Foxit

DEVELOPER GUIDE
Foxit® PDF SDK

Microsoft® Partner
Gold Independent Software Vendor (ISV)

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 Foxit is 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 SDK 1.0, Foxit embedded SDK 2.0 and Foxit PDF SDK DLL 3.1. Foxit PDF SDK enables users to develop their applications with C/C++, object C, Java and C# on multi-platforms such as Windows, Linux, Mac, iOS, Android, Windows 8.1 (for Store App), Windows Phone 8.1 and Windows 10 Universal App.

1.3 What is 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 9 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, font information, text extraction and search, access to PDF objects, asynchronous PDF, text reflow, and 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, etc.).
Form	Access form information, import a FDF file into a form and export data to a FDF file.
Annotation	Create, edit and remove annotations. Create watermarks.
Security	Support password, certificate, DRM and custom encryption.
Pressure Sensitive Ink	Generate PSI and convert PSI to annotation.
Barcode	Generate a barcode bitmap from a given string and barcode type.
Signature	Sign a PDF document, verify a signature, add or delete a signature field.
RMS	Support Microsoft RMS encryption and decryption.

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 the way. Foxit PDF SDK is a cross platform SDK product. It supports the same interfaces for desktop system of Windows, Linux, Mac, Windows 8.1 (for Store App), mobile system of iOS, Android, Windows Phone 8.1, and Universal Windows Platform of Windows 10. 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 samples 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 10.10

- **iOS:**

  iOS 5 and later

  All iOS samples have been tested on Xcode 4.6 or later (Xcode 5.0 or later if you need to build iOS 64-bit App).

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

  1. arm v7/v7s/64 Library for applications running on iPhone, iPod and iPad

  2. simulator i386/x86_64 Library for applications running on i386/x86_64 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. armeabi-v7a/arm64-v8a library for arm devices

- **Windows Phone 8.1:**

  Windows 8.1

  Visual Studio 2013 (with update 2) installed.

  The release package includes arm and x86 dynamic link library for Windows Phone device (arm) and emulator (x86).

- **Windows 8.1 Store App:**

  Windows 8.1

  Visual Studio 2013 installed.

  The release package includes arm, x64 and x86 dynamic link libraries for Windows 8.1 device/simulator.

- **Windows 10 Universal App:**

  Windows 10

  Visual Studio 2015 (with Universal Windows App Development Tools) installed

  The release package includes arm, x64 and x86 dynamic link libraries for Windows 10 Universal app.

Note: If you need Foxit PDF SDK for Windows Phone 8.1 or Windows 8.1 Store App, please contact us at support@foxitsoftware.com.

3.2 Windows

In this guide, one thing to note is that the highlighted rectangles in the figures are the version of the SDK. Here the SDK version is 5.3, so it shows 5_3.

3.2.1   What is in the package

Download Foxit PDF SDK zip for Windows package and extract it to a new directory "foxitpdfsdk_5_3_win", which is shown in Figure 3-1. The release package contains the following folders:

docs: API references, demo tutorial, developer guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos
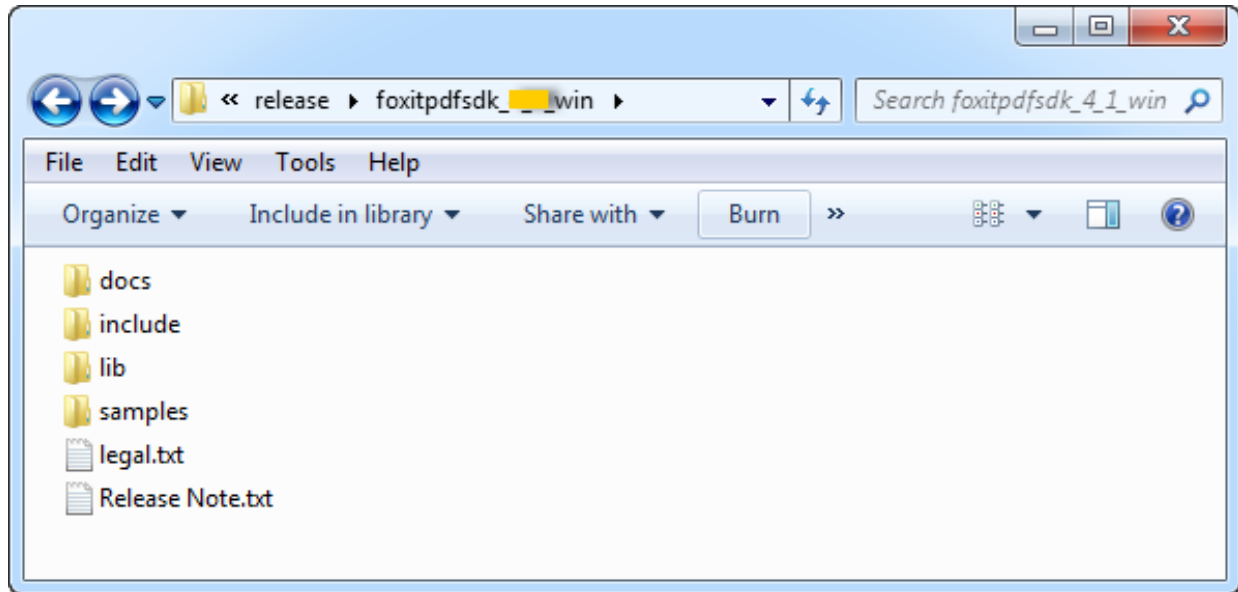
Figure 3-1

