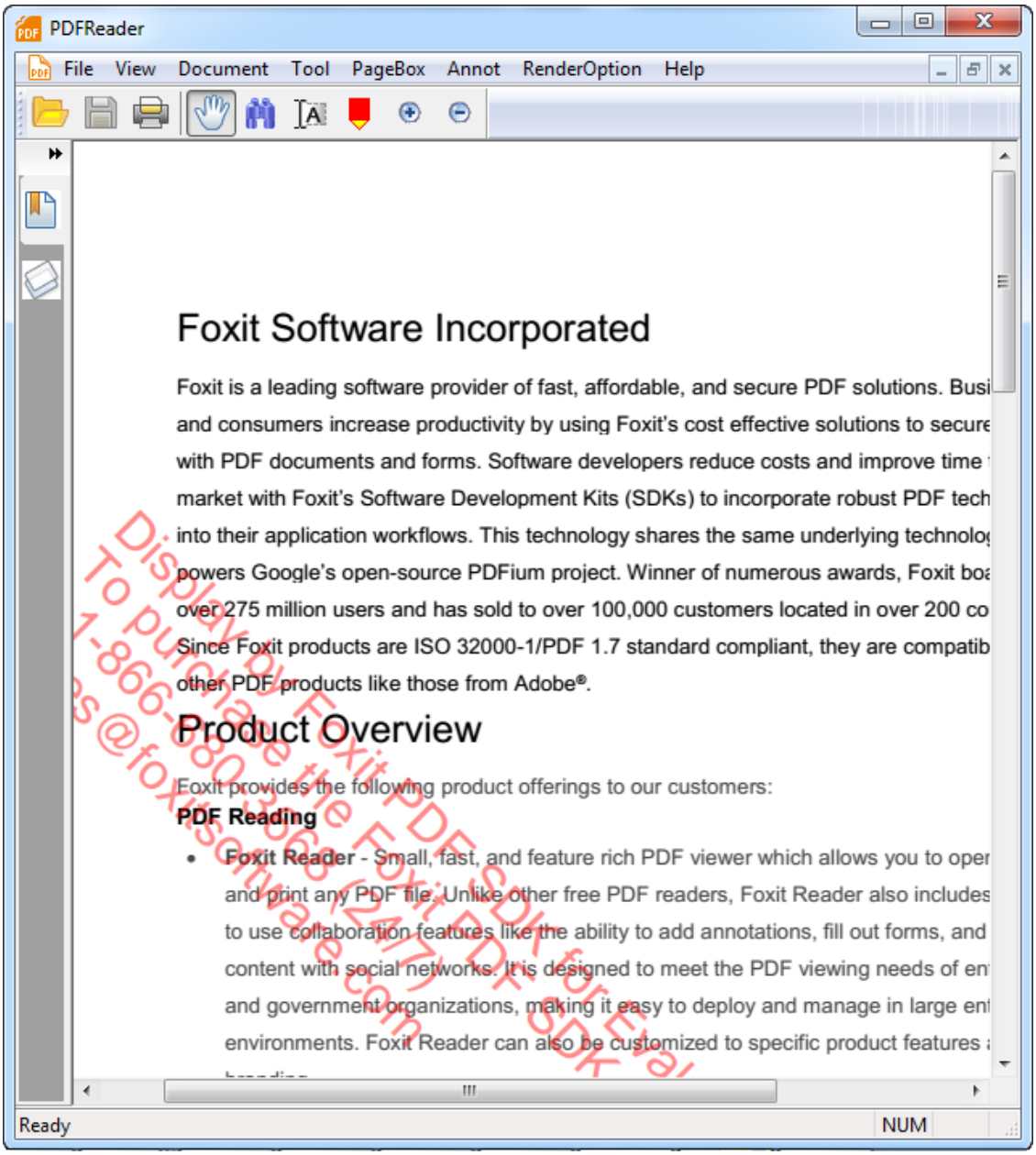

Figure 3-7

3.2.3 How to create your own project

In this section, we will show you how to create your own project by using Foxit PDF SDK. Create a Visual Studio Win32 console application called “test” and copy “include” and “lib” folders from the package to the project folder as shown in Figure 3-8.

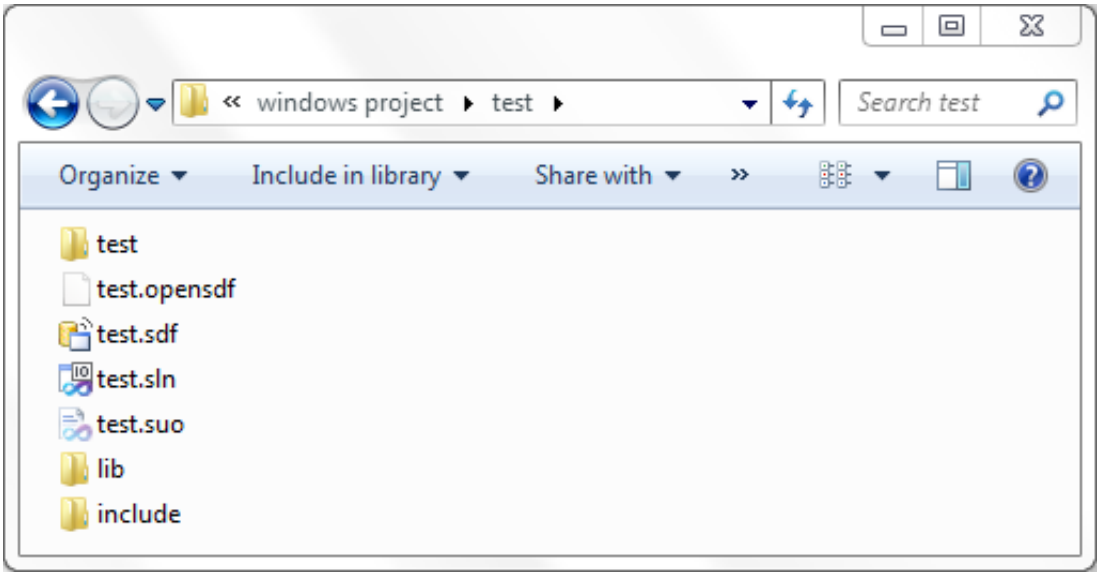

Figure 3-8

To run the project, follow the steps below.

a) Include the Foxit PDF SDK library and header files in the project. The corresponding code is shown in Figure 3-9.

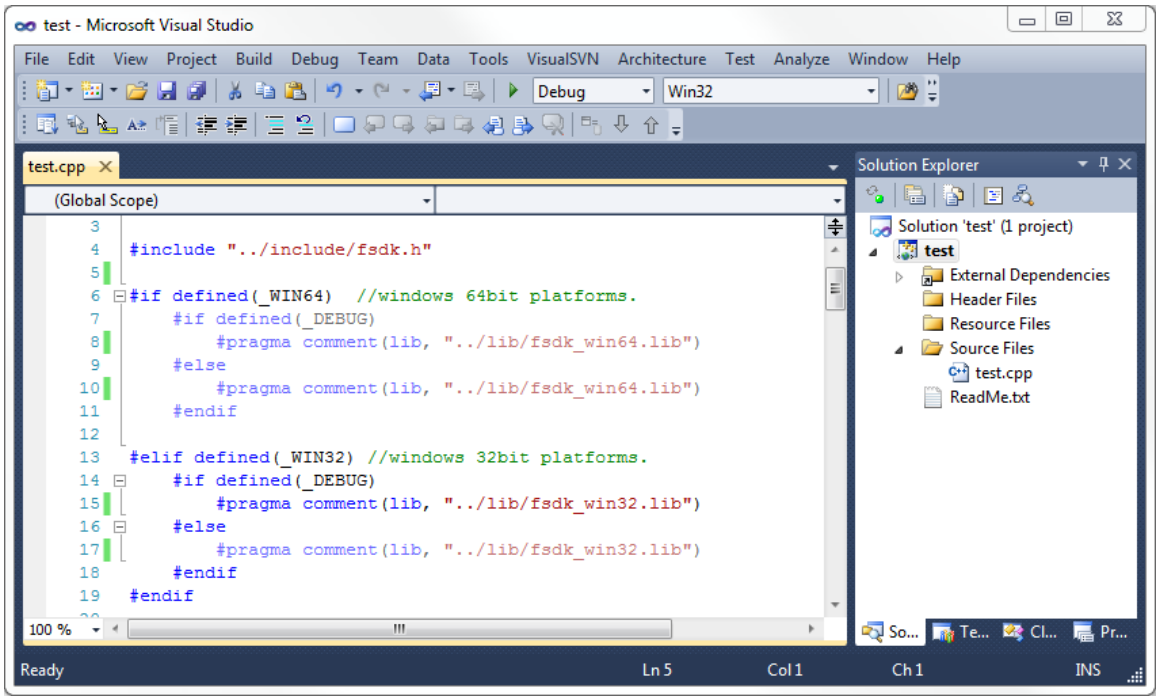

Figure 3-9

b) Construct the code to realize a PDF application. The necessary functions and the structure of the code are shown in Figure 3-10. To use PDF SDK APIs, include the following four steps in your application:

   1) Initiate Foxit SDK library manager.

   2) Apply a license to activate Foxit SDK.

   3) Realize PDF applications.

   4) Finalize Foxit SDK library manager. The specific way of applying a license is detailed in next section 3.2.4.

c) Build the project and copy the library file “fsdk_win64.lib” and “fsdk_win64.dll” (“fsdk_win32.dll” and “fsdk_win32.lib” in x86 system) to the “Debug” or “Release” folder where the executable files are generated. Run the executable and your project will get running!

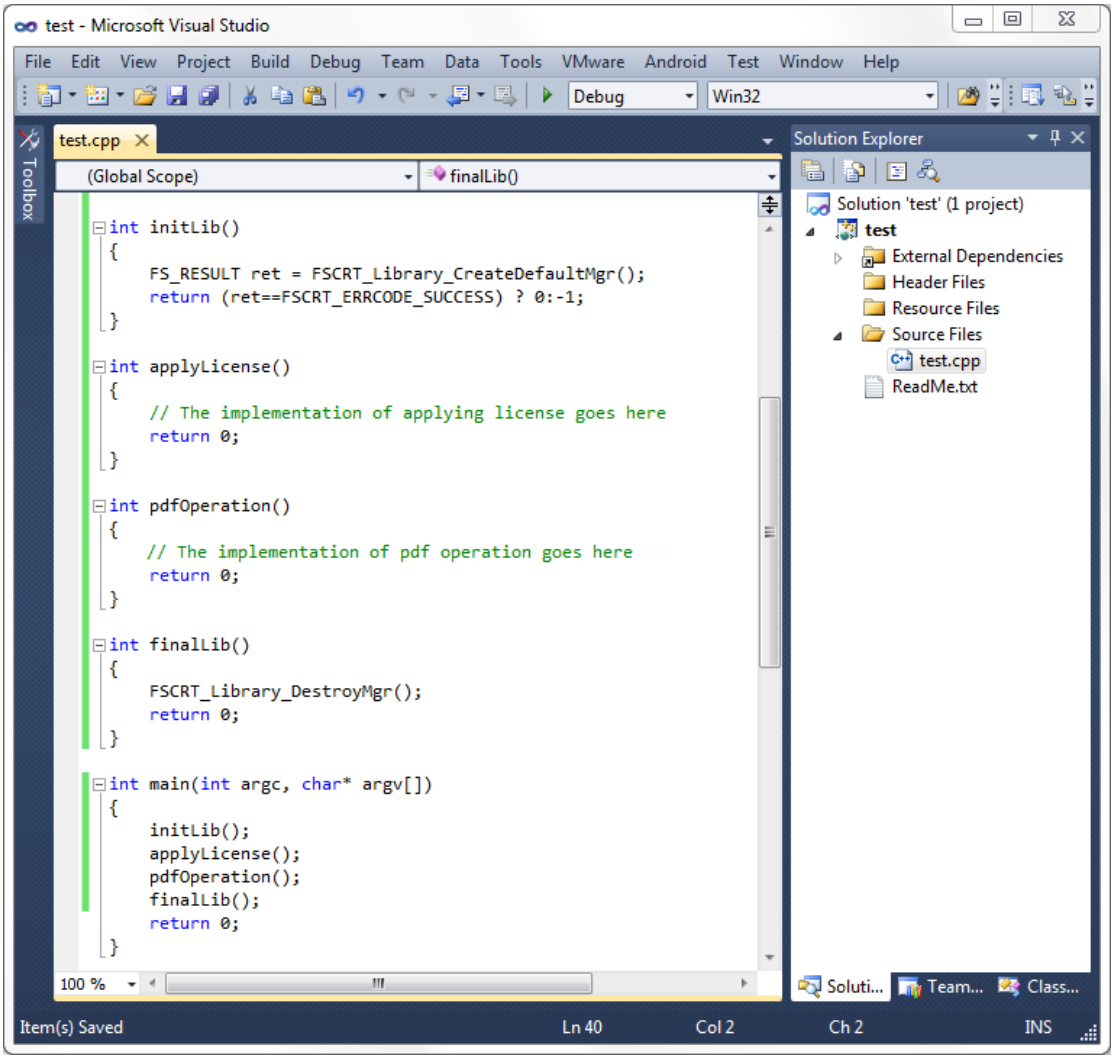

Figure 3-10

3.2.4 Unlock PDF SDK license

It's necessary for applications to unlock PDF SDK license before calling any APIs. Foixt PDF SDK provides a function **FSCRT_License_UnlockLibrary** that allows users to achieve hardcode unlocking license. Figure 3-11 shows the detailed steps. In this example, char* SN is from "gsdk_sn.txt" (the string after "SN=") and char* unlockStr is from "gsdk_key.txt" (the string after "Sign=".)

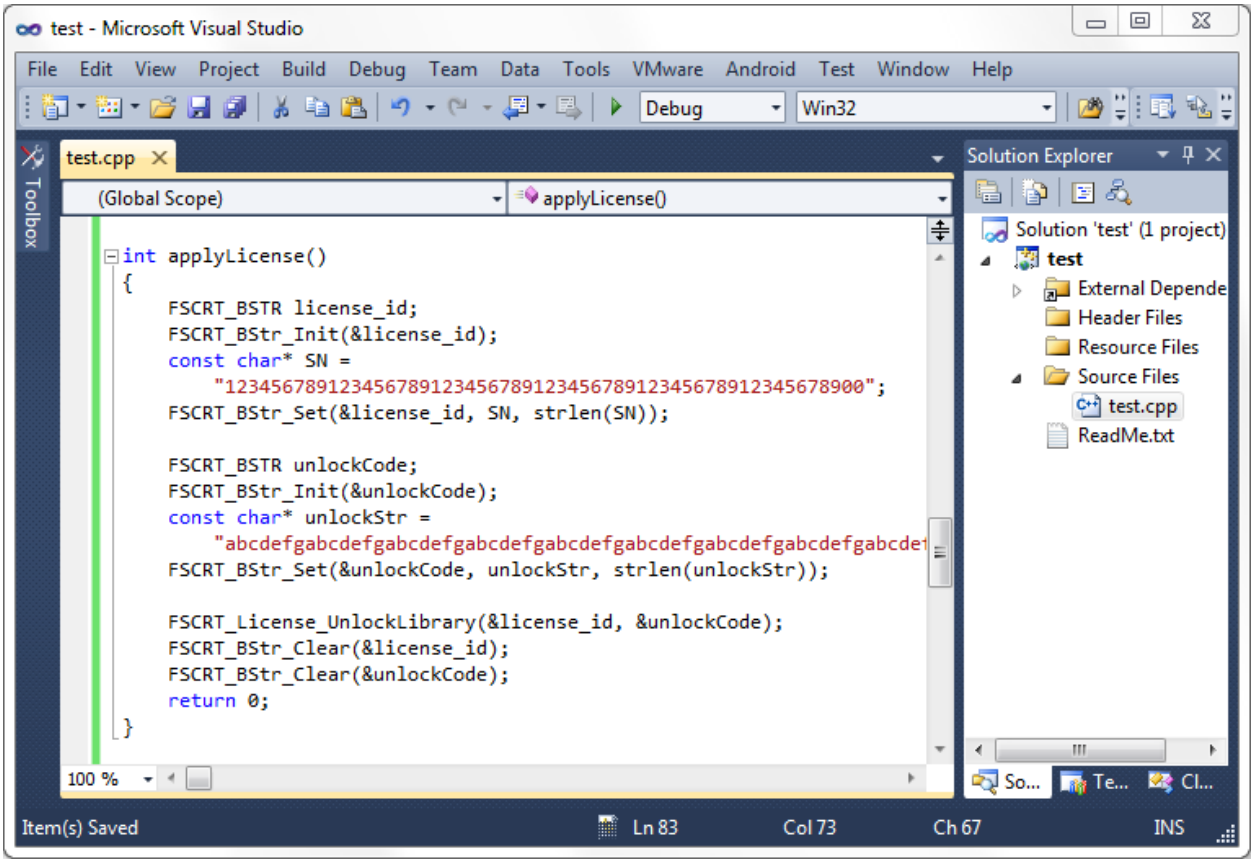

Figure 3-11

3.3 Linux

3.3.1 What's in this package

Download Foxit PDF SDK for Linux and untar it to a directory "foxitpdfsdk_4_1_linux".The structure of the release package is show in Figure 3-12. This package contains the following folders:

docs:	API references, demo tutorials, developer guide
include:	header files for foxit pdf sdk API
lib:	libraries and license files
samples:	sample projects and demos

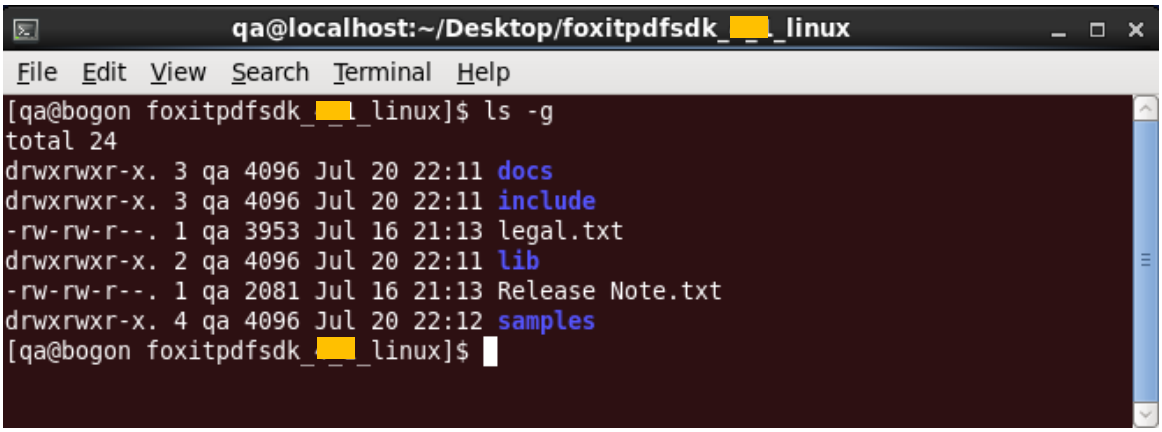

Figure 3-12

3.3.2 How to run a demo

Foxit PDF SDK for Linux provides more than 20 demo projects that cover a wide range of PDF document application. These demos are in folder “samples/simple_samples” as shown in Figure 3-13.

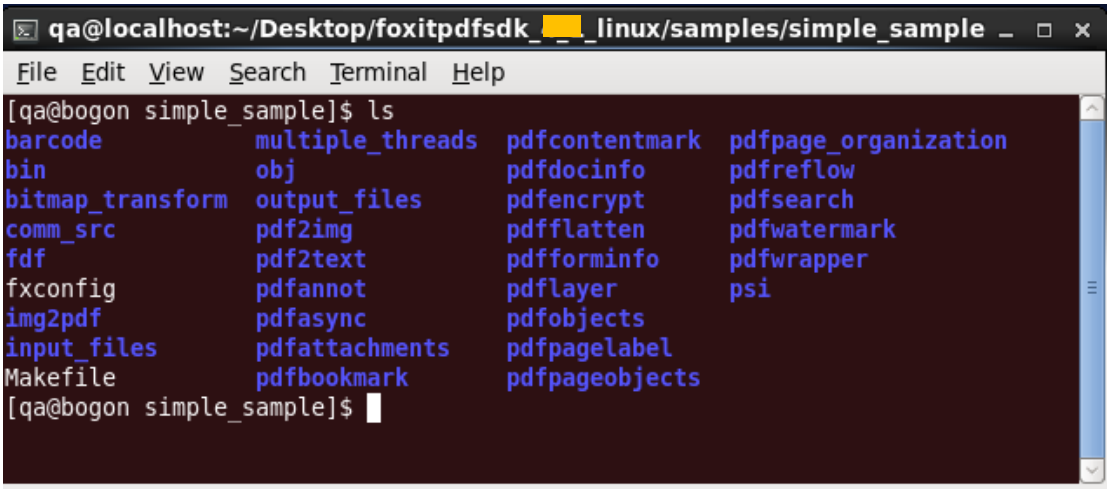

Figure 3-13

In a terminal window, run “make” or “make ver=debug” to build all demos or run “make project_name” to build a demo named “project_name”. For example, Figure 3-14 shows the build process for pdf2text demo.

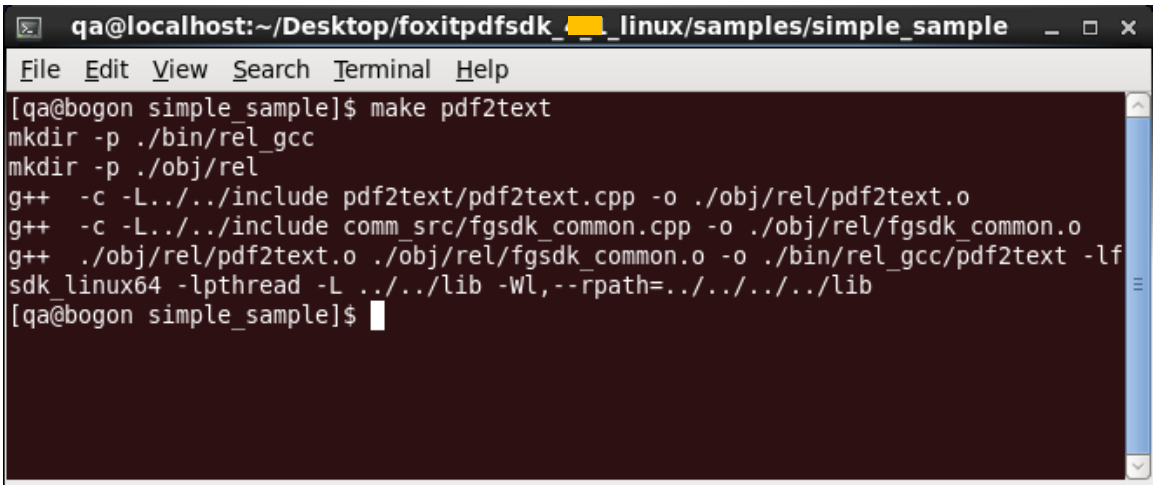

Figure 3-14

After building, the binary files are generated in folder “samples/simple samples/bin/rel_gcc” or “samples/simple samples/bin/dbg_gcc” depending on the build option. In that folder, run the binary file to get the demo running. Figure 3-15 shows a screen shot when running pdf2text demo.

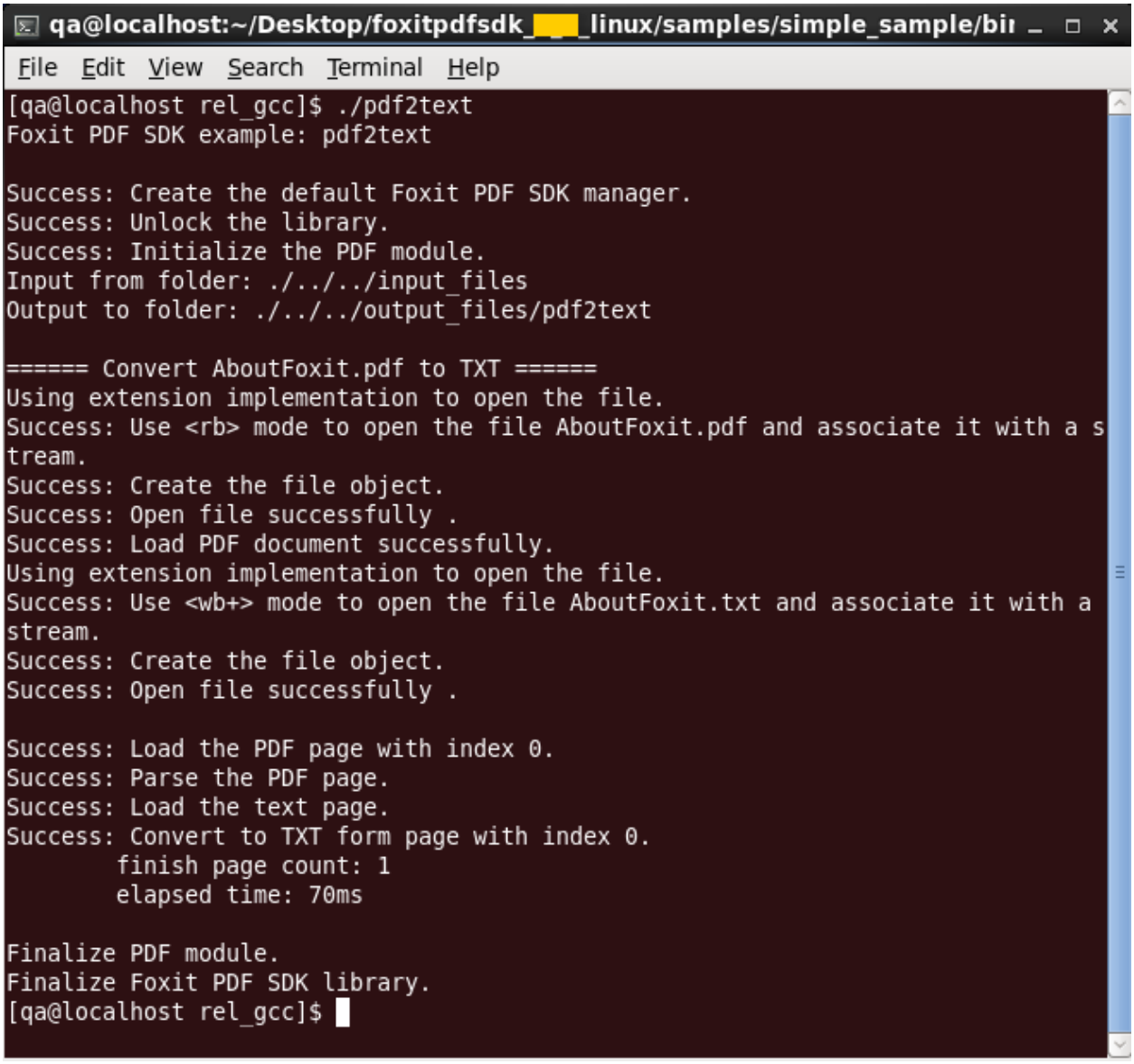

Figure 3-15

Some demos generate output files (pdf, text or image files). These output files are put in a folder under "samples/simple_samples/output_files/". In this example, it is "samples/simple_samples/output_files/pdf2text".

3.3.3 How to create your own project

Suppose you are creating a project in a folder called "test". After finishing the following steps, the folder structure will be like Figure 3-16.

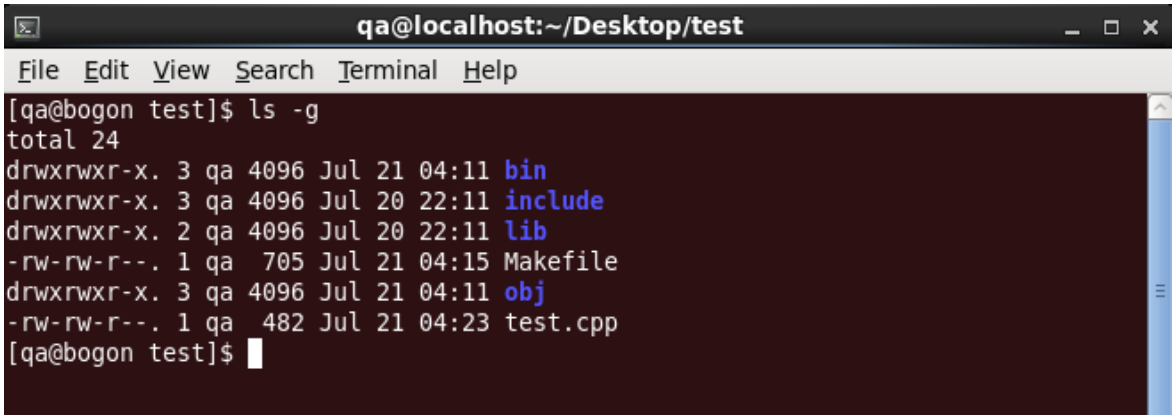

Figure 3-16

a) Copy “include” and “lib” folders from the PDF SDK package to “test”. Create a “test.cpp” file that includes “fsdk.h”. “fsdk.h” is a basic header file that includes other PDF SDK header files.

b) Figure 3-17 shows what a PDF application shall prepare for calling PDF SDK APIs. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.

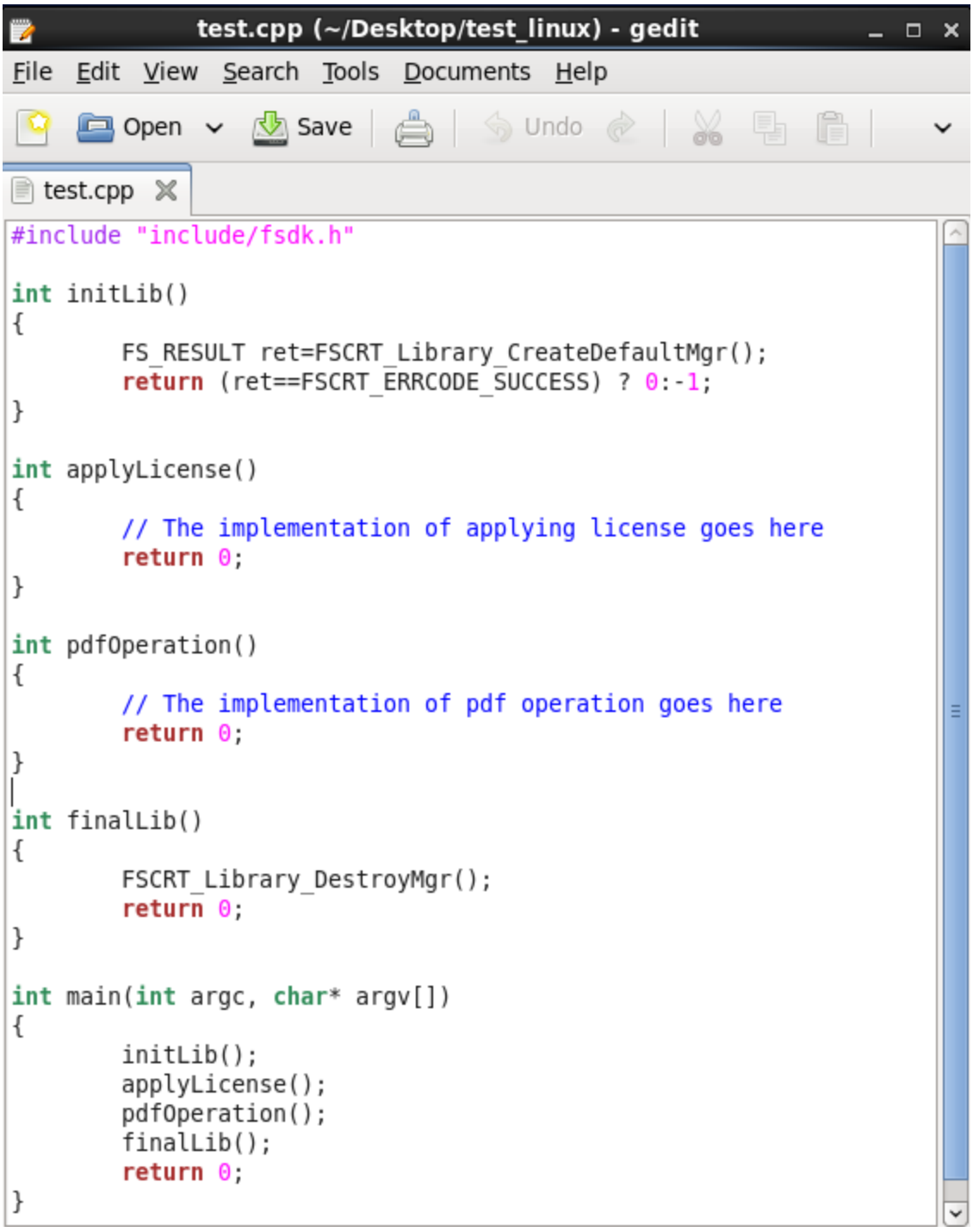

Figure 3-17

c) Create a Makefile. In this Makefile, the PDF SDK library shall be included in the build path. Use libfsdk_linux64.so for 64 bit system or libfsdk_linux32.so for 32 bit system. A sample Makefile is shown in Figure 3-18.

d) Run “make” to generate binary file in “test/bin/rel_gcc” and you are ready to go on your application!

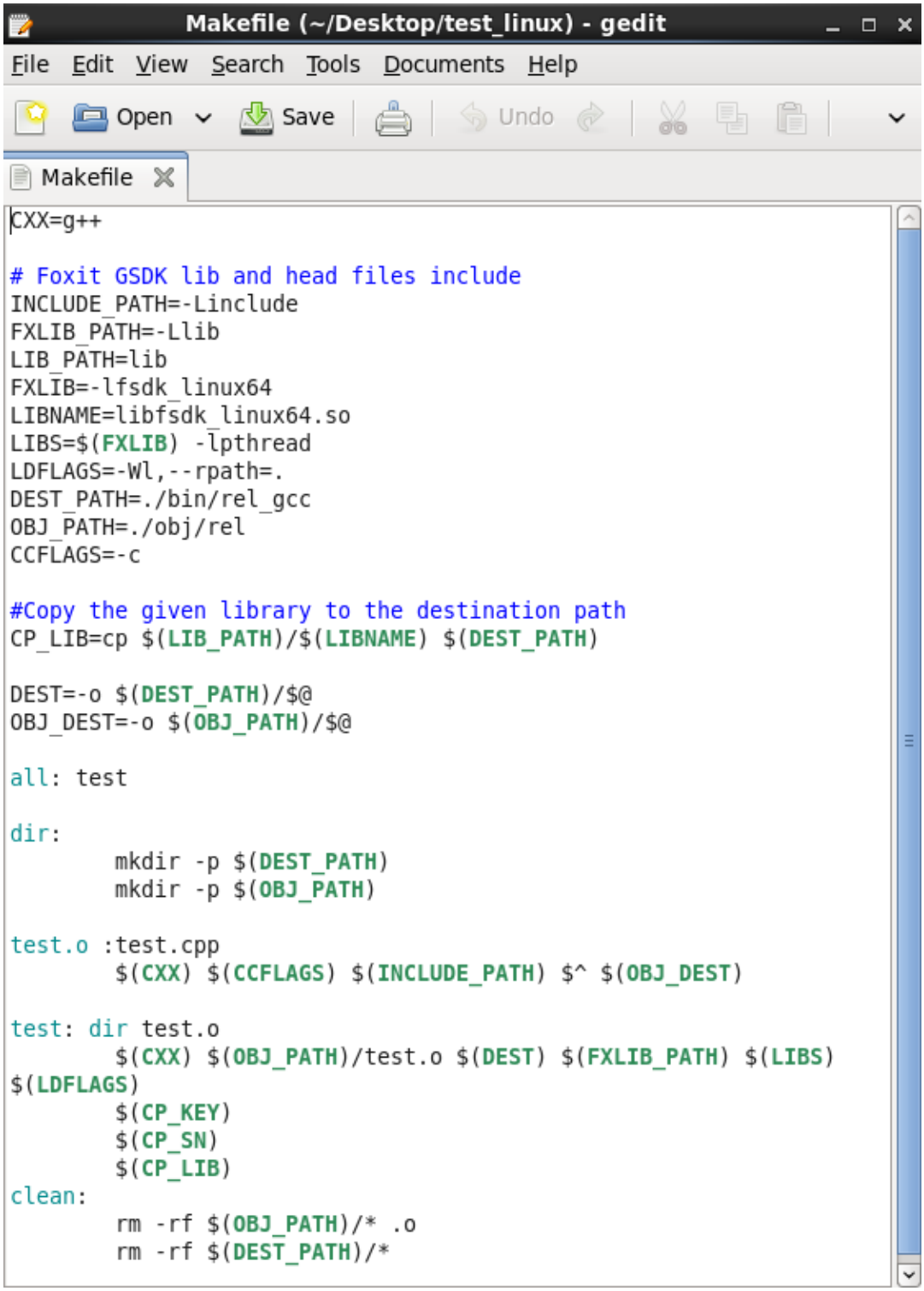

Figure 3-18

3.4 Mac

3.4.1 What's in this package

Download Foxit PDF SDK for mac and untar it to a directory like "foxitpdfsdk_4_1_mac". The structure of the release package is show in Figure 3-19. This package contains the following folders:

docs: API references, demo tutorials, developer guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos

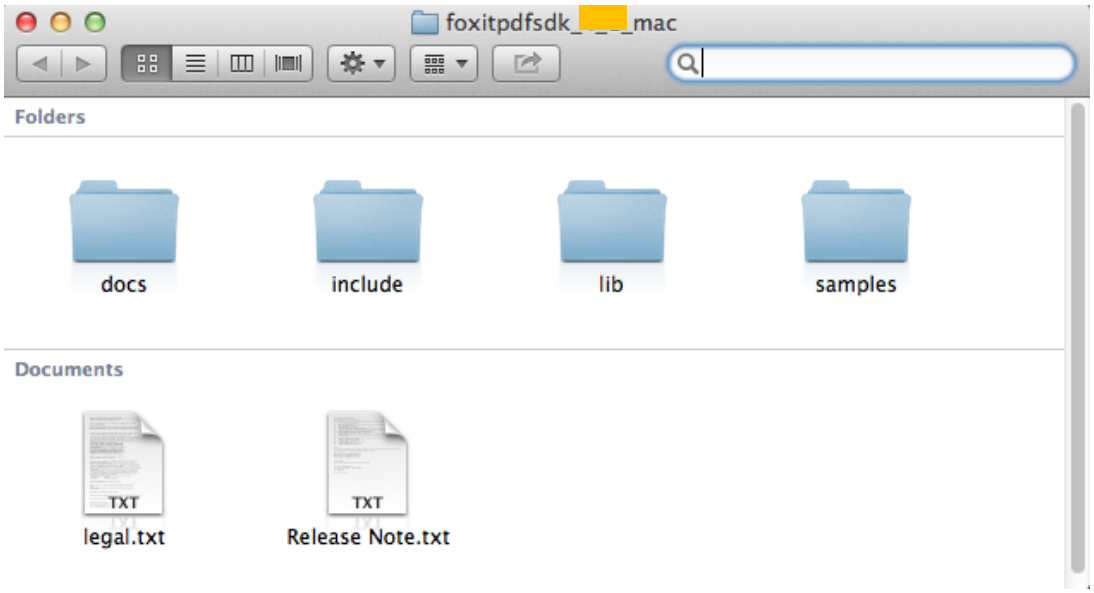

Figure 3-19

3.4.2 How to run a demo

Foxit PDF SDK for mac provides more than 20 demo projects that cover a wide range of PDF document application. These demos are in folder "samples/simple_samples" as shown in Figure 3-20.

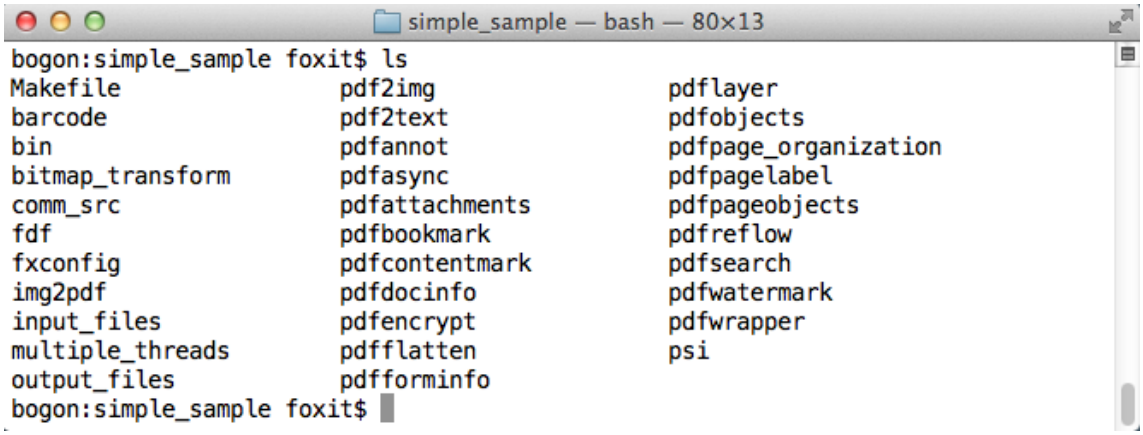

Figure 3-20

In a terminal window, run “make” or “make ver=debug” to build all demos or run “make project_name” to build a demo named “project_name”. For example, Figure 3-21 shows the build process for pdf2text demo.

Figure 3-21

After building, the binary files are generated in “samples/simple samples/bin/rel_gcc” or “samples/simple samples/bin/dbg_gcc” depending on the build option. In that folder, run the binary file to get the demo running. Figure 3-22 shows the screen output when running pdf2text demo.

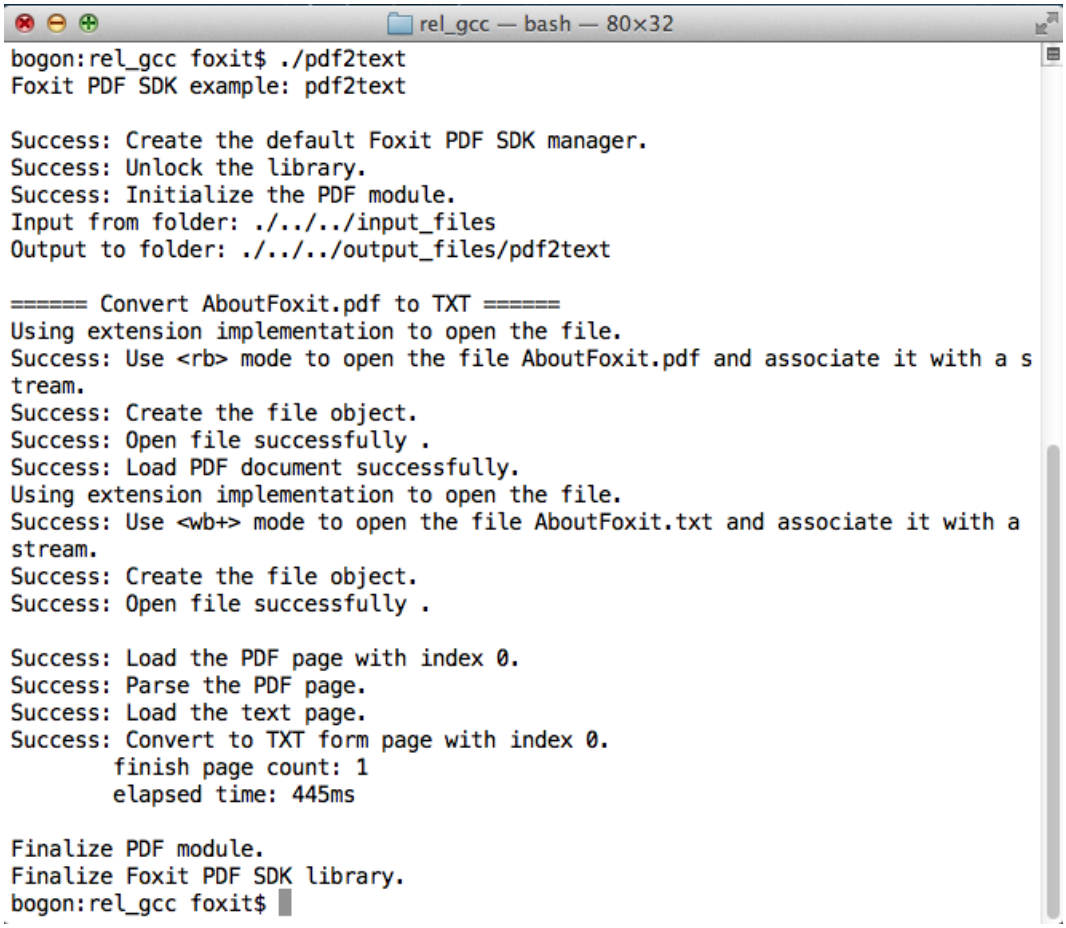

Figure 3-22

Some demos generate output files (pdf, text or image files). These output files are generated in a folder under “samples/simple_samples/output_files/”. In this example, it is “samples/simple_samples/output_files/pdf2text”.

3.4.3 How to create your own project

Suppose you are creating a project in a folder called “test”. After finishing the following steps, the folder structure will be like Figure 3-23.

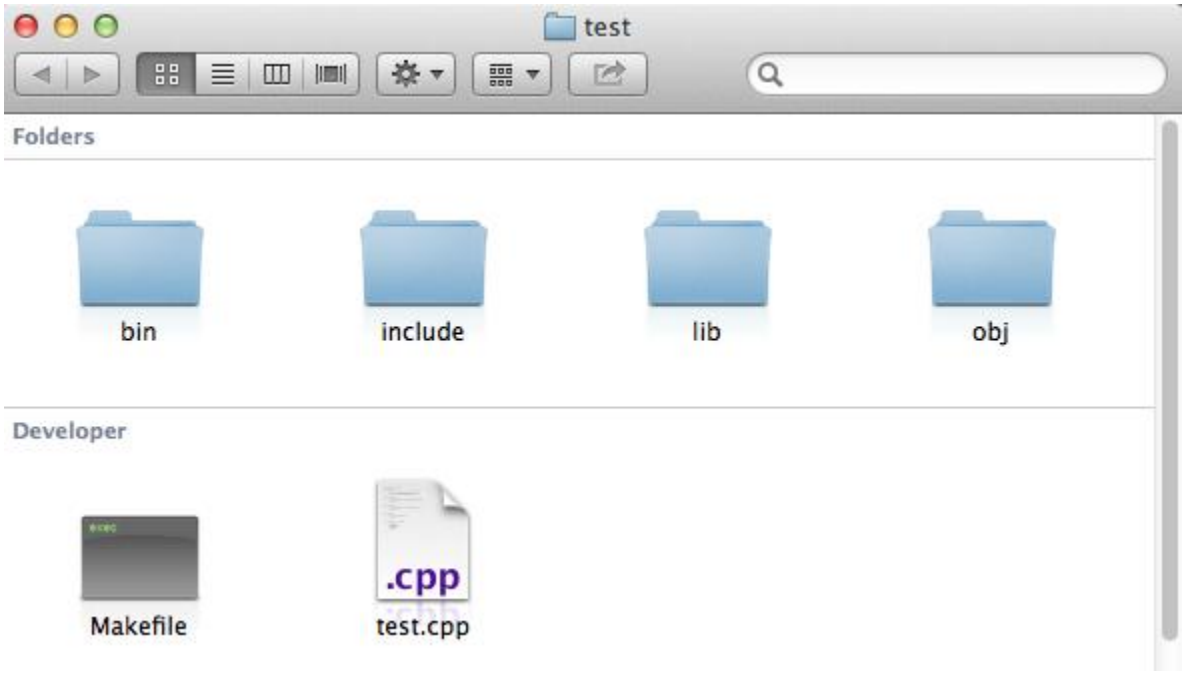

Figure 3-23

a) Copy “include” and “lib” folders from the PDF SDK package to “test”. Create a “test.cpp” file that includes “fsdk.h”. “fsdk.h” is a basic header file that includes other PDF SDK header files.

b) Figure 3-24 shows what a PDF application shall prepare for calling PDF SDK APIs. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.

test.cpp

test.cpp › initLib()

```
#include "include/fsdk.h"

int initLib()
{
    FS_RESULT ret=FSCRT_Library_CreateDefaultMgr();
    return (ret==FSCRT_ERRCODE_SUCCESS) ? 0:-1;
}

int applyLicense()
{
    // The implementation of applying license goes here
    return 0;
}

int pdfOperation()
{
    // The implementation of pdf operation goes here
    return 0;
}

int finalLib()
{
    FSCRT_Library_DestroyMgr();
    return 0;
}

int main(int argc, char* argv[])
{
    initLib();
    applyLicense();
    pdfOperation();
    finalLib();
    return 0;
}
```

Figure 3-24

c) Create a makefile. In this make file, the PDF SDK library shall be included in the build path. Please use libfsdk_mac64.a. A sample ‘makefile’ is shown in Figure 3-25.