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 realize a simple PDF viewer with C++ and C# 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 "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.
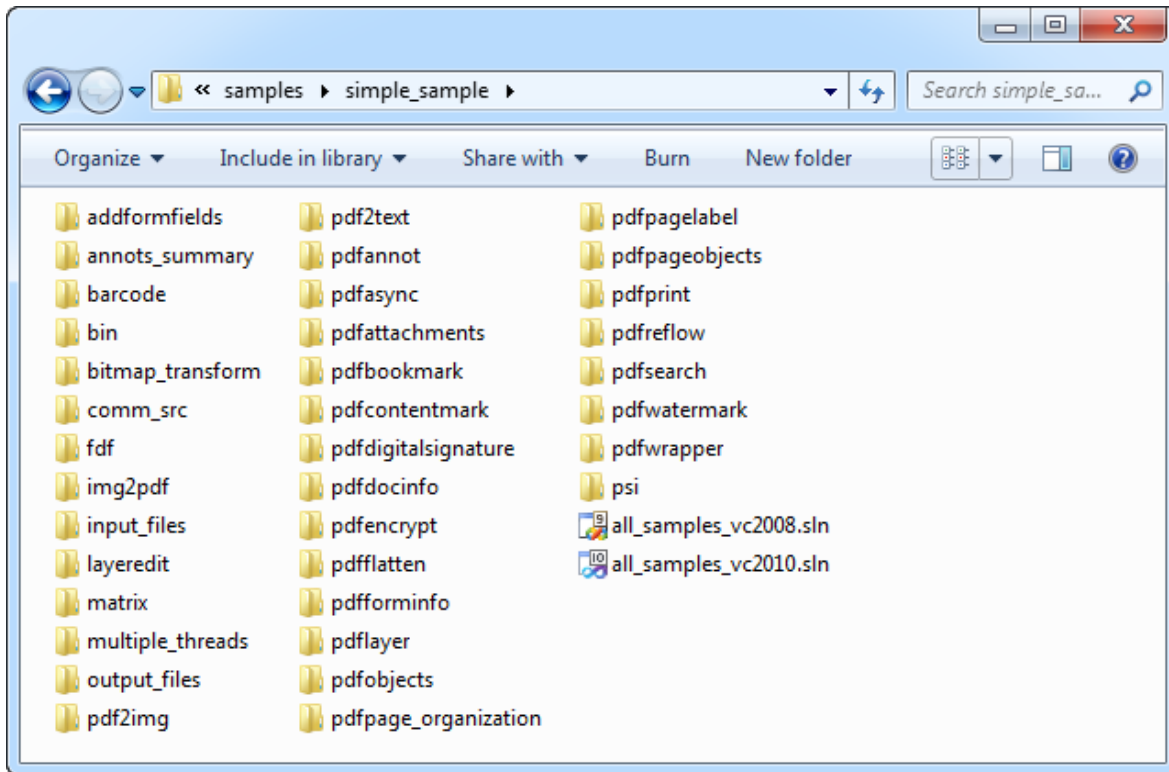
Figure 3-2