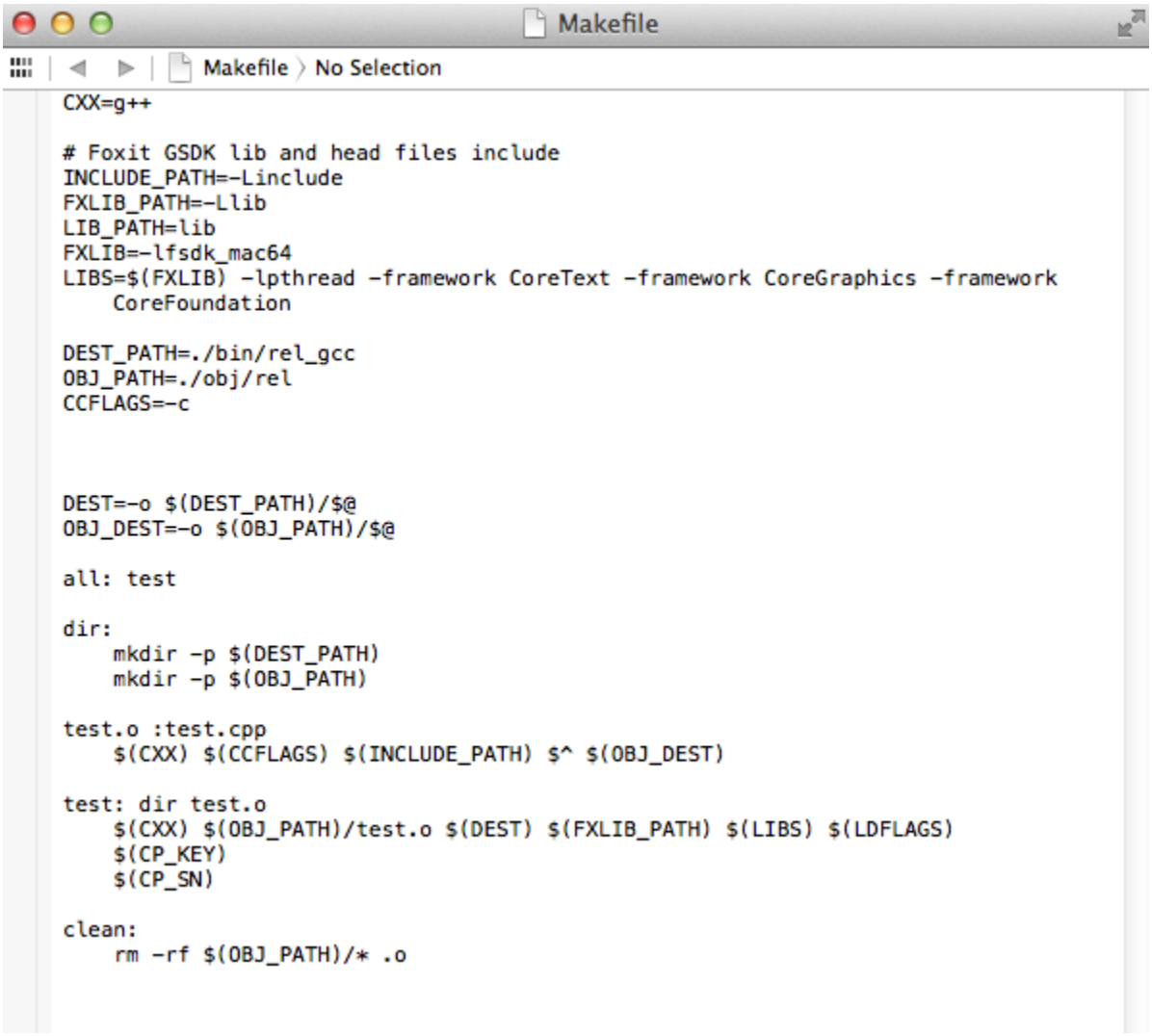

Figure 3-25

d) Run “make” to generate binary file in “test/bin/rel_gcc” and you are ready to go on your application!

3.5 iOS

3.5.1 What’s in the package

Download Foxit PDF SDK for iOS and untar it to a directory “foxitpdfsdk_4_1_ios”.The structure of the release package is show in Figure 3-12. This package contains the following folders:

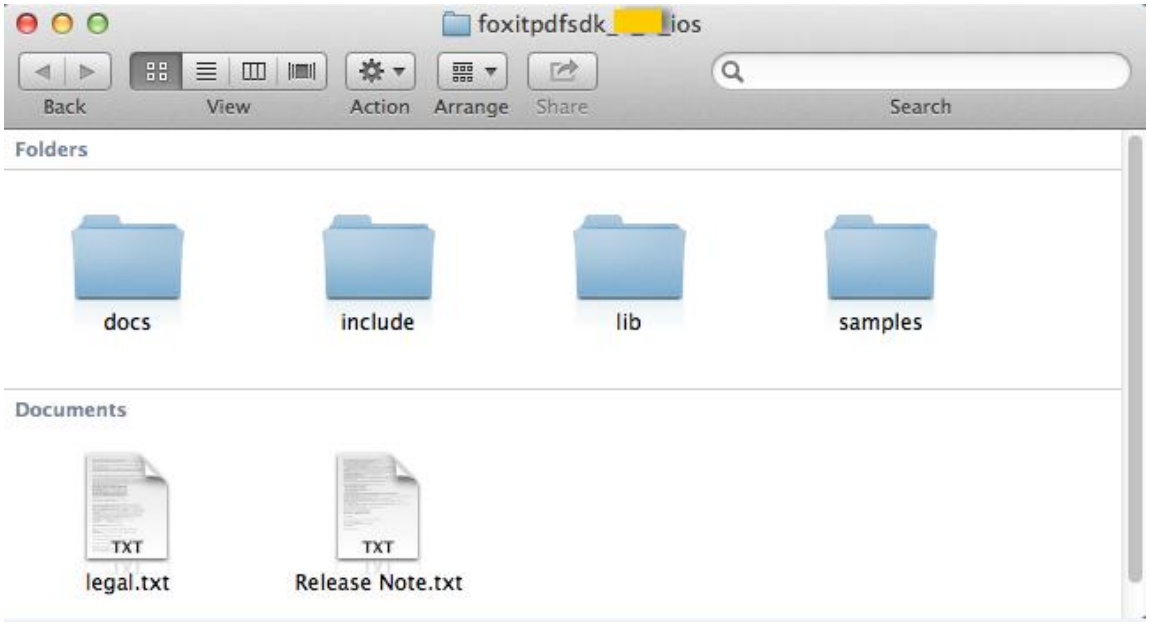

Figure 3-26

docs:	API references, demo tutorials, developer guide
include:	header files for foxit pdf sdk API
lib:	libraries and license files
samples:	sample projects and demos

In “samples”, there are two types of demos. “samples/simple_sample” contains more than 20 demos that cover a wide range of PDF applications. “samples/view_demo” contains two demos that realizes a simple PDF viewer and an OOM recovery schedule respectively.

For the first type of demos under “samples/simple_sample” directory, input files for all demos are put in “input_files”, output files for all demos are put in “**sandbox**”. A snapshot of “samples/simple_sample” folder is shown in Figure 3-27.

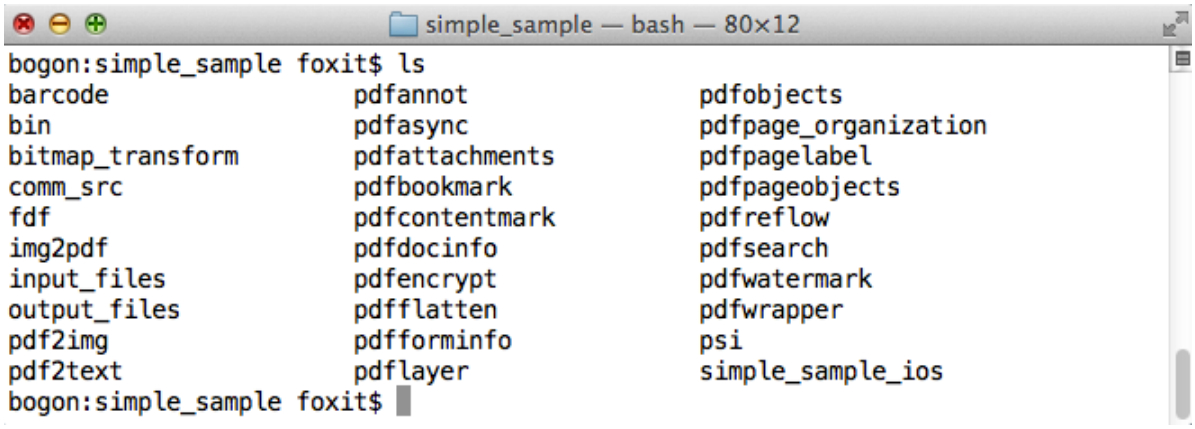

Figure 3-27

"samples/view_demo" contains a demo_view and an oom_demo which is shown in Figure 3-28.

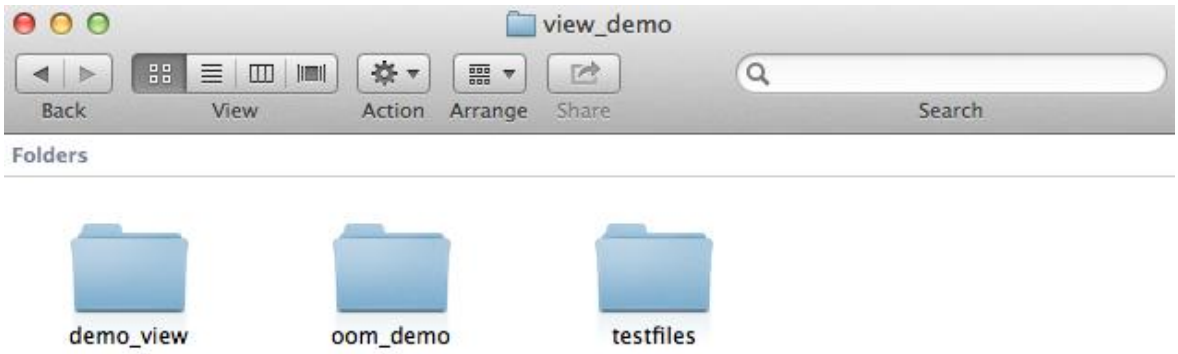

Figure 3-28

3.5.2 How to run a demo

Simple Demo

Simple demo projects provide examples for developers on how to effectively use PDF SDK APIs to complete their applications. To run a demo in Xcode, load the Xcode solution files "simple_sample_ios.xcodeproj" under "samples/simple_sample/simple_sample_ios" folder which is shown in Figure 3-29. Here the Xcode version is 5.0.2.

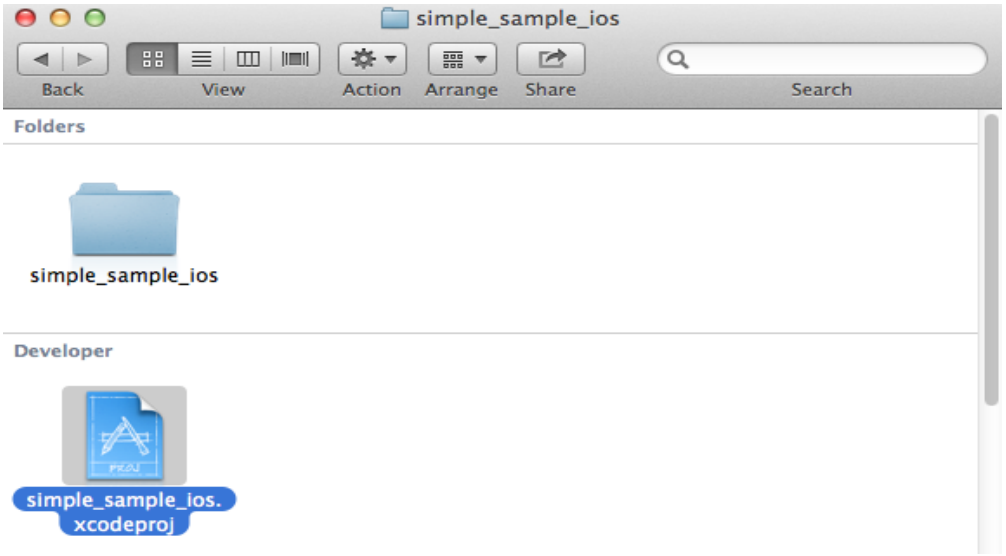

Figure 3-29

After loading the solution, click on "Run" on the menu bar to build the solution. A screenshot of the project is shown in Figure 3-30. The output files for all demos are put in "**sandbox**".

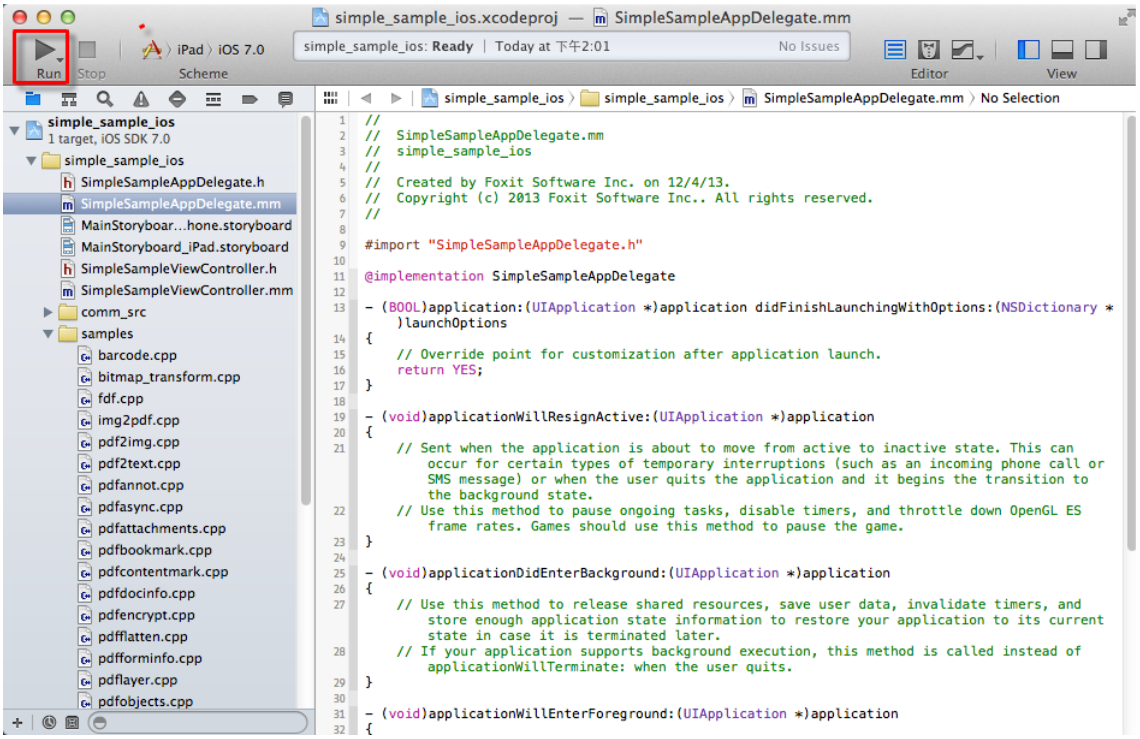

Figure 3-30

After Building the solution successfully, the iOS simulator will be started as shown in Figure 3-31.

Figure 3-31

For example, click on “pdf2text”, the log information will be shown in Figure 3-32.

Figure 3-32

PDF View Demo

View demo contains a demo to realize a simple PDF viewer under “samples/view_demo/demo_view” and a demo to show the OOM recovery schedule under “samples/view_demo/oom_demo”.

For the “**demo_view**”, here is the way to run it in Xcode,

a) Open “samples/view_demo/demo_view/demo_view.xcodeproj” in Xcode and click on “Run” to build the solution.

b) After the iOS simulator starts, click on “Open PDF”, then a default PDF “AboutFoxitCorporation.pdf” under “samples/view_demo/testfiles” folder will be displayed as shown in Figure 3-33.

Figure 3-33

c) Click the “AboutFoxit.PDF”, the contents of the PDF will be displayed. Click the small triangle menu to show the function buttons as shown in Figure 3-34.

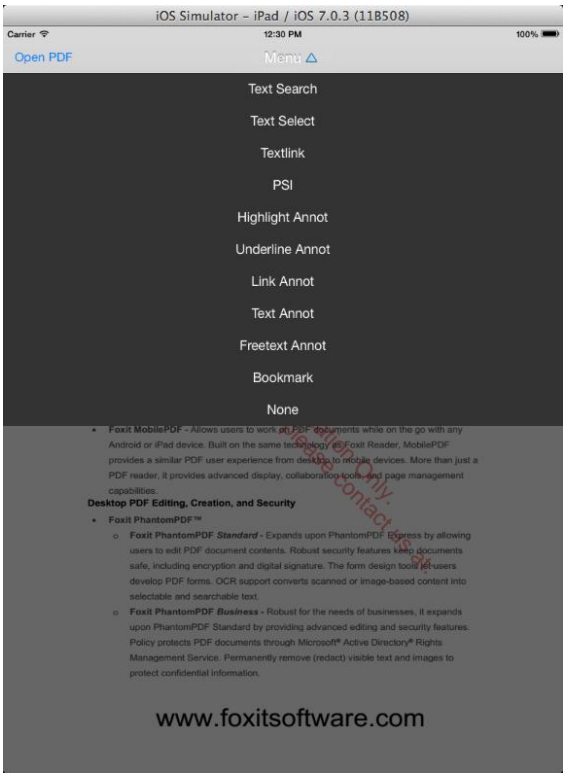

Figure 3-34

d) For example, click the “Highlight Annot”, then select texts in the PDF page by holding the left mouse button. The result is shown in Figure 3-35.

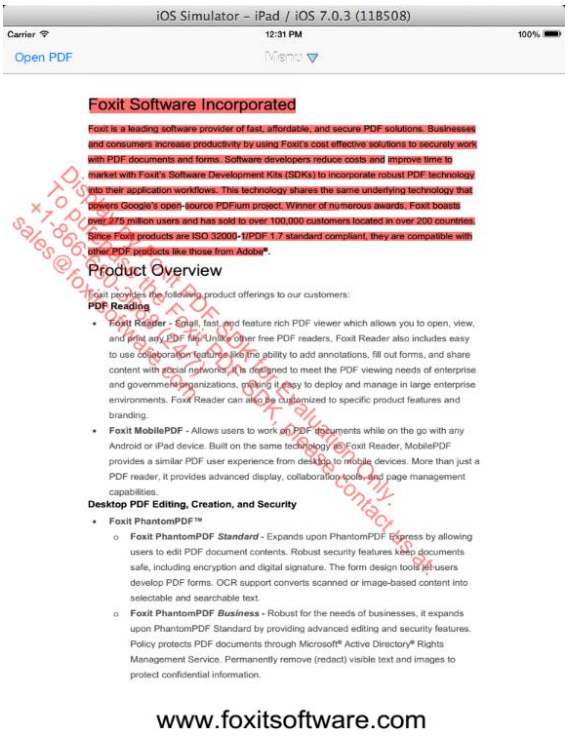

Figure 3-35

For the "**oom_demo**", here is the way to run it in Xcode,

a) Open "samples/view_demo/oom_demo/oom_demo.xcodeproj" in Xcode and click on "Run" to build the solution.

b) After the iOS simulator starts, click on "Open PDF", then a default PDF "FoxitBigPreview.pdf" under "samples/view_demo/testfiles" folder will be displayed as shown in Figure 3-36.

Figure 3-36

c) Click the “FoxitBigPreview.pdf”, then a dialog will pop up to warn that the memory is not enough to parse pages and ask you if want to enlarge 4M bytes for the memory. This is shown in Figure 3-37.

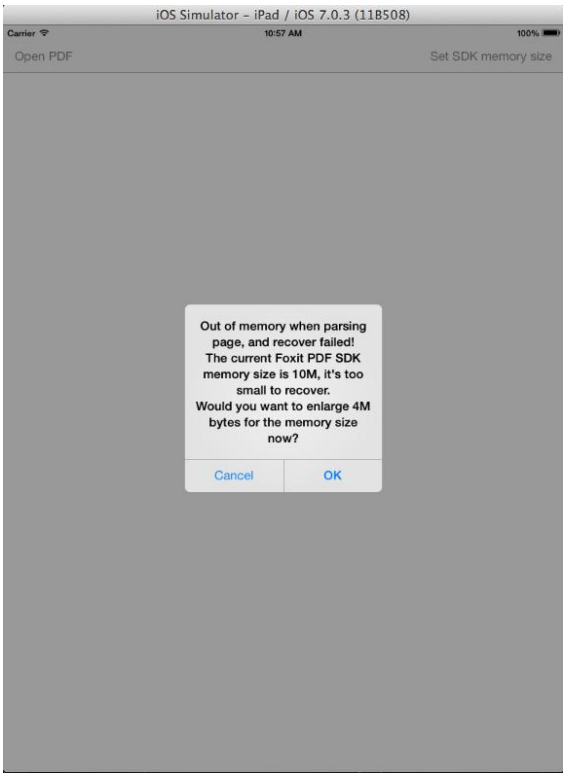

Figure 3-37

d) If you click on “OK”, here will pop up another dialog to warn that the memory is also not enough to parse pages and ask you if need to enlarge 4M bytes for the memory, which is shown in Figure 3-38.

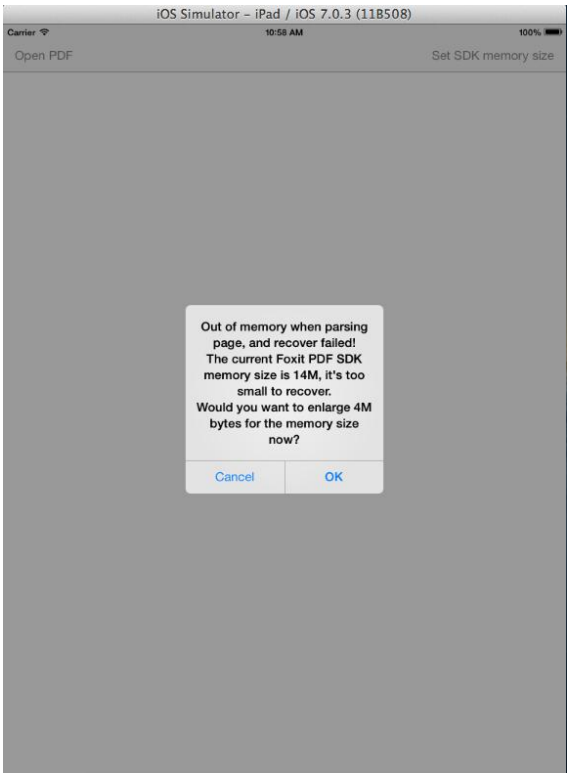

Figure 3-38

e) If you also click on “OK”, the contents of the “FoxitBigPreview.pdf” file will display as shown in Figure 3-39. It means that the PDF SDK could continuously allocate memory until it can parse the pages of the PDF file. In this demo, the memory step size is 4M.

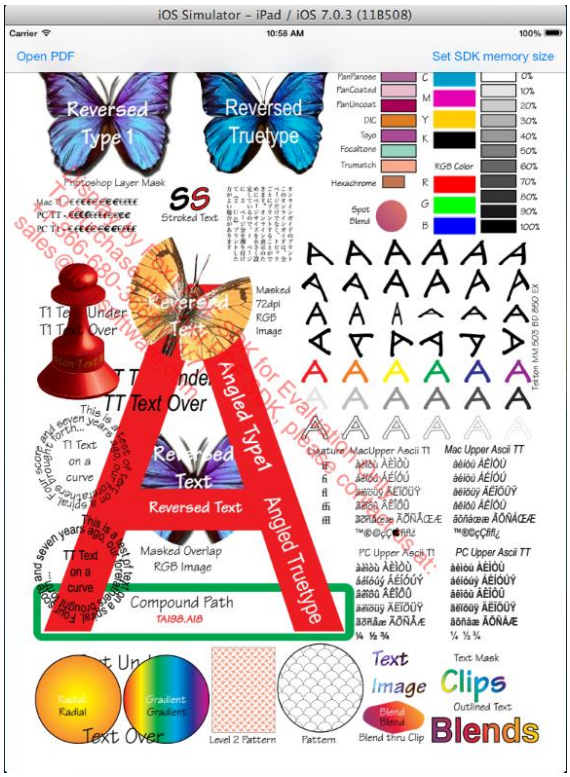

Figure 3-39

f) In Figure 3-37, if you click on “Cancel”, the memory size will not be automatically increased. In that case, you can click the “Set SDK memory size” to set it by yourself. For example, Figure 3-40 shows that setting the memory size to 20M. After clicking on “OK”, you can see the same result with Figure 3-39.

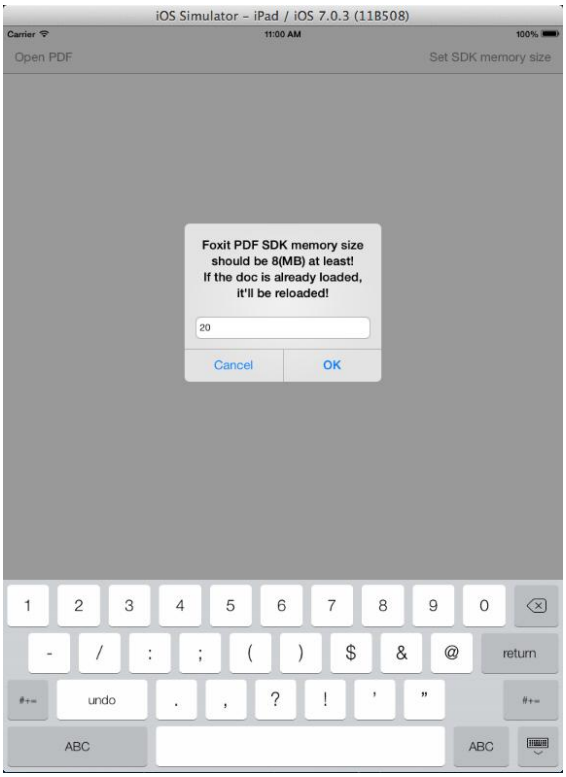

Figure 3-40

3.5.3 How to create your own project

Suppose you are creating a new “Single View Application” iOS project called test. After finishing the following steps, the folder structure of the test project will be like Figure 3-41.

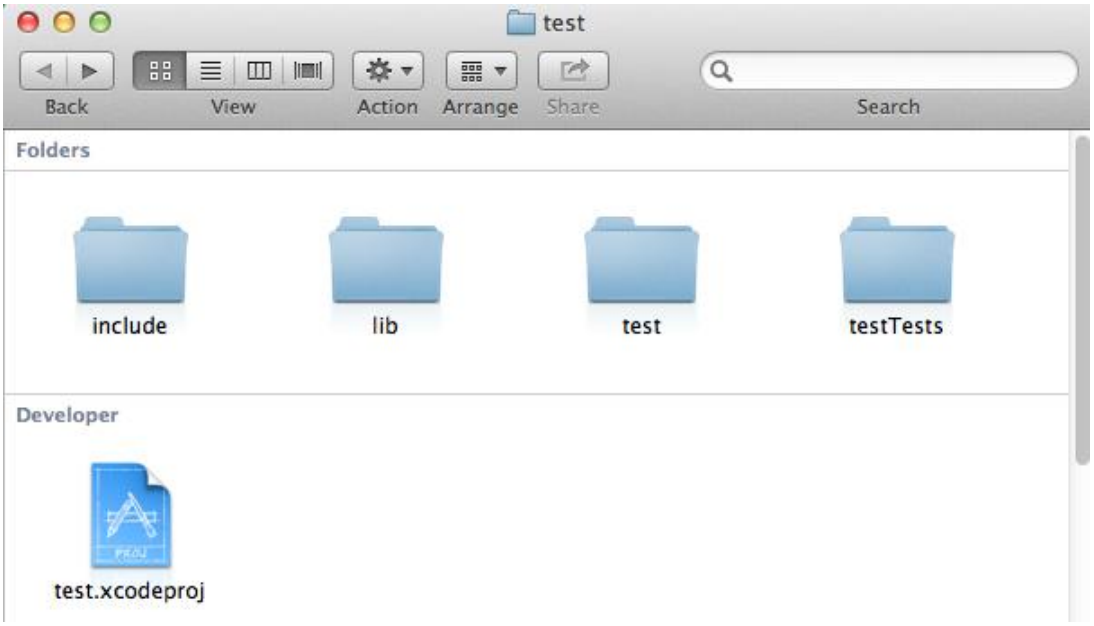

Figure 3-41

a) Copy “include” and “lib” folders from the PDF SDK package to “test”.

b) Open the “test.xcodeproj” to load the solution file, modify the suffix of “.m” to “.mm” which is used for changing the .m file to an Objective-C++ file, such as “ViewController.m” and “AppDelegate.m” under “test/test” folder, which will ensure that the C++ library required by PDF SDK is included at link time.

c) Copy “libfsdk_ios_arm.a” and “libfsdk_ios_emu.a” in the folder of “test/lib” to the test project as shown in Figure 3-42.

d) The right part of the Figure 3-42 shows what a PDF application shall prepare for calling PDF SDK APIs. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.

e) Click on “Run” to build the solution and you are ready to go on your application!

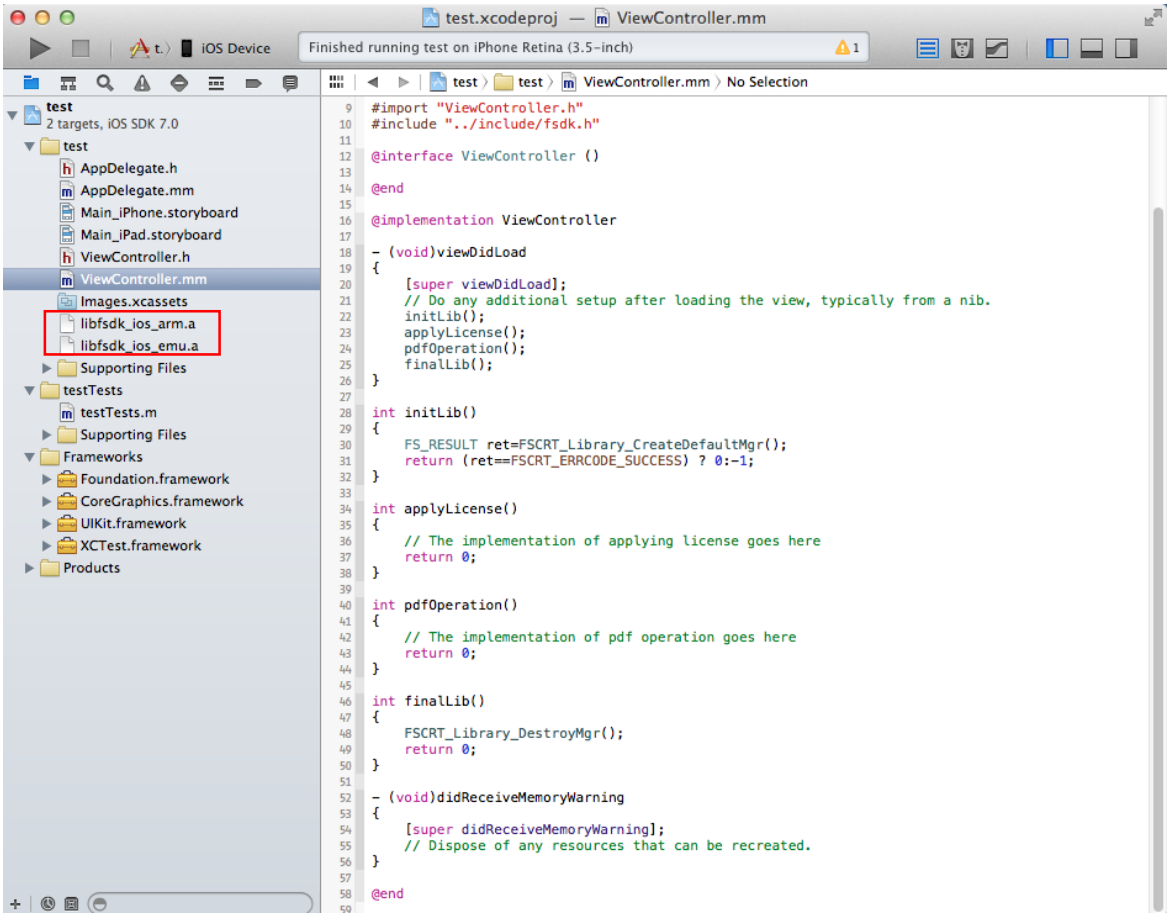

Figure 3-42

3.6 Android

3.6.1 What's in the package

Download Foxit PDF SDK for Android C APIs and untar it to a directory "foxitpdfsdk_4_1_Android_c".The structure of the release package is shown in Figure 3-43. This package contains the following folders:

docs: API references, developer guide

include: header files for foxit pdf sdk API

libs: libraries and license files

samples: sample projects and demos

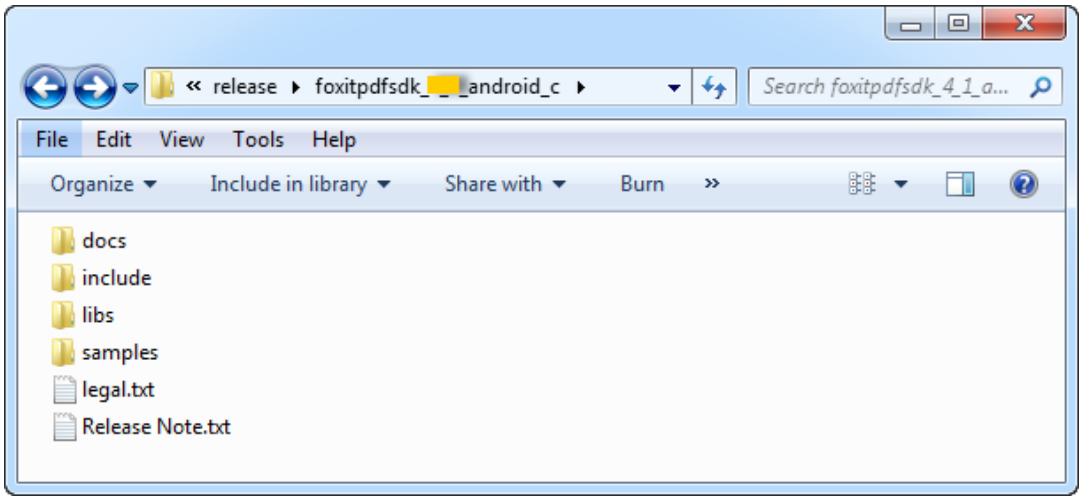

Figure 3-43

3.6.2 How to run a demo

Foxit PDF SDK for Android C APIs provides one viewer demo in folder “samples”. The resources and files of this demo are put under “samples/ViewerDemo” as shown in Figure 3-44.

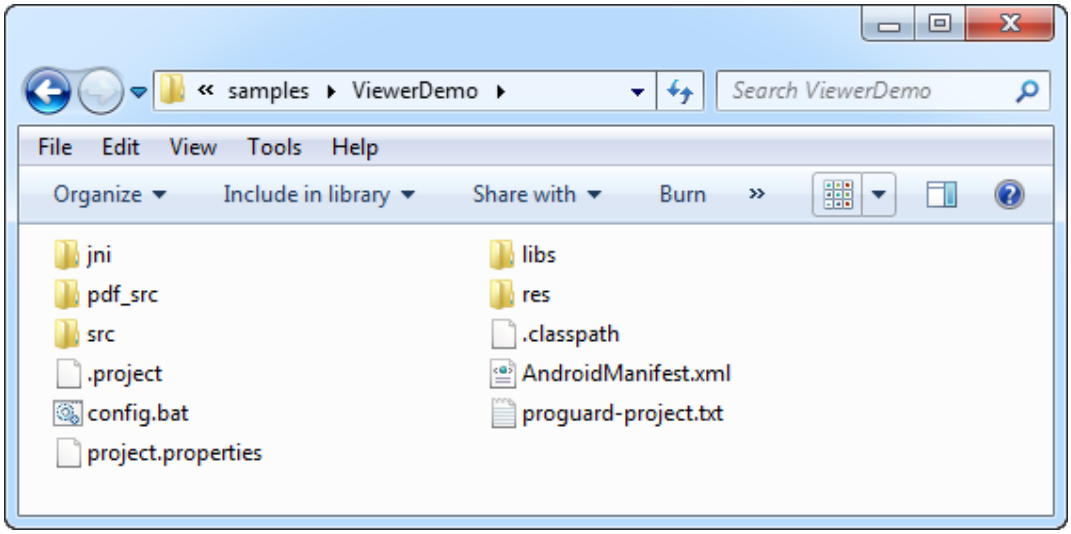

Figure 3-44

To run the demo of C APIs in Android platform, the JNI (Java Native Interface) is playing an important role. JNI can facilitate users to develop applications in Android platform with native-code languages such as C and C++. To use JNI, please download and install Android NDK (Native Development Kit) at first. The NDK is a toolset that helps developers to compile the C/C++ codes and generate dynamic libraries which are called by upper Java interfaces.

The Android NDK can be downloaded from http://developer.android.com/tools/sdk/ndk/index.html. Please choose the latest NDK package that is appropriate for your computer and then download the package, uncompress the NDK package by using tools available on your computer. After that, the NDK files are contained in a directory called android-ndk-<version>. Now the current version is r10, so the directory is called android-ndk-r10.

After installing the NDK successfully, if you run the demo on Windows, Cygwin also needs to be installed. Please download it from http://www.cygwin.com/ and install it. Here we assume that the running system is Windows. To build ".a" libraries with NDK and run the demo in Eclipse, follow the steps below:

a) Set environment variables. Put the directory of "android-ndk-r10" which contains "ndk-build" file into the "Path" of the system variables.

b) Open Cygwin, go to "/samples/ViewerDemo/jni/arm_mk/jni", here we build the "libfsdk_android_arm.a" library under "/foxitpdfsdk_4_1_android_c/libs" folder and some other ".cpp" files under "/samples/ViewerDemo/jni" folder. If your operating system is x86, you can build the "libfsdk_android_x86.a" library under "/foxitpdfsdk_4_1_android_c/libs".

c) Run "ndk-build", and the "libfsdk_android.so" library will be located in folder "ViewerDemo/jni/arm_mk/libs/armeabi", which is shown in Figure 3-45.

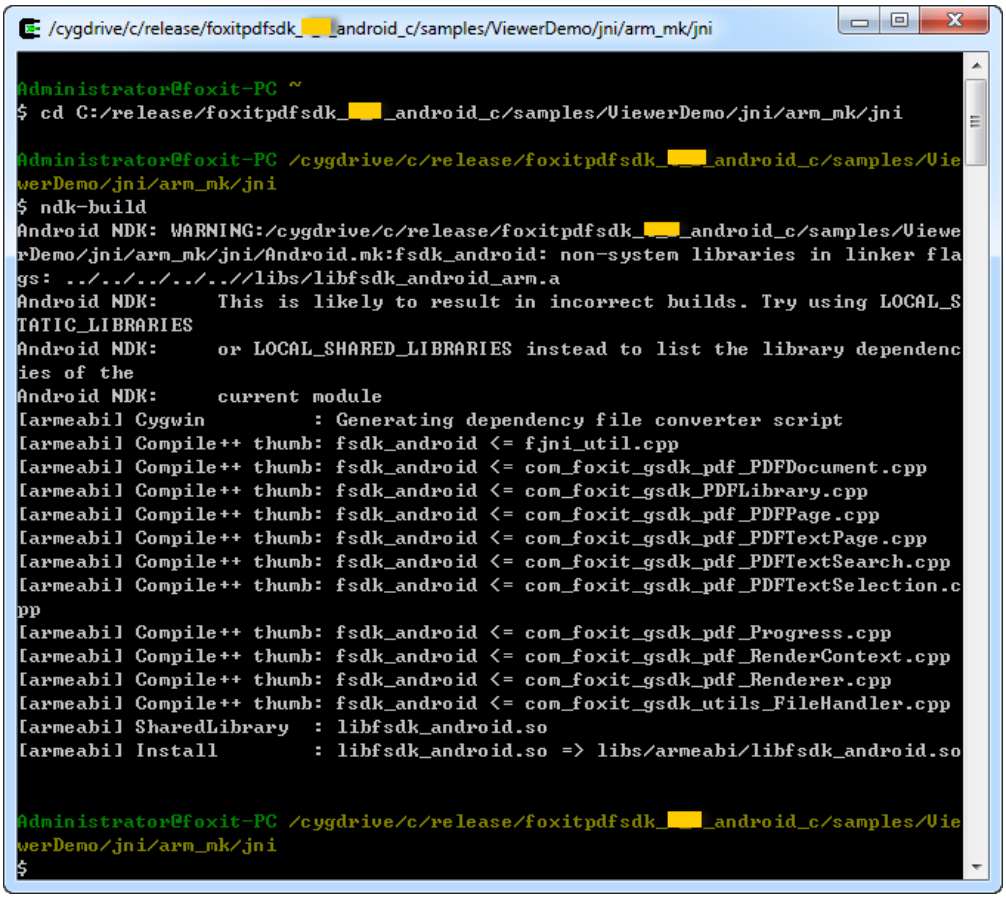

Figure 3-45

d) Open Eclipse, import the “ViewerDemo” project. Click on “Run->Run as->Android Application” to run the demo, here we assume that we have created an AVD targeting 4.4.2 and pushed a PDF file “AboutFoxit.pdf” on this device. Figure 3-46 shows the demo.

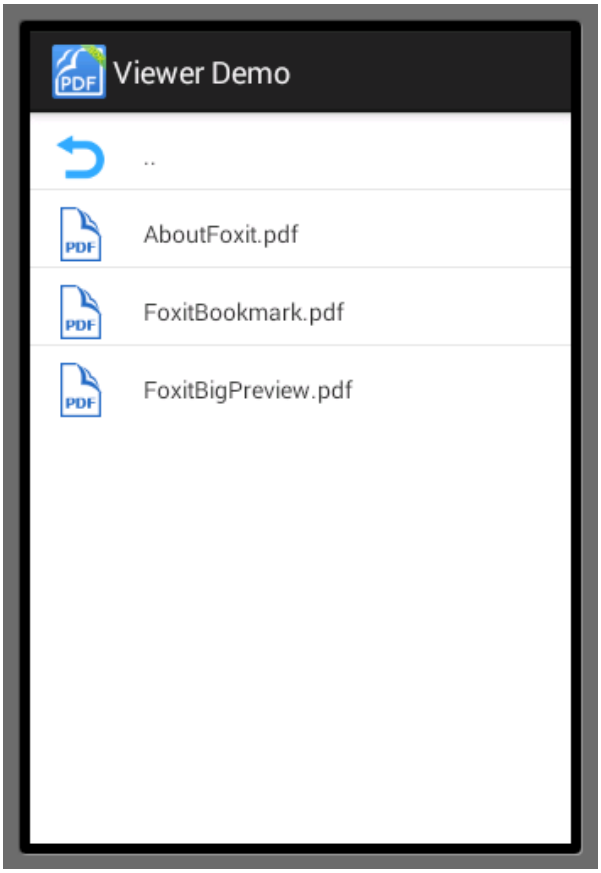

Figure 3-46

e) Click the “AboutFoxit.pdf”, the PDF file will be displayed as shown in Figure 3-47.

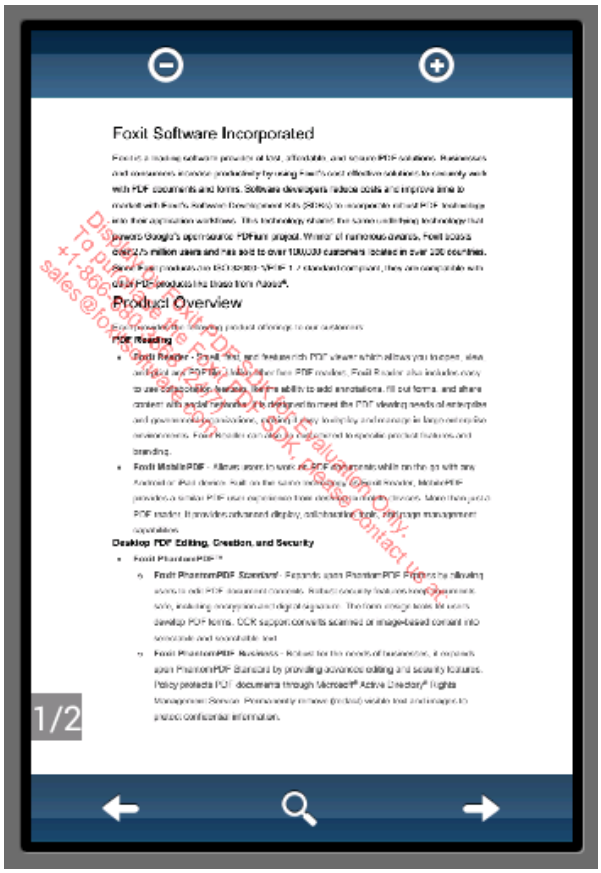

Figure 3-47

f) This demo provides the functionalities like page turning, zooming, text search and extraction. For example, click the search button, type word “overview”, and press the “Enter” key, the first search result will be highlighted as shown in Figure 3-48.

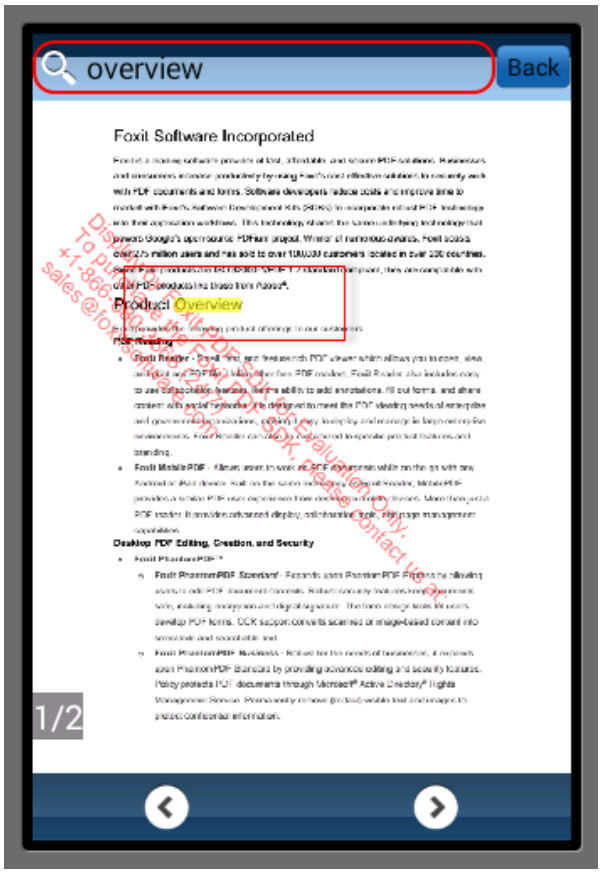

Figure 3-48

3.6.3 How to create your own project

Suppose you are creating a new android application project called test. The package name is com.foxit.test. Once created, the directory structure of the test project will be like Figure 3-49.