For the PDF viewer demo, the C/C++ and C# demos are shown in Figure 3-3. All resources and files are put under "PDFReader_Demo" or "PDFReader_Demo_CSharp" folder.
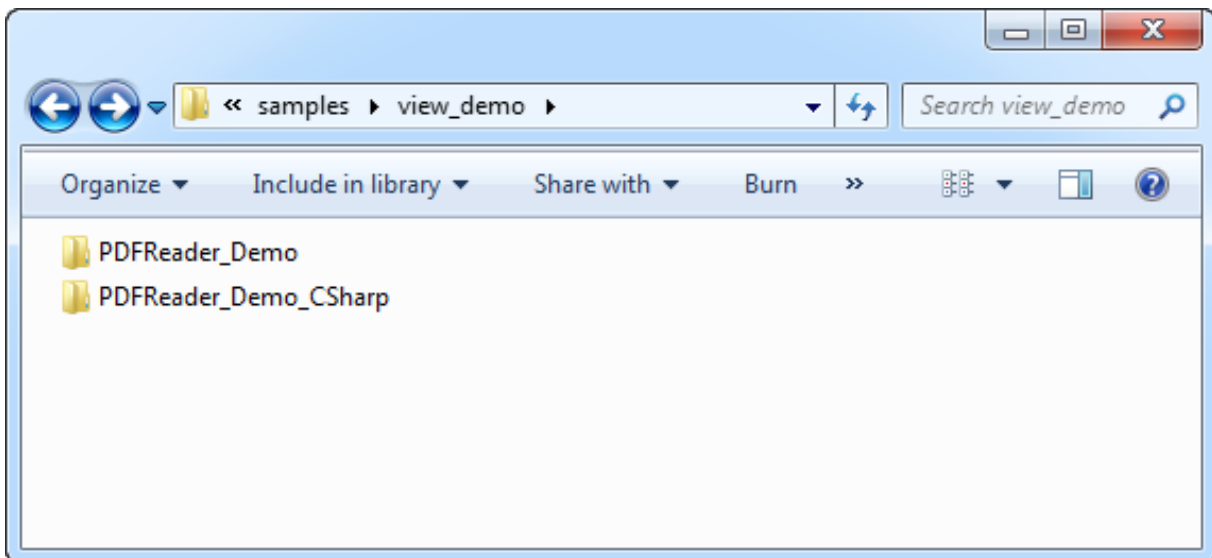
Figure 3-3