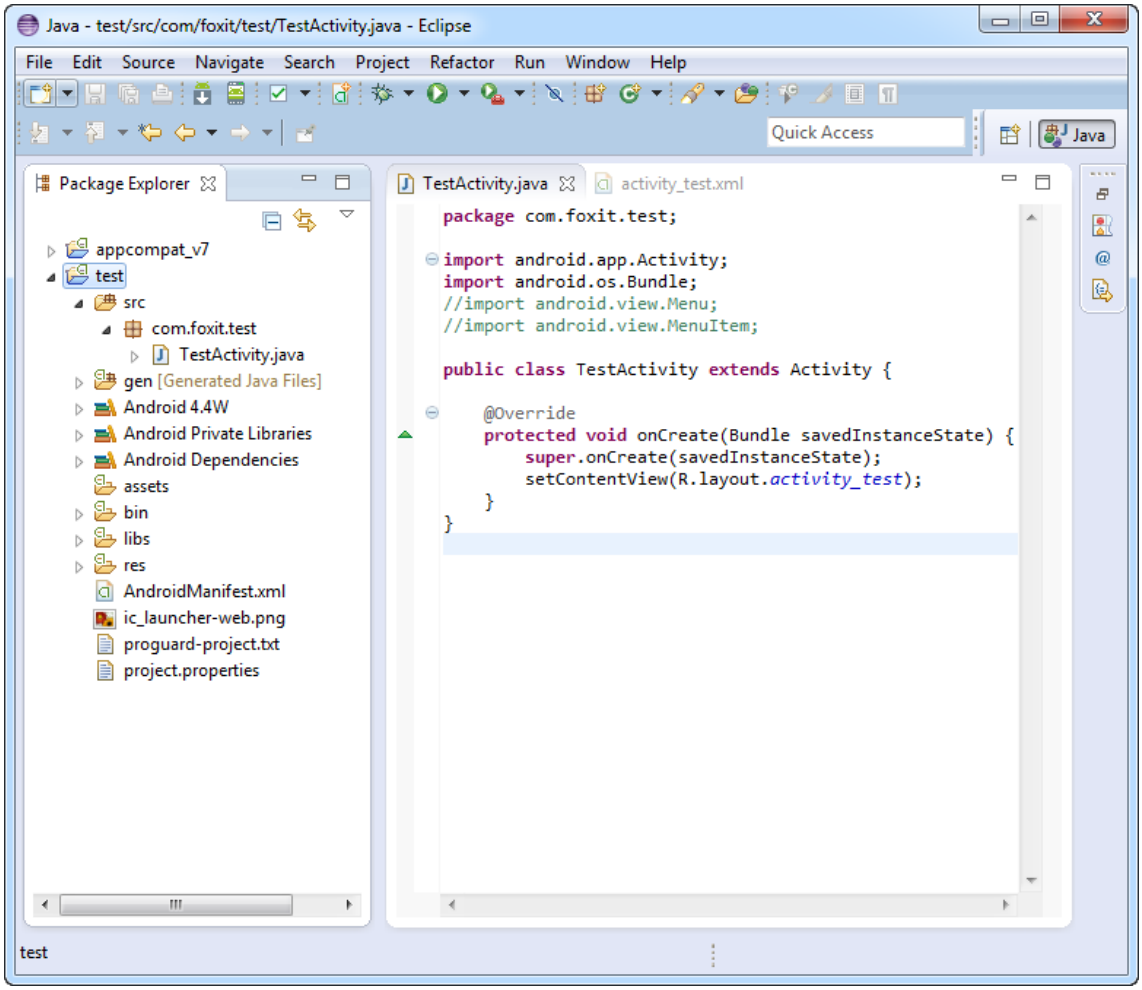

Figure 3-49

To run this project on Android using C APIs, JNI is needed to use according to the following steps.

a) Copy “include” folder from the PDF SDK package to “test” and copy the files in “libs” from the PDF SDK package to “test/libs”.

b) Declare native methods which are necessary to realize a PDF application. Create a new “.java” file called TestJNI under “test/src/com/foxit/test” folder. Open the “TestJNI.java” file, declare four native methods as shown in Figure 3-50.

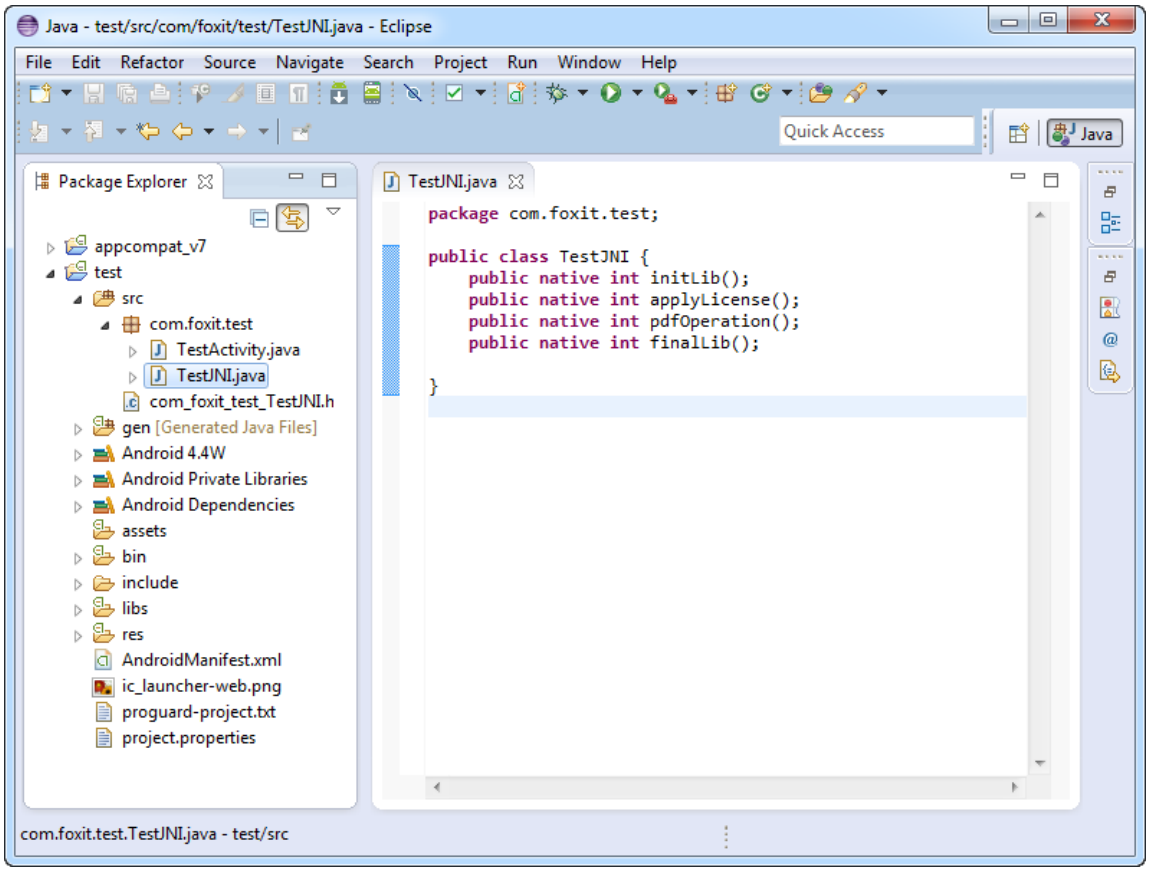

Figure 3-50

c) Generate header file.

1) First, generate “TestJNI.class” file by using “javac” in a terminal. Start “cmd.exe”, go to “test/src/com/foxit/test” and run “javac TestJNI.java”, then the “TestJNI.class” file will be generated in this folder.

2) Second, generate header file by using “javah -jni” in the terminal. Go to “test/src” and run “javah –jni com.foxit.test.TestJNI”, then the “com_foxit_test_TestJNI.h” header file will be generated in this folder. The generating process is shown in Figure 3-51. Now, delete the “TestJNI.class” file in “test/src/com/foxit/test”.

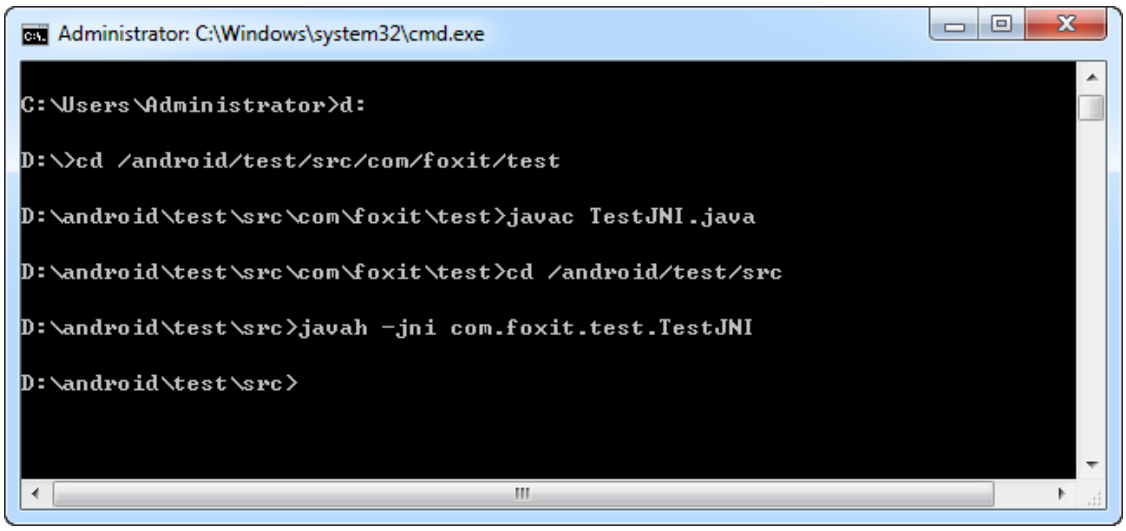

Figure 3-51

d) Create a new folder called jni under “test” folder. Copy the “com_foxit_test_TestJNI.h” header file in “test/src” to the “test/jni”, then delete it in “test/src”. “com_foxit_test_TestJNI.h” is a header file of C/C++ which corresponds with the Java interfaces defined in “TestJNI.java”. The system has completed the interface declarations automatically, which is shown in Figure 3-52.

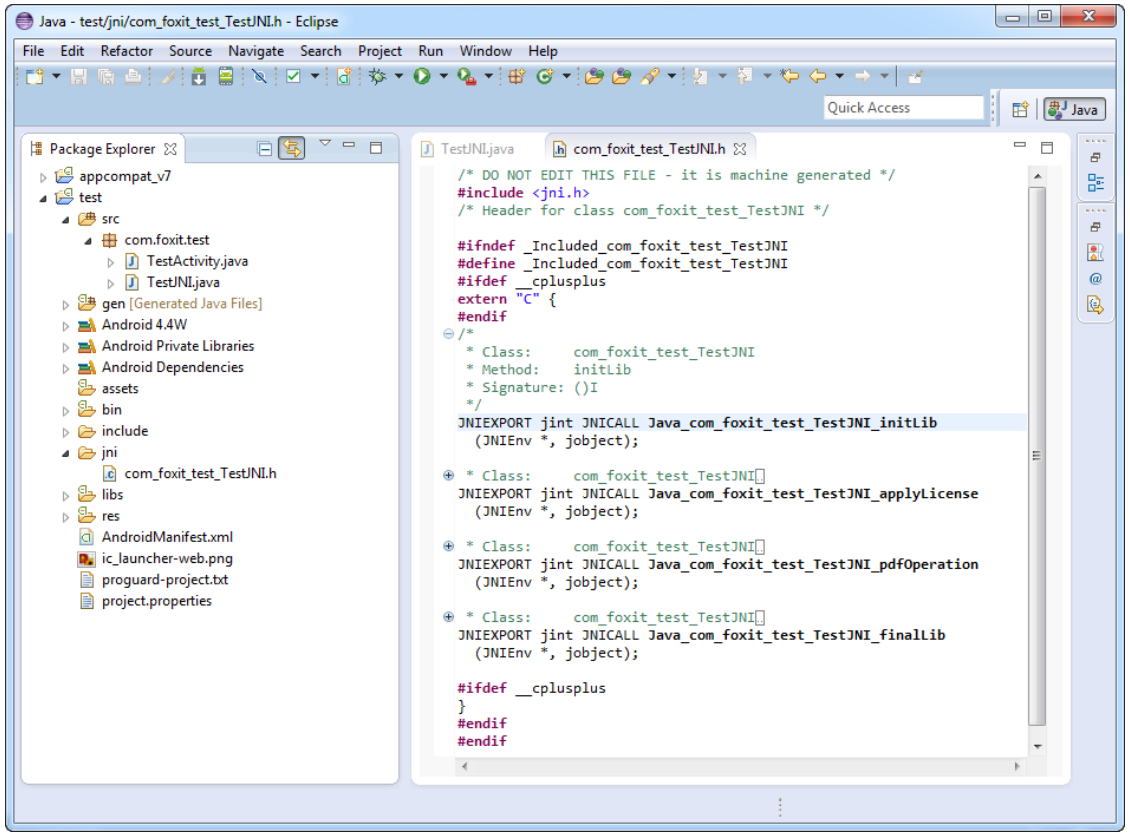

Figure 3-52

e) Create a new "com_foxit_test_TestJNI.cpp" file in "test/jni" which is used for implementing the interfaces defined in "com_foxit_test_TestJNI.h" header file. The simple implement is shown in Figure 3-53 and you can go on your application in this file. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.

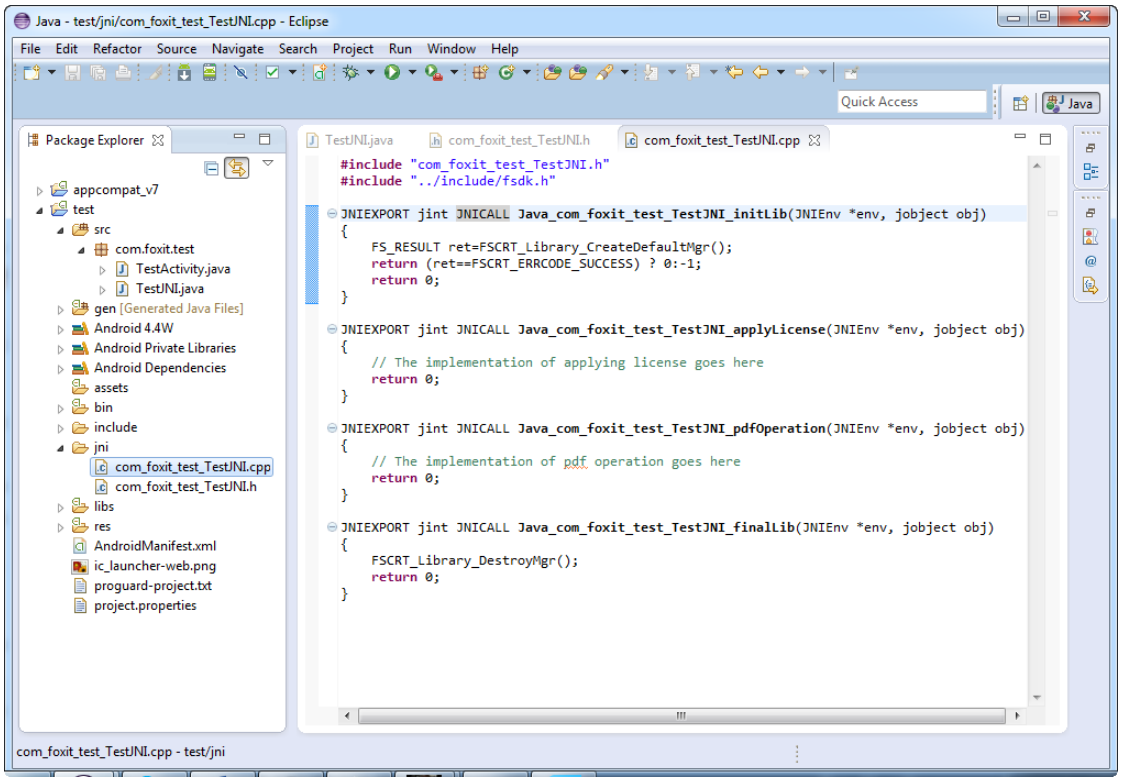

Figure 3-53

e) Create two ".mk" file called Android and Application in "test/jni", which is used for compiling and building ".so" library. Open the "Android.mk" and "Application.mk" files, input the contents referring to the Figure 3-54 and 3-55. Here, we assume that the mobile or virtual device is arm, so compile the "libfsdk_android_arm.a" library in "test/libs".

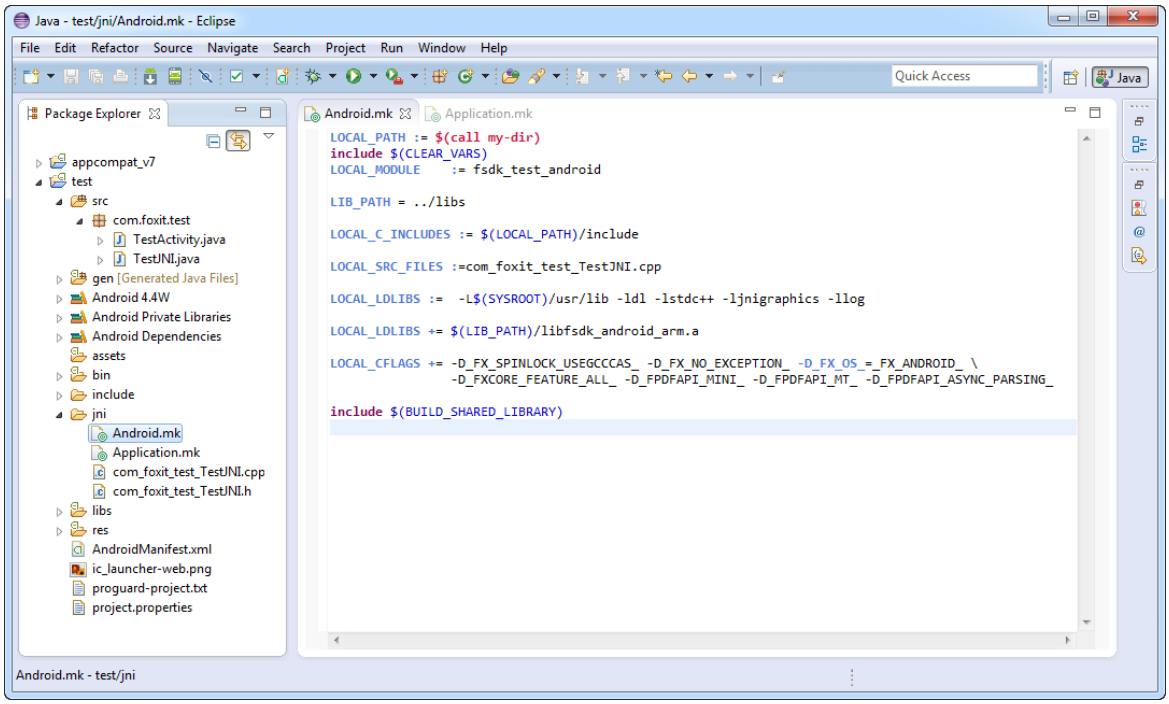

Figure 3-54

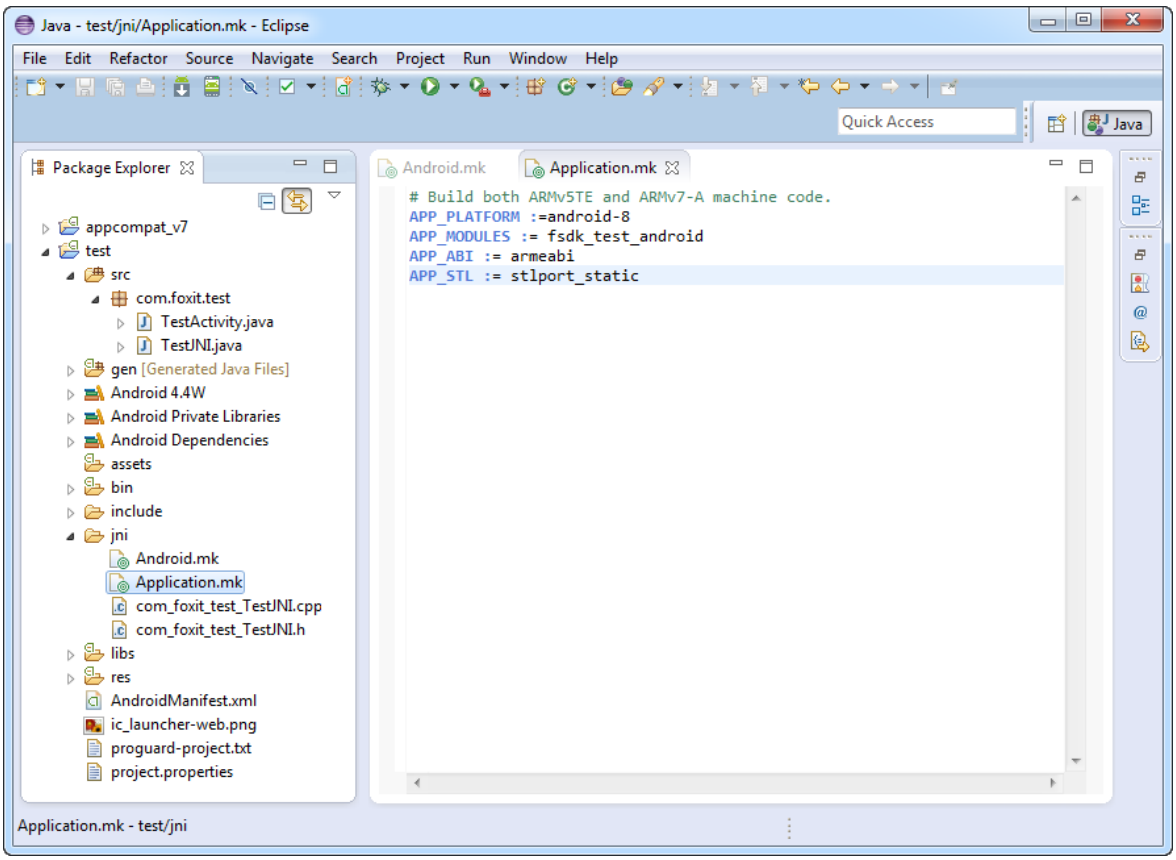

Figure 3-55

f) Open Cygwin, go to “test/jni”, run “ndk-build”, and the “libfsdk_test_android.so” library will be generated in “test/libs/armeabi”. This is shown in Figure 3-56.

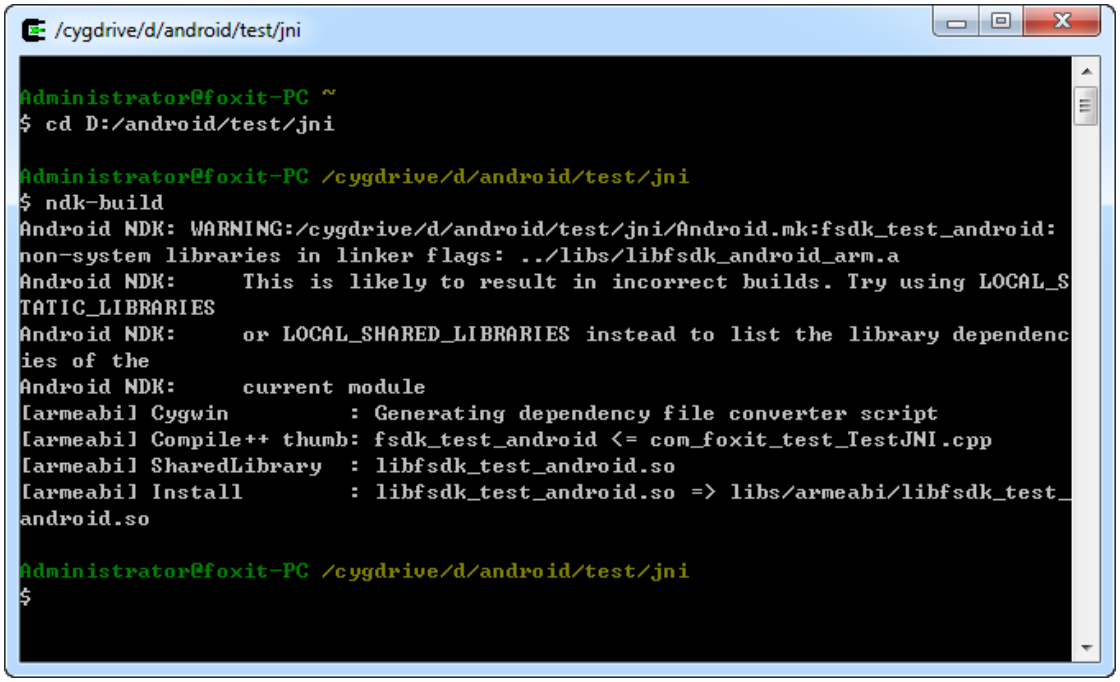

Figure 3-56

g) Call the methods in “libfsdk_test_android.so” library. Open the “TestActivity.java”, input the contents referring to Figure 3-57. First, use “System.loadLibrary("fsdk_test_Android")” to load the “.so” library that is compiled with C/C++. Then, initialize a TestJNI object to call the methods in “.so” library.

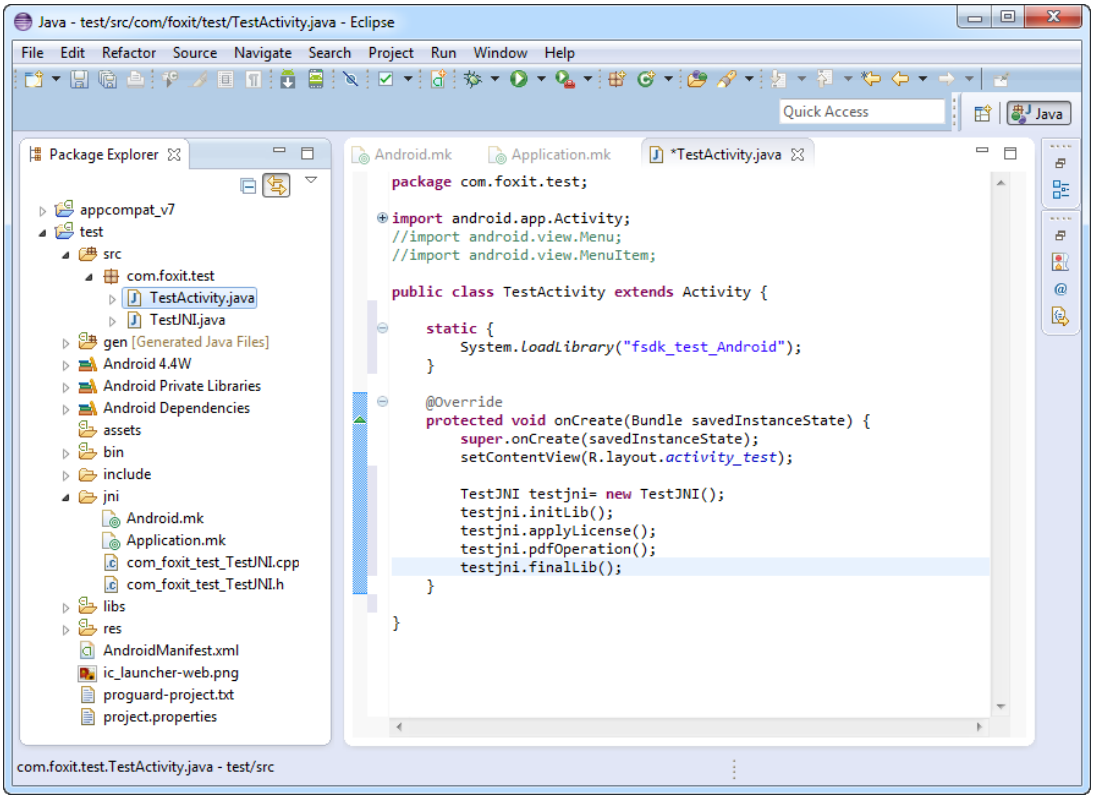

Figure 3-57

h) Click on “Run->Run as->Android Application” to run the test project. The running result is shown in Figure 3-58.

Figure 3-58

4 WORKING WITH SDK API

4.1 Common data structures and operations

The data structure and functions in Foxit PDF SDK are named by using the prefix **FSCRT_** and **FSPDF_**. FSCRT is short for **F**oxit **S**oftware **C R**un **T**ime, and FSPDF is short for **F**oxit **S**oftware **PDF**. Common data structures are listed in Table 4-1. For a complete list, please refer to fs_base_r.h and fpdf_base_r.h or API reference [2].

Table 4-1

Data Type	Usage
FSCRT_BSTR	Structure for byte string
FSCRT_FILE	Structure for file access
FSCRT_DOCUMENT	Handle to a pdf document
FSCRT_PAGE	Handle to a page
FSCRT_ANNOT	Handle to an annotation
FSCRT_PATHDATA	Handle to path data
FSCRT_FONT	Handle to font

In Foxit PDF SDK, object resources (document, page, etc.) are accessed by using handles. Memory allocation and release need to be performed properly. For example, **FSPDF_Doc_StartLoad** is used to create an FSCRT_DOCUMENT document from file. After the operation with this document, **FSPDF_Doc_Close** needs to be called to free the resources of this document. Table 4-2 lists the APIs that have to be called in pairs for memory management.

Table 4-2

Create/Initialize	Release/Clear
FSPDF_ActionData_Init	FSPDF_ActionData_Clear
FSCRT_Archive_Create	FSCRT_Archive_Release
FSCRT_Bitmap_Create	FSCRT_Bitmap_Release
FSCRT_BStr_Init	FSCRT_BStr_Clear
FSCRT_File_Create	FSCRT_File_Release
FSCRT_Font_Create	FSCRT_Font_Release
FSCRT_Image_LoadFromFile	FSCRT_Image_Release
FSCRT_Memory_Alloc	FSCRT_Memory_Free
FSCRT_PathData_Create	FSCRT_PathData_Release
FSPDF_Doc_CreateBookmarkIterator	FSPDF_Bookmark_ReleaseIterator
FSPDF_Bookmark_GetPos	FSPDF_Bookmark_ReleasePos
FSPDF_BookmarkData_Init	FSPDF_BookmarkData_Clear

FSPDF_ChoiceOption_Init	FSPDF_ChoiceOption_Clear
FSPDF_Doc_GetPage	FSPDF_Page_Clear
FSPDF_Doc_LoadAttachments	FSPDF_Attachments_Release
FSPDF_Doc_StartLoad	FSPDF_Doc_Close
FSPDF_Form_Load	FSPDF_Form_Release
FSPDF_Object_CreateBoolean FSPDF_Object_CreateInteger FSPDF_Object_CreateFloat FSPDF_Object_CreateRawByteString FSPDF_Object_CreateUnicodeString FSPDF_Object_CreateDateTime FSPDF_Object_CreateUnicodeName FSPDF_Object_CreateArray FSPDF_Object_CreateRect FSPDF_Object_CreateMatrix FSPDF_Object_CreateDict FSPDF_Object_CreateStream FSPDF_Object_CreateReference FSPDF_Object_CreateReferenceWithObjNum	FSPDF_Object_Release
FSPDF_ReflowPage_Create	FSPDF_ReflowPage_Release
FSPDF_TextPage_ExtractLinks	FSPDF_TextLink_Release
FSPDF_TextPage_Load	FSPDF_TextPage_Release
FSPDF_TextPage_SelectByRange FSPDF_TextPage_SelectByRectangle FSPDF_TextSearch_GetSelection FSPDF_TextLink_GetSelection	FSPDF_TextSelection_Release
FSPDF_TextObject_Create FSPDF_PathObject_Create FSPDF_ImageObject_Create	FSPDF_PageObject_Release
FSPDF_TextPage_StartSearch	FSPDF_TextSearch_Release
FSPDF_Watermark_CreateFromPage FSPDF_Watermark_CreateFromImage FSPDF_Watermark_CreateFromBitmap FSPDF_Watermark_CreateFromText	FSPDF_Watermark_Release
FSPDF_Page_LoadAnnots	FSPDF_Page_UnloadAnnots
FSPDF_LayerNode_Init	FSPDF_LayerNode_Clear
FSPDF_LayerContext_Create	FSPDF_LayerContext_Release

4.2 File

PDF file access (I/O) is managed by file structure FSCRT_FILE and file handler FSCRT_FILEHANDLER. Developers can determine whether to implement reading actions or writing actions in the FSCRT_FILEHANDLER handle based on application intentions, but please note that the reading actions and writing actions cannot be done at the same time. Foxit PDF SDK provides the capability of reading file stream from a file or memory and the flexibility on all platforms that are introduced in this document. An example of reading a file on windows platform is shown below.

```
...

FSCRT_BSTRC(filename, " FoxitText.pdf");
FSCRT_FILE file = NULL;
FS_RESULT ret = FSCRT_File_CreateFromFileName(&filename, FSCRT_FILEMODE_READONLY, &file);

if (FSCRT_ERRCODE_SUCCESS != ret)
return NULL;

FSCRT_DOCUMENT Doc = NULL;
ret = FSPDF_Doc_StartLoad(file, NULL, &Doc, NULL);
if (FSCRT_ERRCODE_SUCCESS != ret)
return NULL;

...
```

4.3 Document

PDF document is represented by FSCR_DOCUMENT handle object, which is used for document level operation such as opening and closing files, getting page, annotation, metadata and etc. An FSCRT_DOCUMENT handle should be initialized by calling **FSPDF_Doc_StartLoad** to allow page or deeper level API to work. Some common APIs at document level are listed in Table 4-3. For a complete list, please refer to fpdf_document_r.h and fpdf_document_w.h or API reference [2]. An example shows how to work with PDF document.

Table 4-3

API Name	Description
FSPDF_Doc_StartLoad	Load a PDF document from file
FSPDF_Doc_Close	Close a PDF document
FSPDF_Doc_GetEncryptionType	Get encryption type
FSPDF_Doc_GetPage	Get a handle to page object
FSPDF_Doc_GetAction	Get PDF document trigger action
FSPDF_Metadata_GetString	Get PDF metadata corresponding to a specified key
FSPDF_Bookmark_GetAction	Get PDF bookmark action
FSPDF_ViewerPref_GetUIVisibility	Get UI visibility status from viewer preferences
FSPDF_Attachments_GetAttachment	Get a specific attachment from PDF document
FSPDF_Doc_Create	Create a new PDF document object
FSPDF_Doc_StartImportPages	Import pages from a source document
FSPDF_Doc_StartSaveToFile	Start saving a PDF document to a file in a progressive manner
FSPDF_Doc_SetAction	Set document trigger action
FSPDF_Bookmark_Insert	Insert a new bookmark and set the data

Example: load PDF document and get the first page handle object

```
#include "fpdf_document_r.h"
#include "fpdf_document_w.h"

...

//Assuming a FSCRT_FILE file has been created.
FSCRT_DOCUMENT pdfDoc = NULL;
if (FSCRT_ERRCODE_SUCCESS == FSPDF_Doc_StartLoad(file, NULL, &pdfDoc))
{
	FSCRT_PAGE pdfPage = NULL;
	if (FSCRT_ERRCODE_SUCCESS == FSPDF_Doc_GetPage(pdfDoc, 0, &pdfPage))
	{
		...
	}
	FSPDF_Page_Clear(pdfPage);
	pdfPage = NULL;
}
FSPDF_Doc_Close(pdfDoc);
pdfDoc = NULL;
...
```

4.4 Attachment

In Foxit PDF SDK, attachments are only referred to attachments of documents rather than file attachment annotation. PDF SDK provides applications APIs to access attachments such as loading attachments, getting attachments, inserting attachments and accessing properties of attachments. Some common APIs are listed in Table 4-4. For a complete list, please refer to fpdf_document_r.h and fpdf_document_w.h or API reference [2]. An example shows how to insert an attachment file into a PDF.

Table 4-4

API Name	Description
FSPDF_Doc_LoadAttachments	Load all attachments of PDF document
FSPDF_Attachments_Release	Release a attachments object
FSPDF_Attachments_CountAttachment	Get the count of attachments
FSPDF_Attachments_GetAttachment	Get a specific attachment
FSPDF_Attachments_InsertAttachment	Insert an attachment
FSPDF_Attachment_GetFileName	Get file name of an attachment
FSPDF_Attachment_SetFile	Set the file of an attachment

Example: insert an attachment file into a pdf

```
#include "fpdf_document_r.h"
#include "fpdf_document_w.h"

//Assuming FSCRT_DOCUMENT pdfDoc has been loaded.
```

```
//Assuming returning values will be checked in active source code.
...
...
FSPDF_ATTACHMENTS     attachs;
FSPDF_ATTACHMENT      attach;
FS_RESULT ret = FSPDF_Doc_LoadAttachments(pdfDoc, &attachs);
...
ret = FSPDF_Attachment_Create(pdfdoc,&attach);
...
ret = FSPDF_Attachments_CountAttachment(attachs, &count);
...
ret = FSPDF_Attachments_InsertAttachment(attachs, count, attach);
...
CFSCRT_File * file = new CFSCRT_File();
//Assuming file path is ready.
FS_BOOL bRet = file->Load(filePath);
ret = FSCRT_File_Create(file, &setFile);

//associating a file to the given attachment.
ret = FSPDF_Attachment_SetFile(attach, setFile);
...
```

4.5 Page

PDF page is represented by FSCRT_PAGE handle object. Page level APIs provide functions to parse, render, read and set the properties of a page. FSCRT_PAGE object is created by **FSPDF_Doc_GetPage** and needs to be cleared by **FSPDF_Page_Clear**. A PDF page needs to be parsed before it is rendered or processed for text extraction. Some common APIs at page level are listed in Table 4-5. For a complete list, please refer to fpdf_page_r.h and fpdf_page_w.h or API reference [2]. Two examples show how to work with PDF page.

Table 4-5

API Name	Description
FSPDF_Page_StartParse	Start parsing a PDF page
FSPDF_Page_GetIndex	Get page index
FSPDF_Page_Clear	Release all page contents and relative resources
FSPDF_Page_GetSize	Get page size
FSPDF_Page_GetMatrix	Get page transformation matrix
FSPDF_RenderContext_StartPage	Start rendering a PDF page in a renderer with a PDF rendering context
FSPDF_Page_SetIndex	Change page index of a PDF page
FSPDF_Page_SetAction	Set a page trigger action
FSPDF_Page_Create	Create a new page

Example 1: get page size

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...
// Get the given page's size
FS_FLOAT width = 0, height = 0;
ret = FSPDF_Page_GetSize(page, &width, &height);
...
```

Example 2: create a page and set the size

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...
// Create a new page and set the page size
FSCRT_PAGE page = NULL;
if (FSCRT_ERRCODE_SUCCESS == FSPDF_Page_Create(document, 0, &page))
{
	ret = FSPDF_Page_SetSize(page, PageWidth, PageHeight);
	...
}
FSPDF_Page_Clear(page);
...
```

4.6 Render

PDF rendering is realized through the Foxit renderer, a graphic engine that is created on a bitmap or platform graphics device. Rendering process requires a renderer and render context. APIs to create renderers on bitmap, windows DC, and Mac OS are **FSCRT_Renderer_CreateOnBitmap**, **FSCRT_Renderer_CreateOnWindowsDC** and **FSCRT_Renderer_CreateOnCGContext** correspondingly. The rendering settings (or render context) are set in FSPDF_RENDERCONTEXT object. Once the renderer and render context are in place, the rendering process of a PDF page can be started using **FSPDF_RenderContext_StartPage**. Some common APIs for rendering are listed in Table 4-6. For a complete list, please refer to fs_renderer_r.h, fs_renderer_windows_r.h, fs_renderer_apple_r.h, fs_psi_w.h, fpdf_reflow_r.h and fpdf_base_r.h or API reference [2]. Two examples are shown below, one is rendering a bitmap and the other is printing a PDF document in windows system.

Table 4-6

API Name	Description
FSCRT_Renderer_CreateOnBitmap	Create a renderer on a bitmap object
FSCRT_Renderer_CreateOnWindowsDC	Create a renderer on a Windows device context
FSCRT_Renderer_CreateOnCGContext	Create a renderer on an apple quartz context
FSCRT_Renderer_Release	Release a given renderer object
FSCRT_Renderer_SetFlags	Set flags of a renderer
FSCRT_Renderer_DrawBitmap	Render a bitmap object
FSCRT_PSI_Render	Render a pressure sensitive ink object
FSPDF_RenderContext_StartAnnots	Render annotations on render context

FSPDF_RenderContext_StartPageAnnots	Render all annotations of a page on render context
FSPDF_RenderContext_StartFormControls	Render a PDF form control
FSPDF_RenderContext_StartPage	Start to render a PDF page
FSPDF_RenderContext_StartReflowPage	Start to render a reflow page

Example 1: render a page to a bitmap

```
#include "fpdf_renderer_r.h"
...

//Create a render context and render a page to get a bmp later
FSPDF_RENDERCONTEXT context;
FS_RESULT ret = FSPDF_RenderContext_Create(&context);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;
... //Assuming an FSCRT_BITMAP bitmap is created here
//Create a renderer on the given bitmap
FSCRT_RENDERER renderer;
ret = FSCRT_Renderer_CreateOnBitmap(bitmap, &renderer);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;

... // Get the FSCRT_MATRIX matrix (page transformation matrix) here

//Set the matrix of the given render context
ret = FSPDF_RenderContext_SetMatrix(context, &matrix);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;

//Start to render with the given render context, renderer and page to get the render progress
FSCRT_PROGRESS renderProgress = NULL;
ret = FSPDF_RenderContext_StartPage(context, renderer, page,FSPDF_PAGERENDERFLAG_NORMAL,
&renderProgress);

... //Continue progressive progress

//Release render progress
FSCRT_Progress_Release(renderProgress);
//Release render
FSCRT_Renderer_Release(renderer);
//Release render context
FSPDF_RenderContext_Release(context);
...
```

Example 2: print a page in windows system

```
#include "fpdf_renderer_r.h"
#include "fs_renderer_windows_r.h"

... //Assuming an HDC hdcPrinter (a printer DC handles) has been prepared.

//Create a renderer on the given Windows device context
FSCRT_RENDERER renderer;
ret = FSCRT_Renderer_CreateOnWindowsDC(hdfPrinter, &renderer);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;
```

```
FSPDF_RENDERCONTEXT context;
FS_RESULT ret = FSPDF_RenderContext_Create(&context);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;

FSPDF_RenderContext_SetMatrix(context, &matrix);

//Start to render with the given render context, renderer and page to get the render progress
FSCRT_PROGRESS renderProgress = NULL;
ret = FSPDF_RenderContext_StartPage(context, renderer, page,FSPDF_PAGERENDERFLAG_NORMAL,
&renderProgress);

... //Continue progressive progress

//Release render progress
FSCRT_Progress_Release(renderProgress);
//Release render
FSCRT_Renderer_Release(renderer);
//Release render context
FSPDF_RenderContext_Release(context);
...
```

4.7 Text

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. Prior to text processing, user should first call **FSPDF_TextPage_Load** to get the textPage object (if page object is not available, user should call **FSPDF_Doc_GetPage** to get the page object). Some common APIs for text processing are listed in Table 4-7. For a complete list, please refer to fpdf_textpage_r.h or API reference [2]. An example shows how to do text search.

Table 4-7

API Name	Description
FSPDF_TextPage_Load	Prepare information about all characters in a page
FSPDF_TextPage_Release	Release all resources allocated for a PDF text page handle
FSPDF_TextPage_GetChars	Get text content in a page, within a specific character range
FSPDF_TextPage_ExportToFile	Export text content in a page to a specific file handle
FSPDF_TextPage_SelectByRange	Get a text selection handle by specific character range
FSPDF_TextSelection_GetChars	Extract the whole text from a PDF text selected area
FSPDF_TextPage_StartSearch	Start a PDF text search process
FSPDF_TextSearch_FindPrev	Search in the direction from page end to start
FSPDF_TextLink_CountLinks	Get count of the URL formatted texts inside a page

Example: search a text pattern in a page

```
#include "fpdf_textpage_r.h"
#include "fpdf_textpage_w.h"
```

```
//searchPattern is an FSCRT_BSTR struct
FSPDF_TEXTPAGE textPage = NULL;
ret = FSPDF_TextPage_Load(pdfPage, &textPage); //Load text page
if (FSCRT_ERRCODE_SUCCESS == ret)
{
	FSPDF_TEXTSEARCH textSearch = NULL;
	ret = FSPDF_TextPage_StartSearch(textPage, &searchPattern, 0, 0, &textSearch);
	if (FSCRT_ERRCODE_SUCCESS == ret)
	{
		FS_BOOL isMatch = TRUE;
		FSPDF_TextSearch_FindNext(textSearch, &isMatch);
		...
	}
}
```

4.8 Form

Foxit PDF SDK provides APIs to view and edit form field programmatically. Form fields are commonly used in PDF documents to gather data. Prior to using the functions in Form module, user should call the interfaces **FSPDF_Form_Load, FSPDF_Doc_InitiateJavaScript** and **FSPDF_Page_LoadAnnots** in turn. **FSPDF_Form_ExportToFDFDoc** can export data in a PDF document to an FDF (Forms Data Format) document, from where data can be extracted for further use. **FSPDF_Form_ImportFromFDFDoc** allows user to import FDF documents to the original PDF so the form data could be displayed. Some common APIs for form processing are listed in Table 4-8. For a complete list, please refer to fpdf_form_r.h, fpdf_form_w.h or API reference [2]. An example shows how to count form fields and get the properties.

Table 4-8

API Name	Description
FSPDF_Form_Load	Retrieve a form handle for specific document
FSPDF_Form_Release	Release the resources of a form handle
FSPDF_Form_GetField	Search and retrieve the name and type of a field satisfying a name filter in a form
FSPDF_FormField_GetAction	Retrieve action associated with a field and a trigger type at a specified index in a form
FSPDF_Form_ExportToFDFDoc	Export data in a form to a FDF document
FSPDF_Form_SetDefaultAppearance	Set default appearance of a form
FSPDF_FormFiller_SetHighlightColor	Set the highlight color for the form field
FSPDF_FormField_InsertAction	Insert an action associated with a field and a trigger type at a specified index in a form

Example: count form fields and get the properties

```
#include "fpdf_form_r.h"
#include "fpdf_form_w.h"
... //Assuming FSCRT_DOCUMENT pdfDoc has been loaded.
```

```
FS_INT32 nFieldCount = 0;
FS_INT32 nType = 0;
FS_INT32 nAliment = 0;
FSCRT_BSTR strName;
FSCRT_BStr_Init(&strName);

FSPDF_FORM pdfForm = NULL;
ret = FSPDF_Form_Load(pdfDoc, &pdfForm);
FSPDF_Form_CountFields(pdfForm, NULL, &nFieldCount);
for (int i = 0; i < nFieldCount; i++)
{
	ret = FSPDF_Form_GetField(pdfForm, NULL, i, &strName, &nType);
	if (FSCRT_ERRCODE_SUCCESS == ret)
	{
		if (FSPDF_FORMFIELDTYPE_CHECKBOX == nType)
		{
			...
		}
		ret = FSPDF_FormField_GetAlignment(pdfForm, &strName, &nAliment);
		...
	}
}
```

4.9 Form Filler

Form filler is the most commonly used feature for users. Form filler provides applications to fill forms dynamically. The key point for applications to fill forms is to construct some callback functions for PDF SDK to call. Those functions are defined in FSPDF_ACTION_HANDLER **and** **FSPDF_FORMINTERACTION_WINDOWLESS** structure. The details and the related APIs can be found in fpdf_document_r.h and fpdf_form_w.h. Table 4-9 lists some APIs provided by the functionality of form filler.

Table 4-9

API Name	Description
FSPDF_Doc_SetActionHandler	Set the action handler to the PDF document to execute form actions.
FSPDF_FormFiller_Begin	Begin the form filling.
FSPDF_FormFiller_TriggerWindowlessEvent	Trigger the platform event for the windowless form filling.
FSPDF_FormFiller_SetHighlightColor	Set the highlight color for the form field.
FSPDF_FormFiller_ShowHighlight	Whether to show the highlight of form field or not.
FSPDF_FormFiller_End	Finish the form filling.

Example: fill a form with form filler.

```
#include "fpdf_form_r.h"
#include "fpdf_form_w.h"
//Assuming FSCRT_DOCUMENT pdfDoc has been loaded.
//Assuming return value will be checked in active source code.

FSPDF_FORM pdfForm = NULL;
FSPDF_FORMINTERACTION_WINDOWLESS windowless;
//Assuming all callback functions in windowless are ready in for use
FSPDF_ACTION_HANDLER actionHandler = NULL;
//Assuming all callback functions in action handler are ready in for use

FSPDF_Doc_SetActionHandler(pdfDoc, &actionHandler);

FS_RESULT ret = FSPDF_Form_Load(pdfDoc, &pdfForm);
...
ret = FSPDF_FormFiller_Begin(pdfForm, &windowLess,  &formFiller);
...
ret = FSPDF_FormFiller_TriggerWindowlessEvent(formFiller, page, &device,
FSCRT_EVENT_MBUTTONDBLCLK, &mouseData);
...
ret = FSPDF_FormFiller_End(formFiller);
...
```

4.10 Annotations

4.10.1 General

An annotation associates an object such as note, line, and highlight with a location on a page of a PDF document. It provides a way to interact with users by means of the mouse and keyboard. PDF includes a wide variety of standard annotation types as listed in Table 4-10. Among these annotation types, many of them are defined as markup annotations for they are used primarily to mark up PDF documents. These annotations have text that appears as part of the annotation and may be displayed in other ways by a conforming reader, such as in a Comments pane. The 'Markup' column in Table 4-10 shows whether an annotation is a markup annotation.

Foxit PDF SDK supports most annotation types defined in PDF reference [1]. PDF SDK provides APIs of annotation creation, properties access and modification, appearance setting and drawing. Some common APIs are listed in Table 4-11. For a complete list, please refer to fpdf_objets_r.h and fpdf_objets_w.h or API reference [2].