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_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.
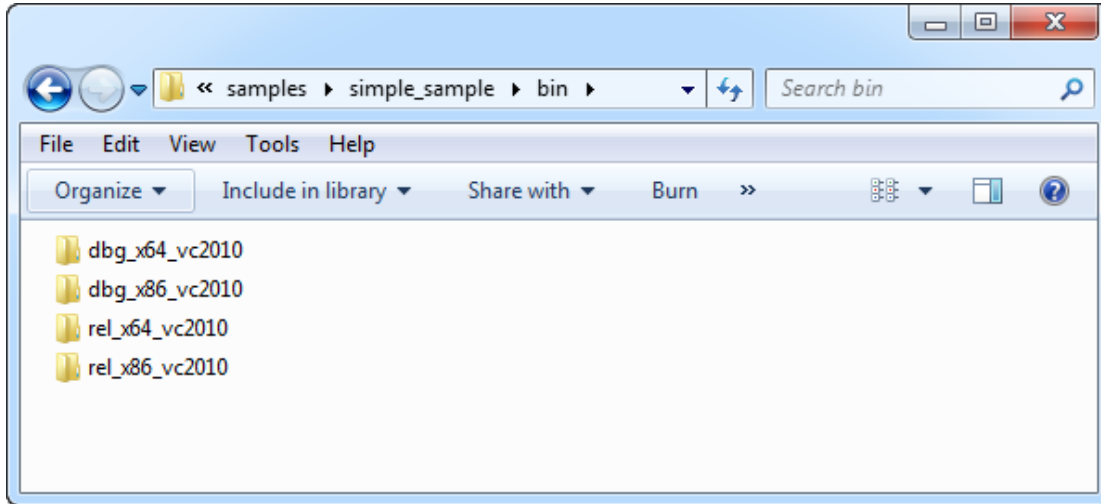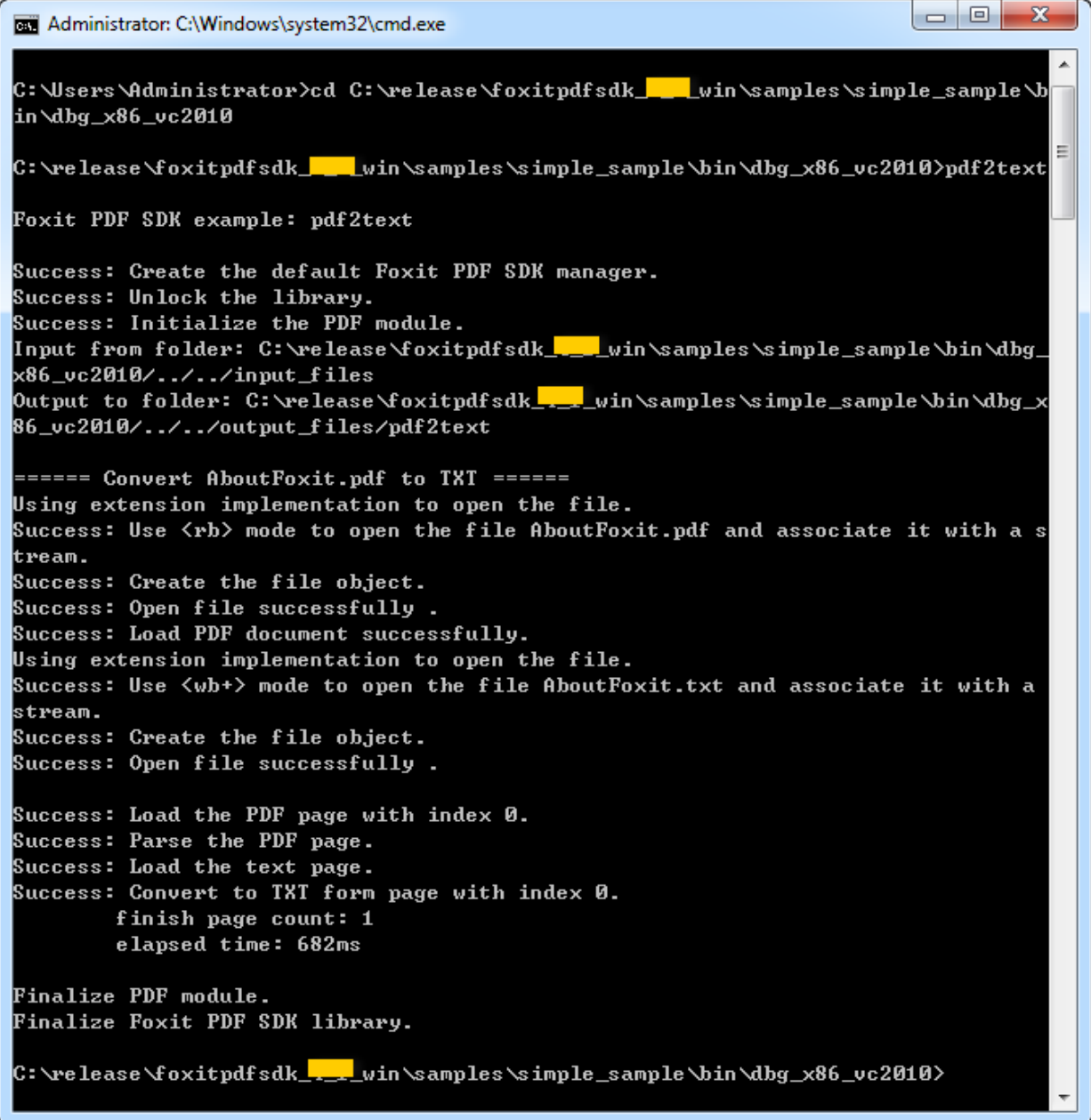
Figure 3-4

To run the executable file, in this example, find it at "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", navigate 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

Some demos will generate output files (pdf, text or image files) to a folder named by the project name under "samples/pdfium_simple_sample/output_files/". In this example, output files are generated to "samples/pdfium_simple_sample/output_files/pdf2text".

PDF Reader Demo (C++)

This view demo provides an example for C++ developers to realize a PDF reader using PDF SDK APIs. To run this demo in Visual Studio, follow the steps below:

a) Open "samples\view_demo\PDFReader_Demo\PDFReader.sln" in Visual Studio 2010 and build the solution.

b) Right click 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 moving 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.
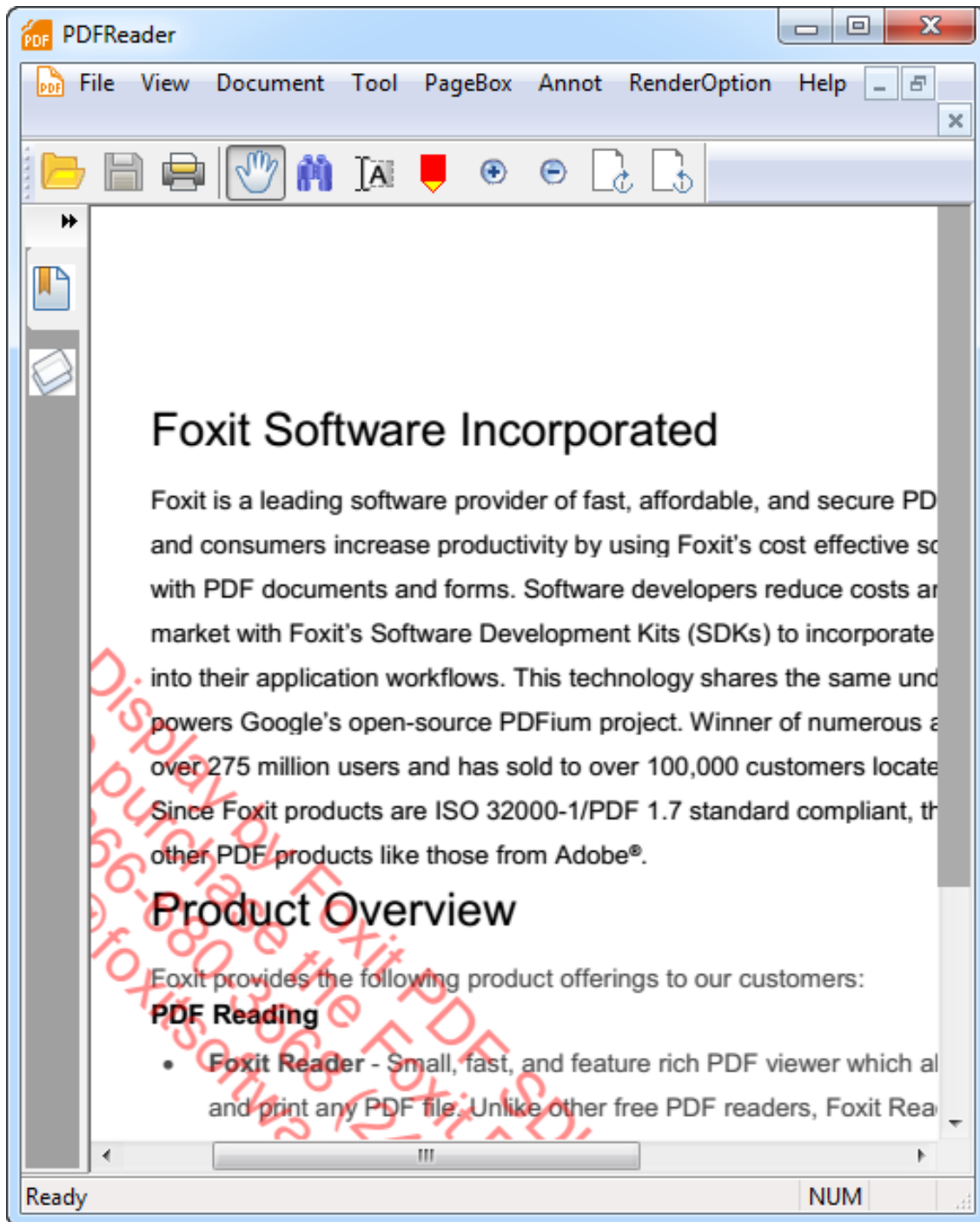
Figure 3-7

PDF Reader Demo (C#)

This view demo provides an example for C# developers to realize a PDF reader using PDF SDK APIs. To run this demo in Visual Studio, follow the steps below:

a) Open "samples\view_demo\PDFReader_Demo_CSharp\PDFReader_Demo_CSharp.sln" in Visual Studio 2010 and build the project "FSDKDLL" by right clicking the project "FSDKDLL" and selecting "Build".