Table 4-10

Annotation type	Description	Markup	Supported by SDK
Text(Note)	Text annotation	Yes	Yes
Link	Link Annotation	No	Yes

FreeText(TypeWritter)	Free text annotation	Yes	Yes
Line	Line annotation	Yes	Yes
Square	Square annotation	Yes	Yes
Circle	Circle annotation	Yes	Yes
Polygon	Polygon annotation	Yes	Yes
PolyLine	PolyLine annotation	Yes	Yes
Highlight	Highlight annotation	Yes	Yes
Underline	Underline annotation	Yes	Yes
Squiggly	Squiggly annotation	Yes	Yes
StrikeOut	StrikeOut annotation	Yes	Yes
Stamp	Stamp annotation	Yes	Yes
Caret	Caret annotation	Yes	Yes
Ink(pencil)	Ink annotation	Yes	Yes
Popup	Popup annotation	Yes	Yes
File Attachment	FileAttachment annotation	Yes	Yes
Sound	Sound annotation	Yes	No
Movie	Movie annotation	No	No
Widget*	Widget annotation	No	Yes
Screen	Screen annotation	Yes	No
PrinterMark	PrinterMark annotation	No	No
TrapNet	Trap network annotation	No	No
Watermark*	Watermark annotation	Yes	Yes
3D	3D annotation	Yes	No

Note:

1. The annotation types of widget and watermark are special. They aren't supported in the module of 'Annotation'. The type of widget is only used in the module of 'form filler' and the type of watermark only in the module of 'watermark'.

2. Foxit SDK supports a customized annotation type called PSI (pressure sensitive ink) annotation that is not described in PDF ISO standard [1]. Usually, PSI is for handwriting features and Foxit SDK treats it as PSI annotation so that it can be handled by other PDF products.

Table 4-11

API Name	Description
FSPDF_Annot_GetCount	Get count of annotations by a specific filter
FSPDF_Annot_Get	Get a specified annotation by a specific filter
FSPDF_Annot_GetIndex	Get index of an annotation filtered by a string
FSPDF_Annot_GetName	Get an annotation's name.

FSPDF_Annot_Add	Add an annotation by a given index and a string filter
FSPDF_Annot_Remove	Remove an annotation from a given page
FSPDF_Annot_Move	Move an annotation to a new position specified by a rectangle
FSPDF_Annot_SetFlags	Set flags of an annotation
FSPDF_Annot_SetName	Set name of an annotation

Example: add a highlight annotation to a page and set the related annotation properties

```
#include "fpdf_annot_r.h"
#include "fpdf_annot_w.h"
...

//The function of load Annots shall be called before any operations on annotations
FS_RESULT ret = FSPDF_Page_LoadAnnots(pdfPage);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;

//Prepare the rectangle object of annotation bounding box, in PDF page coordination.
FSCRT_RECTF rect = {0, 100, 100, 0};
//Prepare the string object of the annotation filter.
FSCRT_BSTR bsAnnotType;
FSCRT_BStr_Init(&bsAnnotType);
FSCRT_BStr_Set(&bsAnnotType, "Highlight", 9);

//Add an annotation to a specific index with specific filter.
FSCRT_ANNOT annot = NULL;
ret = FSPDF_Annot_Add(pdfPage, &rect, &bsAnnotType, &bsAnnotType, 1, &annot);
if (FSCRT_ERRCODE_SUCCESS != ret)
{
	...
}
//Set the quadrilaterals points of annotation.
FSCRT_QUADPOINTSF quadPoints = {0,0,100,0,0,50,100,50};
FSPDF_Annot_SetQuadPoints(annot, &quadPoints, 1);
//Set the stroke color and opacity of annotation.
FSPDF_Annot_SetColor(annot, FALSE, 0x0000FF00);
FSPDF_Annot_SetOpacity(annot, (FS_FLOAT)0.55);
...
```

4.10.2 Import annotations from or export annotations to a FDF file

In Foxit PDF SDK, annotations can be created with data not only from applications but also from FDF files. At the same time, PDF SDK supports to export annotations to FDF files. The APIs for importing or exporting annotations can be found in Table 4-12. For a complete list, please refer to ffdf_document_r.h and ffdf_doument_w.h.

Table 4-12

API Name	Description
FSFDF_Doc_Load	Load a FDF document.
FSFDF_Doc_Close	Close a FDF document.
FSFDF_Doc_GetPDFPath	Get the path of PDF document from FDF document in UTF-8 string.
FSFDF_Annot_GetCount	Count annotations in a FDF document by specific filter.
FSFDF_Annot_Get	Get an annotation from FDF document by specific filter and index.
FSFDF_Doc_Create	Create a new FDF document.
FSFDF_Doc_Save	Save a FDF document.
FSFDF_Doc_SetPDFPath	Set the path of PDF document to FDF document. It's a UTF-8 string.
FSPDF_Annot_ExportToFDFDoc	Export a PDF annotation into a FDF document.
FSFDF_Annot_ExportToPDFPage	Export an annotation object loaded from a FDF document into a PDF page.

Example: load annotations from a FDF file and add them into the first page of a given PDF

```
#include "ffdf_document_r.h "
#include "ffdf_doument_w.h"
...
FSCRT_ANNOT fdfAnnot = NULL;

//Assume that fsFile is ready for opening
FS_RESULT ret = FSFDF_Doc_Load(fsFile, &fdfDoc);

FSCRT_BStr_Init(&filter);
FSCRT_BStr_Set(&filter, "Text", strlen("Text"));

FS_INT32 FS_count = 0;
ret = FSFDF_Annot_GetCount(fdfDoc, &filter, &count);

//Assume that fsPDFFile is ready for opening
FS_RESULT ret = FSFDF_Doc_Load(fsPDFFile, &pdfDoc);
...
FSCRT_PAGE page;
ret = FSPDF_Doc_GetPage(pdfDoc, 0, &page);
...
for(FS_INT32 i=0; i<count; i++)
{
	ret = FSFDF_Annot_Get(fdfDoc, &filter, i, &fdfAnnot);
	...
	//Export annotations retrieved from FDF files to a given PDF page
	FSCRT_ANNOT pdfAnnot = NULL;
	ret = FSFDF_Annot_ExportToPDFPage(fdfAnnot, page, &pdfAnnot);
	...
}
```

```
//Assume all resources will be released here.
...
```

4.11 Image conversion

Foxit PDF SDK provides APIs for conversion between PDF files and images. Applications could easily fulfill functionality like image creation, conversion, input and output operations. Some common APIs are listed in Table 4-13. For a complete list, please refer to fs_image_r.h and fs_image_w.h or API reference [2]. Here shows an example of converting PDF pages to bitmap files.

Table 4-13

API Name	Description
FSCRT_Bitmap_Create	Create a bitmap.
FSCRT_Image_LoadFromFile	Load image from image file.
FSCRT_Image_CountFrames	Count frames of an image.
FSCRT_Image_LoadFrame	Load image frame by index.
FSCRT_Image_GetCurrentFrameBitmap	Retrieve bitmap of current frame.
FSCRT_ImageFile_Create	Create an image file.
FSCRT_ImageFile_AddFrame	Add a frame to image file.

Example: Convert PDF pages to bitmap files.

```
#include "fpdf_image_r.h"
#include "fpdf_image_w.h"
...
//if file and password are ready for use
FS_RESULT ret = FSPDF_Doc_StartLoad(file, password, pdfDoc, NULL);
...
ret = FSPDF_Doc_CountPages(autoPdfDoc.GetDocument(), &nPageCount);
...
for (FS_INT32 i=0; i< nPageCount; i++)
{
	ret = FSPDF_Doc_GetPage(pdfDoc, i, &m_pdfPage);
	//assume pages are parsed before they are handled further.
	...
	//assume width and height are known here.
	FSCRT_BITMAP  bitmap;
	ret = FSCRT_Bitmap_Create((FS_INT32)width, (FS_INT32)height,
FSCRT_BITMAPFORMAT_24BPP_RGB, NULL, 0, &bitmap);
	...
	FSCRT_RENDERER renderer;
	ret = FSCRT_Renderer_CreateOnBitmap(bitmap, &renderer);
	...
	FSPDF_RENDERCONTEXT rendercontext;
	ret = FSPDF_RenderContext_Create(&rendercontext);
```

```
        //create a rendering process.
        FSCRT_PROGRESS renderProgress = NULL;
        ret = FSPDF_RenderContext_StartPage(rendercontext, renderer,
page,FSPDF_PAGERENDERFLAG_NORMAL, &renderProgress);
        ...
        //continue to render the current page if rendering process isn't finished.
        ret = FSCRT_Progress_Continue(renderProgress, NULL);
        ...

        //save a bitmap to an image files here.
        FSCRT_BSTRC(filePath, "./output.bmp");
        FSCRT_FILE file = NULL;
        FSCRT_File_CreateFromFileName(&filePath, FSCRT_FILEMODE_WRITE, &file);
        FSCRT_IMAGEFILE imageFile = NULL;
        FSCRT_ImageFile_Create(file, FSCRT_IMAGETYPE_BMP, 1, &imageFile);
        FSCRT_ImageFile_AddFrame(imageFile, bitmap);
        FSCRT_ImageFile_Release(imageFile);
        ...
}
...
```

4.12 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. Some common APIs are listed in Table 4-14. For a complete list, please refer to fpdf_security_r.h and fpdf_security_w.h or API reference [2].

Table 4-14

API Name	Description
FSPDF_Security_RegisterHandler	Register a custom security handler to Foxit PDF SDK, enabling access to a PDF document which is protected by customized security handler
FSPDF_Security_UnregisterHandler	Unregister a customer security handler to Foxit PDF SDK
FSPDF_Security_SetCertificateHandler	Set certificate security handler to Foxit PDF SDK
FSPDF_Security_CheckPassword	Detect type of password
FSPDF_Security_StartCustomEncryption	Start custom PDF Encryption, and the PDF document will be protected by non-standard security handler

Example: encrypt a PDF file with user password "123" and owner password "456"

```
#include "fpdf_security_r.h"
#include "fpdf_security_w.h"
...

FSCRT_BSTRC(userPwd,"123");
FSCRT_BSTRC(ownerPwd,"456");
FSCRT_PROGRESS encryptProgress = NULL;
```

```
ret = FSPDF_Security_StartPasswordEncryption(pdfDoc, permissions, &userPwd, &ownerPwd,
FSCRT_CIPHER_AES, 16, TRUE, encryptedFile, &encryptProgress);

if (ret == FSCRT_ERRCODE_SUCCESS)
{
	FSCRT_Progress_Continue(encryptProgress, NULL);
	FSCRT_Progress_Release(encryptProgress);
}
...
```

4.13 Watermark

Watermark is a type of PDF annotation and is widely used in PDF document. Watermark is a visible embedded overlay on a document consisting of text, a logo, or a copyright notice. The purpose of a watermark is to identify the work and discourage its unauthorized use. Foxit PDF SDK provides APIs to work with watermark, allowing applications to create, insert, release and remove watermarks. Some common APIs are listed in Table 4-15. For a complete list, please refer to fpdf_watermark_w.h or API reference [2].

Table 4-15

API Name	Description
FSPDF_Watermark_CreateFromText	Create a text watermark
FSPDF_Watermark_CreateFromBitmap	Create a bitmap watermark
FSPDF_Watermark_GetSize	Retrieve the size (width and height) of a specific watermark.
FSPDF_Watermark_InsertToPage	Insert a watermark to a specific page
FSPDF_Watermark_Release	Release a watermark object
FSPDF_Page_RemoveWatermarks	Remove all watermarks from a given page

Example: create a text watermark and insert it into the first page.

```
#include "fpdf_watermark_w.h"
...

FSPDF_WATERMARK_TEXTPROPERTIES textproperties;
textproperties.font = NULL;
FS_RESULT ret = FSCRT_Font_CreateStandard(FSCRT_STDFONT_HELVETICA, &textproperties.font);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;

textproperties.fontSize = 100.0f;
textproperties.color = FSCRT_ARGB_Encode(0xff, 0xff, 0x00, 0x00);
textproperties.fontStyle = FSPDF_WATERMARK_FONTSTYLE_UNDERLINE;
textproperties.lineSpace = 1;
textproperties.alignment = FSPDF_WATERMARK_TEXTALIGNMENT_CENTER;

FSCRT_PAGE page = NULL;
FSPDF_WATERMARK watermark = NULL;
FSCRT_PROGRESS parseProgress = NULL;
```

```
FSPDF_WATERMARK_SETTINGS settings = {FSPDF_WATERMARKPOS_CENTER,. 0, 0,
FSPDF_WATERMARKFLAG_ONTOP, 0.25f, 0.25f, 0, 100};

//Init insert string.
FSCRT_BSTRC(unicodeString, "Hello!");
//Create watermark from text.
ret = FSPDF_Watermark_CreateFromText(doc, &unicodeString, &textproperties, &settings,
&watermark);

... //Parse a given page

FSPDF_Watermark_InsertToPage(watermark, page);

...
```

4.14 Barcode

A barcode is an optical machine-readable representation of data relating to the object to which it is attached. Originally barcodes systematically represented data by varying the widths and spacing of parallel lines, and may be referred to as linear or one-dimensional (1D). Later they evolved into rectangles, dots, hexagons and other geometric patterns in two dimensions (2D). Although 2D systems use a variety of symbols, they are generally referred to as barcodes as well. Barcodes originally were scanned by special optical scanners called barcode readers. Later, scanners and interpretive software became available on devices including desktop printers and smartphones. Foxit SDK provides applications to generate a barcode bitmap from a given string. The barcode types that Foxit SDK supports are listed in Table 4-16. Some common APIs are listed in Table 4-17. For a complete list, please refer to fs_barcode_w.h or API reference [2].

Table 4-16

Barcode Type	Code39	Code128	EAN8	UPCA	EAN13	ITF	PDF417	QR
Dimension	1D	1D	1D	1D	1D	1D	2D	2D

Table 4-17

API name	Description
FSCRT_BCModule_Initialize	Initialize barcode module
FSCRT_BCModule_Finalize	Finalize barcode module.
FSCRT_Barcode_GenerateBitmap	Generate a bitmap of barcode with a given information string and specific barcode format

Example: generate a barcode bitmap from a string.

```
#include "fs_barcode_w.h"
```

```
...
FSCRT_BCModule_Initialize();
FS_INT32 format = FSCRT_BARCODEFORMAT_QR_CODE;
FS_INT32 unitWidth = 2;
FS_INT32 unitHeight = 120;
FSCRT_BITMAP *bitmap = new FSCRT_BITMAP;
FSCRT_BStr_SetLength(&info, strlen("070429"));
strcpy(info.str, "070429");
ret = FSCRT_Barcode_GenerateBitmap(&info, format, unitWidth, unitHeight, 0, bitmap);
if(ret == FSCRT_ERRCODE_SUCCESS)
{
	...	//save bitmap to a bitmap file
}
if (bitmap)
{
	...	//release
}
FSCRT_BCModule_Finalize();
...
```

4.15 Reflow

Reflow is a function that rearranges page content when the page size changes. It is useful for applications that have output devices with difference sizes. Reflow frees the applications from considering layout for different devices. This function provides APIs to create, render, release and access properties of 'reflow' pages. Some common APIs are listed in Table 4-18. For a more complete list, please refer to fpdf_reflow_r.h or API reference [2].

Table 4-18

API name	Description
FSPDF_ReflowPage_Create	Create a reflow page from a given PDF page
FSPDF_ReflowPage_Release	Release resources allocated for a reflow page before
FSPDF_ReflowPage_SetSize	Set screen size. It shall be called before FSPDF_ReflowPage_StartReflow
FSPDF_ReflowPage_StartParse	Start parsing progress for a reflow page
FSPDF_RenderContext_StartReflowPage	Start a rendering progress to render a reflow page
FSPDF_ReflowPage_GetFocusData	Get focus data by a given position in device coordinate system

Example: create a reflow page and render it to a bmp file.

```
#include "fpdf_reflow_r.h"
//Assume a pdfPage has been opened here.
//Assume all return values will be checked here.
...

FS_RESULT ret = FSPDF_ReflowPage_Create(pdfPage, &reflowPage);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;
//Set screen size for reflow. This must be set before parse reflow page.
```

```
FS_FLOAT fScreenWidth = 400;
FS_FLOAT fScreenHeight = 600;
FSPDF_ReflowPage_SetSize(reflowPage, fScreenWidth, fScreenHeight);

FSCRT_BITMAP bitmap = NULL;
ret = FSCRT_Bitmap_Create(m_ReflowPageWidth, m_ReflowPageHeight, FSCRT_BITMAPFORMAT_24BPP_RGB,
NULL, 0, &bitmap);

FSCRT_PROGRESS reflowProgress = NULL;
ret = FSPDF_ReflowPage_StartParse(reflowPage, FSPDF_REFLOWFLAG_NORMAL, &reflowProgress);
if (FSCRT_ERRCODE_SUCCESS == ret)
{
	//Continue the progress of paring PDF page.
	ret = FSCRT_ERRCODE_TOBECONTINUED;
	while (FSCRT_ERRCODE_TOBECONTINUED == ret)
		ret = FSCRT_Progress_Continue(reflowProgress, NULL);
	if (FSCRT_ERRCODE_FINISHED == ret)
	{
		FSPDF_RENDERCONTEXT pdfRenderContext = NULL;
		ret = FSPDF_RenderContext_Create(&pdfRenderContext);
		...
		FSCRT_RENDERER renderer = NULL;
		ret = FSCRT_Renderer_CreateOnBitmap(bitmap, &renderer);
		...
		ret = FSPDF_RenderContext_StartReflowPage(pdfRenderContext, renderer, reflowPage,
&reflowProgress);
		...
		while (ret == FSCRT_ERRCODE_TOBECONTINUED) {
			ret = FSCRT_Progress_Continue(reflowProgress, &pause);
			...
		}

	}
	//Release progress resources.
	FSCRT_Progress_Release(reflowProgress);
	reflowProgress = NULL;
}
...
```

4.16 Asynchronous PDF

Asynchronous PDF technique is a way to access PDF pages without loading the whole document when it takes a long time. It's especially designed for accessing PDF files on internet. With asynchronous PDF technique, applications do not have to wait for the whole PDF file to be downloaded before accessing it. Applications can open any page when the data of that page is available. It provides a convenient and efficient way for web reading applications. Some common APIs are listed in Table 4-19. For a complete list, please refer to fpdf_async_r.h or API reference [2].

Table 4-19

API name	Description
FSPDF_Doc_AsyncLoad	Load PDF file in the asynchronous mode
FSPDF_Doc_IsLinearized	Check if an asynchronous file is a linearized PDF file
FSPDF_Doc_GetFirstAvailPageIndex	Get the page index of first available page in a linearized PDF
FSPDF_Doc_IsDocAvail	Check whether document information is available
FSPDF_Doc_IsPageAvail	Check whether a page in the given document is available

Example: open and parse pages with asynchronous mode.

```
#include "fpdf_async_r.h"
... //Assume m_asyncFile is ready. Please refer to examples for details.

//Load PDF file through asynchronous mode
FSCRT_DOCUMENT m_document = NULL;
FS_RESULT ret = FSPDF_Doc_AsyncLoad(m_asyncFile, NULL, &m_document);

FS_BOOL bDocAvail = FALSE;
//Check whether document information is available or not and wait when it's not available.
while(!bDocAvail)
ret = FSPDF_Doc_IsDocAvail(m_document, &bDocAvail);

...

FS_BOOL bPageAvail = FALSE;
//wait for page data available
while(!bPageAvail)
	ret = FSPDF_Doc_IsPageAvail(m_document, index, &bPageAvail);

if (ret == FSCRT_ERRCODE_SUCCESS)
{
	...
	//Start parsing PDF page
	FSCRT_PROGRESS progress = NULL;
	if (FSCRT_ERRCODE_SUCCESS != FSPDF_Page_StartParse(page, FSPDF_PAGEPARSEFLAG_NORMAL,
&progress))
	{
		... //report errors
	}
	//Continue to parse
	if (FSCRT_ERRCODE_FINISHED != FSCRT_Progress_Continue(progress, NULL))
	{
		... //reports errors
	}
}
...
```

4.17 Pressure Sensitive Ink

Pressure **S**ensitive **I**nk (**PSI**) is a technique to obtain varying electrical outputs in response to varying pressure or force applied across a layer of pressure sensitive devices. In PDF, PSI is usually used for hand writing signatures. PSI data are collected by touching screens or handwriting on boards. PSI data contains coordinates and canvas of the operating area which can be used to generate appearance of PSI. Foxit PDF SDK allows applications to create PSI, access properties, operate on ink and canvas, and release PSI. Some common API functions are listed in Table 4-20. For a complete list, please refer to fs_psi_w.h or API reference [2].

Table 4-20

API name	Description
FSCRT_PSI_Create	Create a pressure sensitive ink object
FSCRT_PSI_Release	Destroy a pressure sensitive ink object
FSCRT_PSI_InitCanvas	Initialize canvas for pressure sensitive ink
FSCRT_PSI_SetInkColor	Set color of ink for a pressure sensitive ink object
FSCRT_PSI_AddPoint	Add a point to a pressure sensitive ink object
FSCRT_PSI_Render	Render a pressure sensitive ink object
FSCRT_PSI_ConvertToPDFAnnot	Convert A pressure sensitive ink object to a PDF annotation
FSCRT_PSI_SetInkDiameter	Set ink diameter of a pressure sensitive ink object
FSCRT_PSI_SetOpacity	Set ink opacity of a pressure sensitive ink object

Example: create a PSI and set the related properties for it.

```
#include "fs_psi_w.h"
...

FSCRT_PSI psi;
//Create a pressure sensitive ink
FS_RESULT ret = FSCRT_PSI_Create(FALSE, &psi);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;
ret = FSCRT_PSI_InitCanvas(psi, 1024, 768);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;
//Set ink color of pressure sensitive ink
ret = FSCRT_PSI_SetInkColor(psi, 0xff0000ff);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;
//Set diameter of ink for pressure sensitive ink
ret = FSCRT_PSI_SetInkDiameter(psi, 10);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;
//Set ink opacity of pressure sensitive ink
ret = FSCRT_PSI_SetOpacity(psi, 1.f);
if (ret != FSCRT_ERRCODE_SUCCESS) return ret;
float fPx = 121.304344;
float fPY = 326.684662;
float fPressure = 0.096680;
ret = FSCRT_PSI_AddPoint(psi, fPx, fPy, fPressure, FSCRT_PSI_PT_MOVETO);
...
```

4.18 Wrapper

Wrapper provides a way for users to save their own data related to a PDF document. For example, when opening an encrypted unauthorized PDF document, users may get an error message. In this case, users can still access wrapper data even when they do not have permissions to the PDF content. The wrapper data could be used to provide information like where to get decryption method of this document. Some common APIs for wrapper is shown in Table 4-21. For a complete list, please refer to fpdf_document_r.h and fpdf_document_w.h or API reference [2].

Table 4-21

API name	Description
FSPDF_WrapperData_Init	Initialize wrapper data.
FSPDF_WrapperData_Clear	Clear wrapper data.
FSPDF_Doc_IsWrapper	Check whether a document includes wrapper data.
FSPDF_Doc_GetWrapperOffset	Get the offset of wrapper data.
FSPDF_Doc_GetWrapperData	Get wrapper data.
FSPDF_Doc_SaveAsWrapperFile	Save a PDF document as a wrapper file.

Example: open a document including wrapper data.

```
BOOL isWrapper = FALSE;
FSPDF_Doc_IsWrapper(document, &isWrapper);
If (isWrapper)
{
	FSCRT_FILESIZE fileOffset = {0};
	FSPDF_Doc_GetWrapperOffset(document, &fileOffset);

	CFSCRT_File *pFileStream= new CFSCRT_File();
	if (!pFileStream || !pFileStream->Load(lpszPathName)) return FALSE;
	pFileStream->m_fileSize = fileOffset.loSize;

	if (FSCRT_File_Create(pFileStream, &m_file) != FSCRT_ERRCODE_SUCCESS)
	{
		//...
	}

	FSCRT_DOCUMENT wrapperDoc;
	if (FSPDF_Doc_StartLoad(m_file, NULL, &wrapperDoc) != FSCRT_ERRCODE_SUCCESS)
	{
		//...
	}
}
...
```

4.19 PDF Objects

There are eight types of object in PDF: Boolean object, numerical object, string object, name object, array object, dictionary object, stream object and null object. PDF objects are document level objects that are different from page objects (see 4.20) which are associated with a specific page each. Foxit PDF SDK provides APIs to create, modify, retrieve and delete these objects in a document. Some common APIs are listed in Table 4-22. For a complete list, please refer to fpdf_objets_r.h and fpdf_objets_w.h or API reference [2].