b) Right click the project "PDFReader_Demo_CSharp" 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 moving the PDF page by holding the left mouse button. Click the page turning button to view the previous or next page. A screenshot of the demo is shown in Figure 3-8.
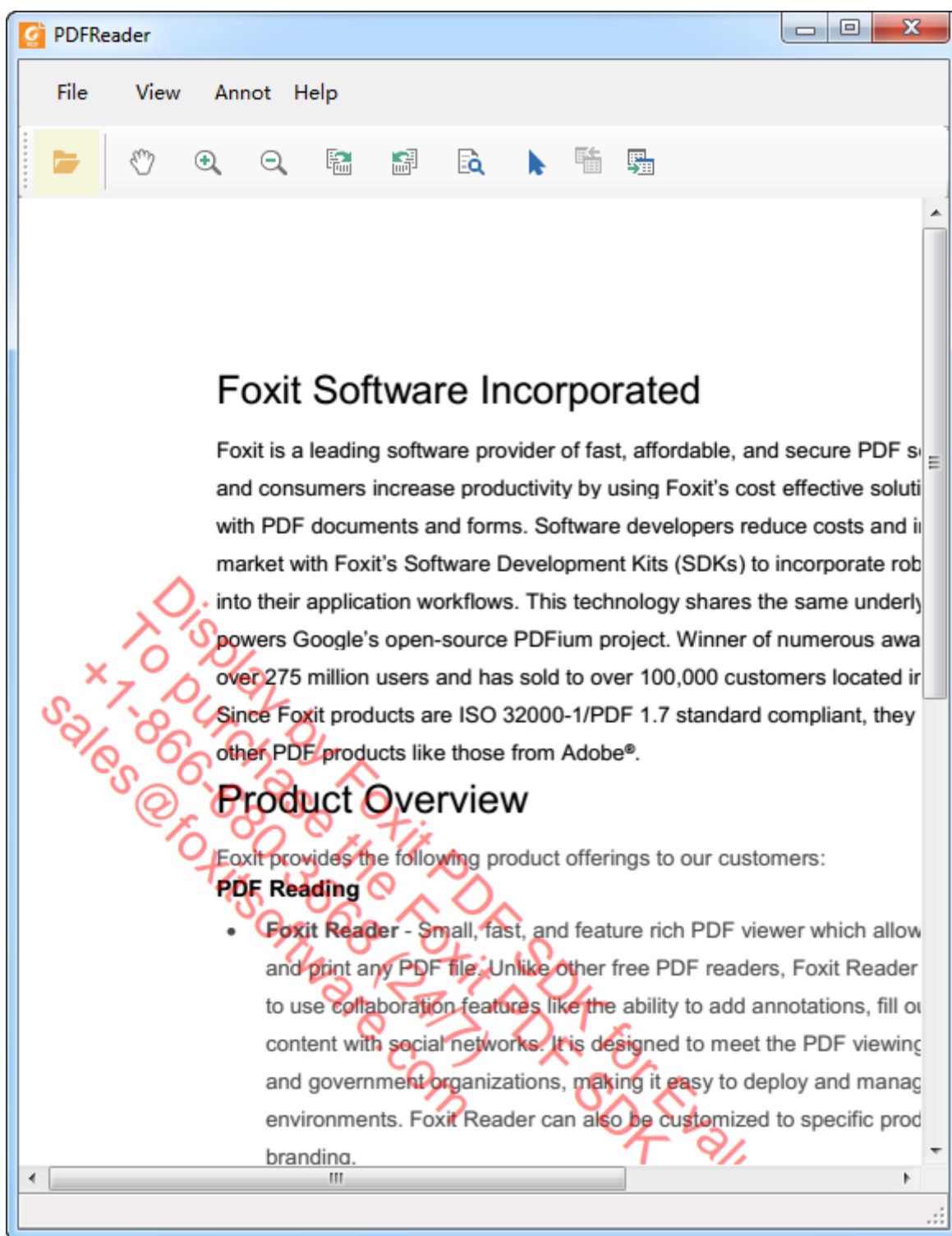
Figure 3-8

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 3-9.
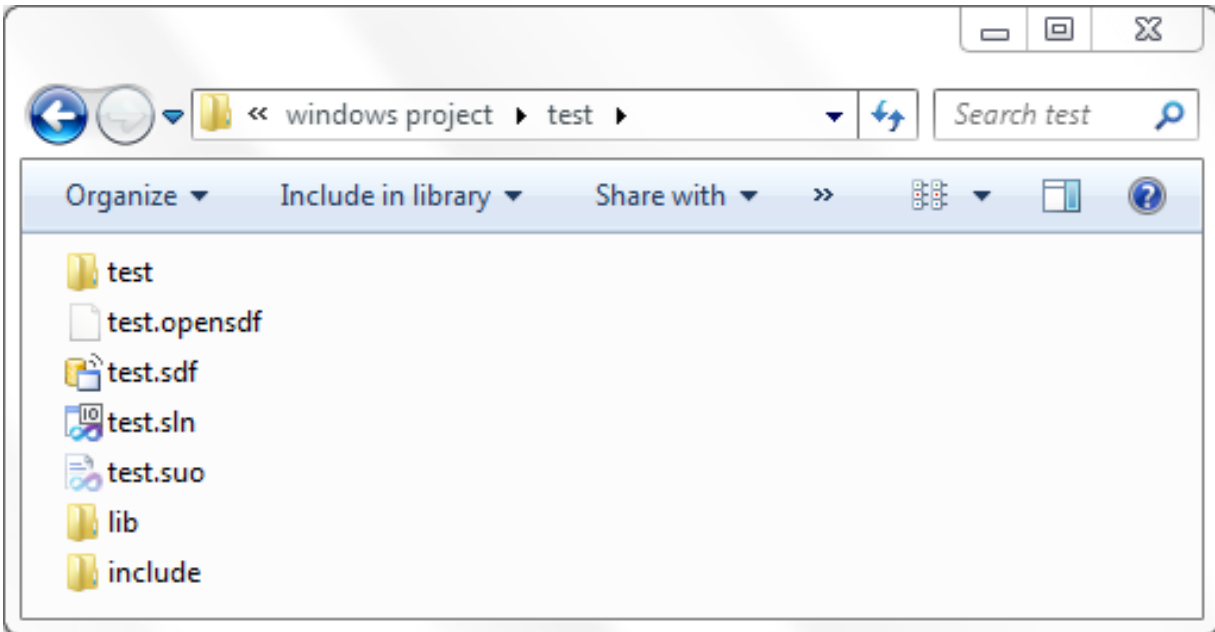
Figure 3-9

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 3-10.
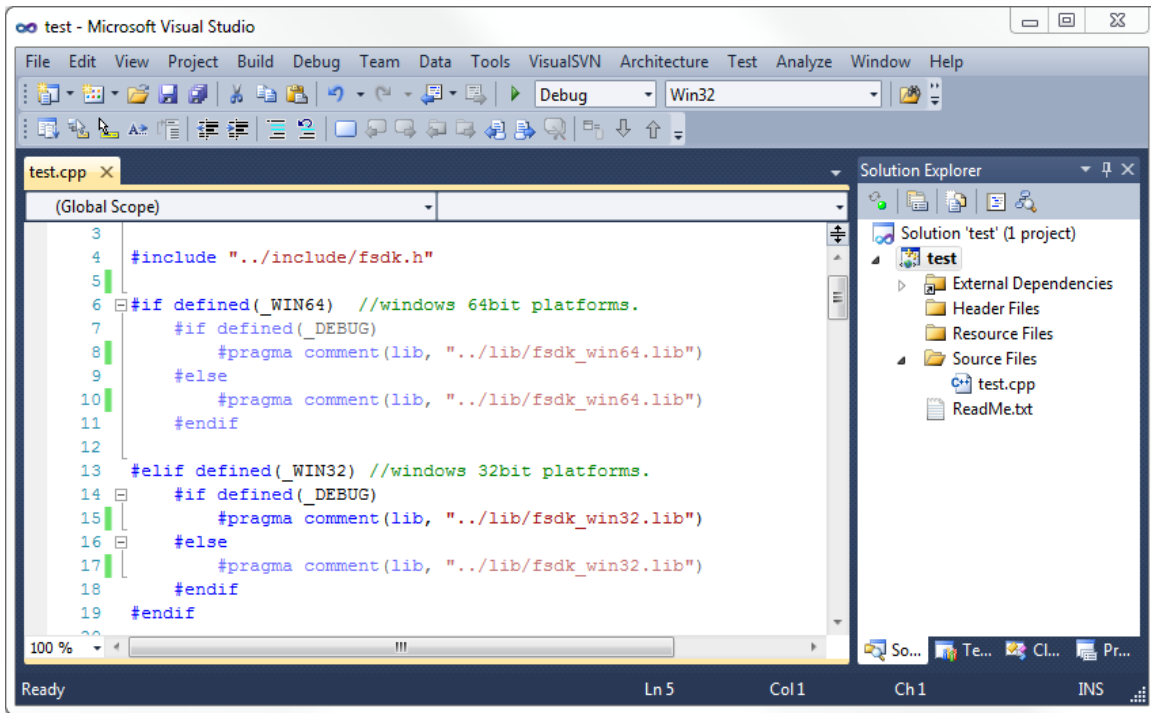
Figure 3-10