Table 4-22

API name	Description
FSPDF_Doc_GetCatalog	Retrieve a catalog dictionary object from a document.
FSPDF_Doc_GetInfoDict	Retrieve a document info dictionary object from a document.
FSPDF_Doc_GetEncryptDict	Retrieve a document encrypt dictionary object from a document
FSPDF_Object_GetType	Get the type of a given PDF object
FSPDF_Object_GetObjNum	Get the object number of a given PDF object
FSPDF_Object_GetBoolean	Get the boolean value of a given PDF object
FSPDF_Object_GetRect	Retrieve a position rectangle from a PDF object
FSPDF_Object_GetMatrix	Retrieve a matrix from a given PDF object
FSPDF_Object_CreateBoolean	Create a boolean PDF object
FSPDF_Object_CreateInteger	Create an integer number PDF object
FSPDF_Object_CreateMatrix	Create a matrix PDF object

Example: set "PageLayout" property to "TwoColumnRight" in the catalog dictionary.

```
#include "fpdf_objects_r.h"
#include "fpdf_objects_w.h"
...

FSPDF_OBJECT catalogObj = NULL;
FS_RESULT ret = FSPDF_Doc_GetCatalog(pdfDoc, &catalogObj);
if (ret != FSCRT_ERRCODE_SUCCESS)
return ret;

FSCRT_BSTR bstrKey;
FSCRT_BSTR bstrValue;
FSCRT_BStr_InitConstString(bstrKey, "PageLayout");
FSCRT_BStr_InitConstString(bstrValue, "TwoColumnRight");

ret = FSPDF_Dictionary_SetAtUnicodeName(pdfDoc, catalogObj, &bstrKey, &bstrValue);
FSCRT_BStr_Clear (bstrKey);
FSCRT_BStr_Clear(bstrValue);
...
```

4.20 Page Object

Page object is a feature that allows novice users having limited knowledge of PDF objects to be able to work with text, path, image, and canvas objects (see 4.19 for details of PDF Objects). Foxit PDF SDK provides APIs to add and delete PDF objects in a page and set specific attributes. Using page object, users can create PDF page from object contents. Other possible usages of page object include adding headers and footer to PDF documents, adding an image logo to each page, or generating a template PDF on demand. Some common APIs are listed in Table 4-23. For a complete list, please refer to fpdf_pageobjects_r.h and fpdf_pageobjects_w.h or API reference [2].

Table 4-23

API name	Description
FSPDF_Page_GetPageObjects	Get page objects in a page
FSPDF_PageObjects_GetObject	Get a page object from page objects
FSPDF_PageObject_GetType	Get type of a page object
FSPDF_TextObject_GetTextState	Get text states of a text object
FSPDF_ImageObject_CloneBitmap	Clone a bitmap from an image object
FSPDF_PageObject_GetClipPath	Get a clip path from a page object
FSPDF_PageObjects_InsertObject	Insert a page object and it will be automatically freed
FSPDF_PageObject_SetColor	Set color of a page object
FSPDF_PageObject_AddClipPath	Add path for clipping a page object

Example: create a text object in a page

```
#include "fpdf_pageobjects_r.h"
#include "fpdf_pageobjects_w.h"
...

FSPDF_PAGEOBJECT textObj = NULL;
ret = FSPDF_TextObject_Create(page, &textObj);
if (ret == FSCRT_ERRCODE_SUCCESS)
{
	//Set text states to the text object.
	FSPDF_TextObject_SetTextState(page, textObj ,&textStateTemp, italic, weight);

	//Set matrix to the text object.
	FSCRT_MATRIX textmatrix = {1, 0, 0, 1, 100, 600};
	FSPDF_PageObject_SetMatrix(page, textObj, &textmatrix);

	...
}
```

4.21 Marked content

In PDF document, a portion of content can be marked as marked content element. Marked content helps to organize the logical structure information in a PDF document and enables stylized tagged PDF. Tagged PDF has a standard structure types and attributes that allow page content to be extracted and reused for other purposes. More details about marked content and tagged PDF could be found in chapter 10.5 of PDF reference 1.7 [1]. Some common APIs for marked content is shown in Table 4-24. For a complete list, please refer to fpdf_markedcontent_r.h and fpdf_markedcontent_w.h or API reference [2].

Table 4-24

API name	Description
FSPDF_PageObject_GetMarkedContent	Get the marked content object from a page object
FSPDF_MarkedContent_GetTagName	Get the tag name of a specific marked content item
FSPDF_MarkedContent_AddItem	Add a new marked content item to the current marked content object
FSPDF_MarkedContent_DeleteItem	Delete a specific marked content item from current marked content sequence

Example: get marked content in a page and get the tag name

```
#include "fpdf_markedcontent_r.h"
... //Assuming an FSCRT_PAGE page, and an FSPDF_PAGEOBJECT pageOjb has been obtained

FSPDF_MARKEDCONTENT mc = NULL;
FS_RESULT ret = FSPDF_PageObject_GetMarkedContent(page, pageObj, &mc);
if (ret != FSCRT_ERRCODE_SUCCESS)
       return;

FS_INT32 count;
ret = FSPDF_MarkedContent_CountItems(page, mc, &count);
FSCRT_BSTR tagName;
FSCRT_BStr_Init(&tagName);
ret = FSPDF_MarkedContent_GetTagName(page, mc, 0, &tagName);

...
```

4.22 How to utilize OOM provided by PDF SDK to implement robust applications on mobile platforms

As described in “What’s new”, the OOM mechanism is introduced in PDF SDK to support developers to implement robust applications on mobile platforms. In this chapter, more details will be introduced, such as the conceptions of OOM, OOM APIs and return values, OOM recovery and sample codes.

4.22.1 Introduction to OOM conceptions

OOM means out-of-memory. It can be a scenario that memory is exhausted and applications may have to exit or crash directly. It also represents a solution or mechanism for handling with this scenario in Foxit PDF SDK. In the following descriptions, either of them will be used without emphasized.

As mentioned above, OOM is an evolved feature in Foxit PDF SDK due to its complexity. Currently, Foxit PDF SDK provides auto-recovery for non-editing functionalities of documents and pages. The related functionalities are implemented in the function of **FSCRT_Library_OOMRecover**. During the recovery process, Foxit PDF SDK reloads the document and page automatically and restores the status to the original before OOM. However, the data generated from editing will be lost.

Basic concept of memory: Long-term and Short-term.

- Long-term: The memory managed by the handle is always valid. Even the OOM event happens, the long-term handle is always still accessible.
- Short-term: The memory managed by the handle becomes invalid when the OOM event happens. In that situation, user should invoke APIs to receive a new short-term handle.

Foxit PDF SDK categorizes all the handles in PDF SDK into three types based on their OOM recovery schemes:

- **Long-term recoverable handle**

This handle is usable when OOM happens. Its contents are completely stored in long-term memory, or it relies on short-term handle that has been recovered. In either scenario, it is still valid after OOM.

- **Long-term, partially recoverable handle**

This handle is usable after OOM happens but relies on a short-term handle that has not been recovered, making its content not accessible.

- **Short-term handle**

This handle is invalid and should not be used after OOM.

Every handle in Foxit PDF SDK belongs to one of the three categories described above, which you can reference from the API reference [2].

We also categorize all the APIs into three types:

- **Long term, recoverable API**

This type of API uses long-term recoverable handle. These APIs are not affected by OOM.

- **Long term, partially recoverable API**

This type of API uses long-term partially recoverable handle. These APIs are affected by OOM and users should implement the recovery based corresponding instructions.

- **Short term API**

This type of API uses short-term handle. These APIs are affected by OOM and users should implement the recovery based on corresponding instructions.

Every API in Foxit SDK belongs to one of the three categories described above, which you can reference from the API reference [2].

Due to the complexity of the recovery scheme after OOM, Foxit PDF SDK only supports thread safety for single thread mode. The thread safety for multi-thread is not guaranteed.

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

a) Prompt users of OOM and exit the application.

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

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

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

4.22.2 Introduction to OOM APIs and return value

The key of OOM mechanism is to report OOM to applications when OOM happens and then PDF SDK or applications take recovery operations based on different scenarios. It's required that Foxit PDF SDK shall work together with applications in some ways to deal with OOM. Based on this, it's designed in PDF SDK that some OOM APIs and OOM related return values to synchronize OOM status with applications.

1) *OOM APIs*

a. FS_RESULT (***OnRecover**)(FS_LPVOID clientData, FS_LPVOID senderObject, FS_DWORD senderObjectType);

This function is a callback function which is defined in **FSCRT_APPHANDLER** of fs_app_r.h. The **FSCRT_APPHANDLER** can be set to Foxit PDF SDK for applications by calling **FSCRT_Library_SetAppHandler**. This function may be implemented by applications to do recovery operations and will be called by Foxit PDF SDK when OOM happens. If this function isn't implemented by applications, Foxit PDF SDK performs recovery operations by default.

b. FS_RESULT **FSCRT_Library_OOMRecover**(FS_LPVOID senderObject, FS_DWORD senderObjectType);

This function is designed for applications to call when APIs return **FSCRT_ERRCODE_MEMORYREBUILT or FSCRT_ERRCODE_UNRECOVERABLE or FSCRT_ERRCODE_ROLLBACK** to indicate that applications need to rebuild memory and recover data. The callback function of **OnReocver** will be called automatically by this function.

c. FS_RESULT **FSCRT_Library_TriggerRecover**(FS_LPVOID senderObject, FS_DWORD senderObjectType)

This function is designed for applications to call when APIs return **FSCRT_ERRCODE_MEMORYREBUILT** to indicate that applications need to recover documents.

2) *Return values in OOM APIs*

Foxit PDF SDK monitors memory usage status dynamically when applications run in the limited memory environment. PDF SDK can detect OOM and inform applications by returning different values once OOM occurs.

There are 4 return values to indicate applications what happens:

a. FSCRT_ERRCODE_MEMORYREBUILT

It will be returned from short term interfaces or long term interfaces with partially recovered when OOM is detected by PDF SDK. The long term interfaces with partially recovered will only be returned when the documents or pages have been edited. If this return value is returned from short-term interfaces, the parameters in these interfaces will be invalid when OOM happens. These parameters need to be reconstructed after OOM before they are used.

Applications can customize recovery operations in the callback function **OnRecover** and call the function **FSCRT_Library_OOM_Recover** to recover or re-build the related objects when receiving the indication, **FSCRT_ERRCODE_MEMORYREBUILT**, from PDF SDK before continuing to run.

b. FSCRT_ERRCODE_UNRECOVERABLE

It indicates that OOM occurs and the interface can't be executed in the limited memory environment which are detected by PDF SDK. Applications can skip this interface or request more memory to execute it.

c. **FSCRT_ERRCODE_OUTOFMEMORY**

It is returned when memory is exhausted. It indicates that applications have to restart.

d. **FSCRT_ERRCODE_ROLLBACK**

It is returned when OOM occurs during the period of PDF SDK handling progressive operations. Applications need to call **FSCRT_Library_OOM_Recover** to recover and take different OOM actions according to varied progressive operations, such as re-rendering the background of a given page, reset the position and size of saved files or other users involved recovering operations. Foxit PDF SDK will list all progressive related interfaces.

4.22.3 Implement OOM recovery in applications and sample codes

Based on the description of OOM above, it will be introduced how to implement applications with OOM recovery supported in this chapter. Some samples are provided to help developers understand how OOM works and how to implement applications with OOM supported.

Foxit PDF SDK has four return values which are introduced in chapter 2) to indicate different types of OOM. When applications are informed of OOM with a given return value, there are two ways to deal with OOM:

a. Exit application. The application has to be restarted to continue.

b. Continue the application but lose some data, including those obtained from PDF document (see instructions from handle/API reference) and some of the changes made to the PDF document. If no change has been made and no short-term handle has been used, then no further action needs to be taken. Foxit PDF SDK has automatically recovered from OOM.

Foxit PDF SDK provides some OOM recovery interfaces (as explained in the API reference [2]) to help applications recover some of “lost” data.

There are 3 examples for application developers to reference for developing PDF projects with OOM supported.

Example 1: An example to show how to do recovery in applications after OOM indication is received from Foxit PDF SDK.

```
FSCRT_FILE  hFile = NULL;
FSCRT_DOCUMENT document= NULL;
int pageCounts = 0;
FSCRT_APPHANDLER  appHandler;
appHandler.clientData =  &appHandler;
```

```
appHandler.OnRecover = _OnRecover;
//This function is called by applications to set appHandler with the related callback
functions to Foxit PDF SDK.
FSCRT_Library_SetAppHandler(&appHandler);
//Extend the function of memory allocation. Implementation to FSCRT_MEMMGRHANDLER::Alloc
static FS_LPVOID     FSDK_Alloc(FS_LPVOID clientData, FS_DWORD size) {
         return malloc((size_t)size);
}
//Extend the function of memory reallocation. Implementation to FSCRT_MEMMGRHANDLER::Realloc
static FS_LPVOID     FSDK_Realloc(FS_LPVOID clientData, FS_LPVOID ptr, FS_DWORD newSize) {
         return realloc(ptr, (size_t)newSize);
}
//Extend the function of memory free. Implementation to FSCRT_MEMMGRHANDLER::Free
static void FSDK_Free(FS_LPVOID clientData, FS_LPVOID ptr) {
         free(ptr);
}

#define FSDK_GLOBALBUFFER_SIZE1     (1024 * 1024 * 50)
#define FSDK_GLOBALBUFFER_SIZE2     (1024 * 1024 * 100)
//Global variables for extension manager
static FSCRT_MEMMGRHANDLER g_MemMgrHandler = {NULL, FSDK_Alloc, FSDK_Realloc, FSDK_Free};
FS_LPVOID g_pGlobalBuffer = malloc(FSDK_GLOBALBUFFER_SIZE1);

FSCRT_Library_CreateMgr(g_pGlobalBuffer, FSDK_GLOBALBUFFER_SIZE1, &g_MemMgrHandler);
FSCRT_PDFModule_Initialize();
//Open the input file
hFile = FSDK_OpenFileW(L"test.pdf", L"r+b", 0);
// Recover in foxit sdk.
FSPDF_Doc_StartLoad(hFile, NULL, &document, NULL);
FS_RESULT ret = FSPDF_Doc_CountPages(document, &pageCounts);
//Check the return value from OOM APIs
if (ret == FSCRT_ERRCODE_OUTOFMEMORY || ret == FSCRT_ERRCODE_UNRECOVERABLE) {
       //Destroy the current PDF module.
       FSCRT_PDFModule_Finalize();
       //Destroy the current SDK manager.
       FSCRT_Library_DestroyMgr();
       free(g_pGlobalBuffer);
       g_pGlobalBuffer = malloc(FSDK_GLOBALBUFFER_SIZE2);
       pdfDocument = NULL;
       //Rebuild the SDK manager to use the larger memory to run the App.
       FSCRT_Library_CreateMgr(g_pGlobalBuffer, FSDK_GLOBALBUFFER_SIZE2, &g_MemMgrHandler);
       //Reinitialize the PDF module.
       FSCRT_PDFModule_Initialize();

       //User should do these steps again.
       hFile = FSDK_OpenFileW(L"test.pdf", L"r+b", 0);
       FSPDF_Doc_StartLoad(hFile, NULL, &document, NULL);
       ret = FSPDF_Doc_CountPages(document, &pageCounts);
       ...
}
```

Example 2: A page refresh example to show how to deal with the return value 'FSCRT_ERRCODE_MEMORYREBUILT'.

```
FSCRT_PAGE  pdfPage = NULL;
FSCRT_ANNOT annot1 = NULL;
FSCRT_ANNOT annot2 = NULL;
FSCRT_ANNOT annot3 = NULL;
FSCRT_ANNOT FSCRT_ANNOT[3];
static FS_RESULT _OnRecover(FS_LPVOID clientData, FS_LPVOID senderObject, FS_DWORD
senderObjectType, FS_DWORD eventType, FS_LPVOID eventData)
{
	//....
	annot1 = NULL;
	annot2 = NULL;
	annot3 = NULL;
	//Because the FSCRT_ANNOT handle is a short-term handle, we need to reload annotations.
	FSPDF_Page_LoadAnnots(pdfPage);
	FSPDF_Annot_Get(pdfPage, NULL, 0, &annot1);
	FSPDF_Annot_Get(pdfPage, NULL, 1, &annot2);
	FSPDF_Annot_Get(pdfPage, NULL, 2, &annot3);
	annots[0] = annot1;
	annots[1] = annot2;
	annots[2] = annot3;
	//.....
}

//...
FSPDF_Annot_Get(pdfPage, NULL, 0, &annot1));
FSPDF_Annot_Get(pdfPage, NULL, 0, &annot2));
FSPDF_Annot_Get(pdfPage, NULL, 0, &annot3));
annots[0] = annot1;
annots[1] = annot2;
annots[2] = annot3;
FS_INT32 count = 3;
FSPDF_RENDERCONTEXT renderContext;
FSPDF_RenderContext_Create(&renderContext);
FSCRT_RENDERER render;
FSCRT_BITMAP bitmap = NULL;
FSCRT_PROGRESS progress = NULL;
FSCRT_Bitmap_Create(width, height, FSCRT_BITMAPFORMAT_32BPP_RGBA, NULL, 0, &bitmap);
FSCRT_Renderer_CreateOnBitmap(bitmap, &render);
FS_RESULT ret = FSPDF_RenderContext_StartAnnots(renderContext,render,annots,count,&progress);
if (ret == FSCRT_ERRCODE_MEMORYREBUILT) {
//Applications call FSCRT_Library_OOM_Recover to trigger OnRecover to be called by Foxit PDF
SDK.
FSCRT_Library_OOM_Recover(pdfDocument, FSCRT_OBJECTTYPE_DOCUMENT);
 //Recall FSPDF_RenderContext_StartAnnots.
FSPDF_RenderContext_StartAnnots(renderContext,render,annots,count,&progress);
}
FSCRT_Progress_Continue(progress, NULL);
//...
```

Example 3: A page refresh example to show how to deal with the return value 'FSCRT_ERRCODE_ROLLBACK'.

```
//....
HBRUSH hBrush =   CreateSolidBrush(RGB(0xff, 0xff, 0xff));
FillRect(hDC, &rect, hBrush);
while (ret == FSCRT_ERRCODE_TOBECONTINUED || ret == FSCRT_ERRCODE_ROLLBACK)
{
       //if OOM is detected, applications need to call FSCRT_Library_OOMRecover to do recovery.
       if (ret == FSCRT_ERRCODE_ROLLBACK) {
       ret = FSCRT_Library_OOMRecover(m_document.m_document, FSCRT_OBJECTTYPE_DOCUMENT);
       //Users need to refresh the background of a page here.
       FillRect(hDC, &rect, hBrush);
       }
       ret = FSCRT_Progress_Continue(progress, &pause);
}

DeleteObject(hBrush);
//....
```

4.23 Layer

PDF Layers, in other words, Optional Content Groups (OCG), now supported in Foxit PDF SDK 4.1. Users can selectively view or hide the contents in different layers of a multi-layer PDF document. Multi-layer PDF is widely used in many application domains such as CAD drawings, maps, layered artwork and multi-language document, etc. **FSPDF_LayerContext_Create** can create a view layer context with a given type. **FSPDF_Doc_EnumLayers** could enumerate all PDF layers of a PDF document. Some common APIs for Layers processing are listed in Table 4-26. For a complete list, please refer to fpdf_layer_r.h or API reference [2]. An example shows how to traverse layer tree and set the opposite visible state of every PDF layer.

Table 4-26

API name	Description
FSPDF_Layer_GetName	Get a name of a PDF layer
FSPDF_LayerNode_Init	Initialize a PDF layer node
FSPDF_LayerContext_IsVisible	Check whether a PDF layer is visible or not
FSPDF_LayerContext_SetVisible	Change a PDF layer visibility state
FSPDF_Layer_IsInPage	Check whether a PDF layer is in a given page or not
FSPDF_LayerContext_CopyStates	Copy a PDF layer context state from another PDF layer context

Example: traverse layer tree and set the opposite visible state of every PDF layer

```
#include "fpdf_layer_r.h "
... //Assuming a FSPDF_LAYERNODE pLayers and its depth have been obtained
```

```
... //Assuming the layer node is a PDF layer

if (pLayers->children == NULL && pLayers->count == 0)
	{
		for (int j = 0; j < depth + 1; j++)
		//Define a FSCRT_BSTR object to store layer name
		FSCRT_BSTR layerName;
		nRet = FSCRT_BStr_Init(&layerName);//Initialize layer name
		nRet = FSPDF_Layer_GetName(pLayers->layer, &layerName);//Get the layer name
		if (FSCRT_ERRCODE_SUCCESS == nRet)
		{
			//Get layer name  successfully and output layer name
			FSDK_OutputLog("Layer Name: ");
			string name(layerName.str, layerName.len);
			FSDK_OutputStringLog(name.c_str(), (int)name.length());

			//Get layer visible state and output visible state
			FS_BOOL isVisible = FALSE;
			nRet = FSPDF_LayerContext_IsVisible(context, pLayers->layer, &isVisible);
			FSDK_OutputLog("/t visible: %d/r/n", isVisible);
			//Set the opposite visible state of getting
		       nRet = FSPDF_LayerContext_SetVisible(context, pLayers->layer, !isVisible);
		}
		//Clear layer name
		nRet = FSCRT_BStr_Clear(&layerName);
...
```

5 FAQ

1. **What's the price of Foxit PDF SDK?**
   To receive a price quotation, please send a request to sales@foxitsoftware.com or call Foxit sales at 1-866-680-3668.

2. **How can I activate after purchasing Foxit PDF SDK?**
   There are detailed steps on how to apply a license in the section 3.2.4 “Unlock PDF SDK license”. You can refer to the steps to activate a license.

3. **How can I look for technical support when I try Foxit PDF SDK?**
   You can send email to support@foxitsoftware.com for any questions or comments or call our support at 1-866-693-6948.

REFERENCES

[1] PDF reference 1.7
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[2] Foxit PDF SDK API reference
sdk_folder/docs/Foxit PDF SDK 4.1 API Reference.chm

[3] Foxit PDF SDK Demo guide
sdk_folder/docs/demo_tutorials.pdf

Note: sdk_folder is the directory of unzipped package.

SUPPORT

Foxit support home link:

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

Sales contact phone number:

Phone: 1-866-680-3668

Email: sales@foxitsoftware.com

Support & General contact:

Phone: 1-866-MYFOXIT or 1-866-693-6948

Email: support@foxitsoftware.com

GLOSSARY OF TERMS & ACRONYMS

boolean object	Keyword true or false
catalog	The primary dictionary object containing references directly or indirectly to all other objects in the document, with the exception that there may be objects in the trailer that are not referred to by the catalog
character	Numeric code representing an abstract symbol according to some defined character encoding rule
developer	Any entity, including individuals, companies, non-profits, standards bodies, open source groups, etc., who are developing standards or software to use and extend ISO 32000-1
dictionary object	An associative table containing pairs of objects, the first object being a name object serving as the key and the second object serving as the value and may be any kind of object including another dictionary
direct object	Any object that has not been made into an indirect object
FDF file	File conforming to the Forms Data Format containing form data or annotations that may be imported into a PDF file
filter	An optional part of the specification of a stream object, indicating how the data in the stream should be decoded before it is used
font	Identified collection of graphics that may be glyphs or other graphic elements
function	A special type of object that represents parameterized classes, including mathematical formulas and sampled representations with arbitrary resolution
glyph	Recognizable abstract graphic symbol that is independent of any specific design
indirect object	An object that is labelled with a positive integer object number followed by a non-negative integer generation number followed by object and having end object after it
integer object	Mathematical integers with an implementation specified interval centred at 0 and written as one or more decimal digits optionally preceded by a sign
name object	An atomic symbol uniquely defined by a sequence of characters introduced by a SOLIDUS (/), (2Fh) but the SOLIDUS is not considered to be part of the name

null object	A single object of type null, denoted by the keyword null, and having a type and value that are unequal to those of any other object
numeric object	An integer object representing mathematical integers or a real object representing mathematic real numbers
object	Basic data structure from which PDF files are constructed. Types of objects in PDF include: boolean, numerical, string, name, array, dictionary, stream and null
object reference	An object value used to allow one object to refer to another; that has the form “<n> <m> R” where <n> is an indirect object number, <m> is its version number and R is the uppercase letter R
PDF	Portable Document Format file format defined by this specification [ISO 32000-1]
real object	This object used to approximate mathematical real numbers, but with limited range and precision and written as one or more decimal digits with an optional sign and a leading, trailing, or embedded PERIOD (2Eh) (decimal point)
rectangle	A specific array object used to describe locations on a page and bounding boxes for a variety of objects and written as an array of four numbers giving the coordinates of a pair of diagonally opposite corners, typically in the form [llx lly urx ury] specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates of the rectangle, in that order
SDK	Software Development Kits
stream object	This object consists of a dictionary followed by zero or more bytes bracketed between the keywords stream and endstream
string object	This object consists of a series of bytes (unsigned integer values in the range 0 to 255). String objects are not integer objects, but are stored in a more compact format