b) Construct the code to build a PDF application. The necessary functions and the structure of the code are shown in Figure 3-11. 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" for 64 bit system ("fsdk_win32.dll" and "fsdk_win32.lib" for 32 bit system) to the "Debug" or "Release" folder where the executable files are generated. Run the executable and your project will get running!
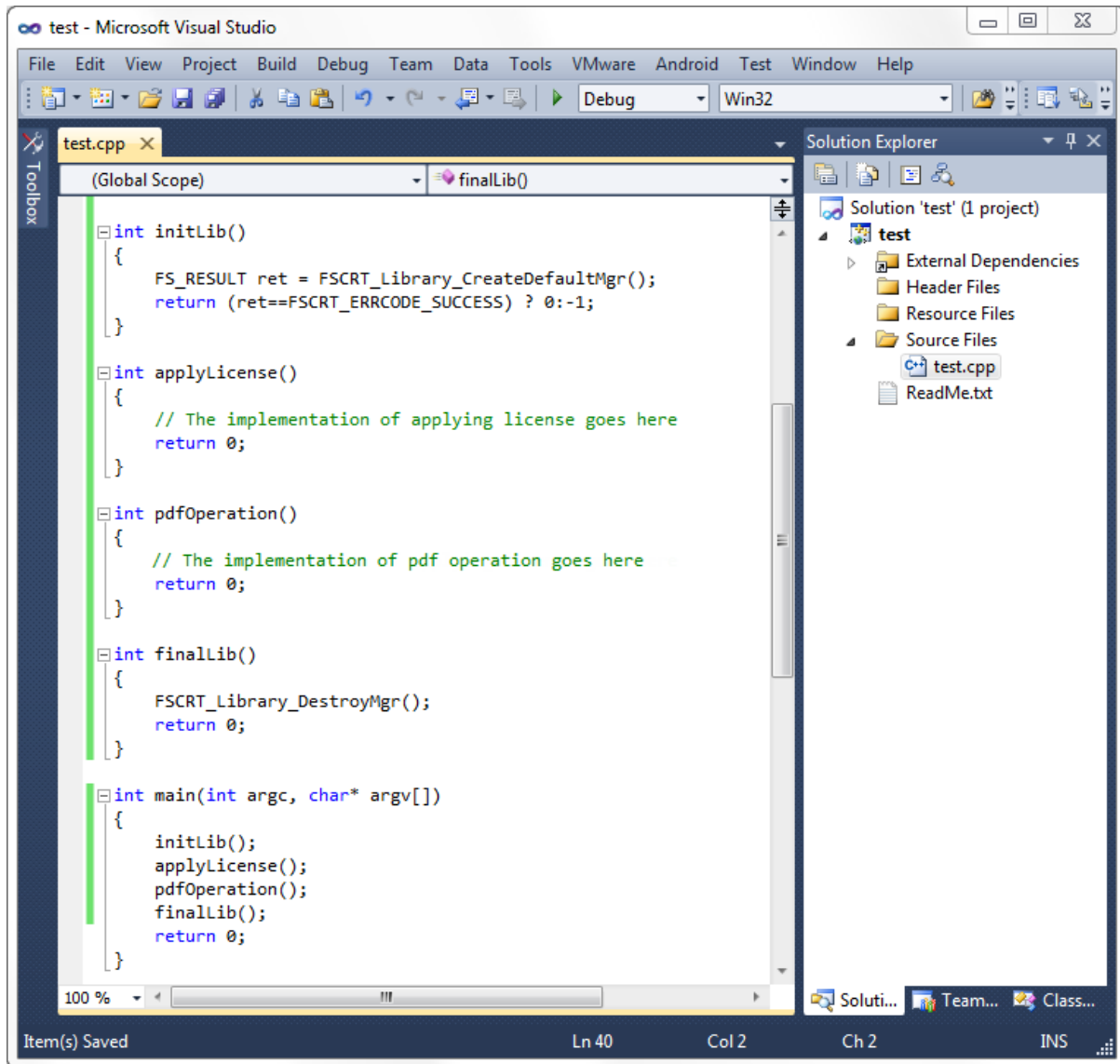
Figure 3-11

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 3-12 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 3-12

3.3 Linux

3.3.1 What is in this package

Download Foxit PDF SDK zip for Linux package and extract it to a new directory "foxitpdfsdk_5_3_linux".The structure of the release package is shown in Figure 3-13. This package contains the following folders:

docs: API references, demo tutorial, developer guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos

Figure 3-13

3.3.2 How to run a demo
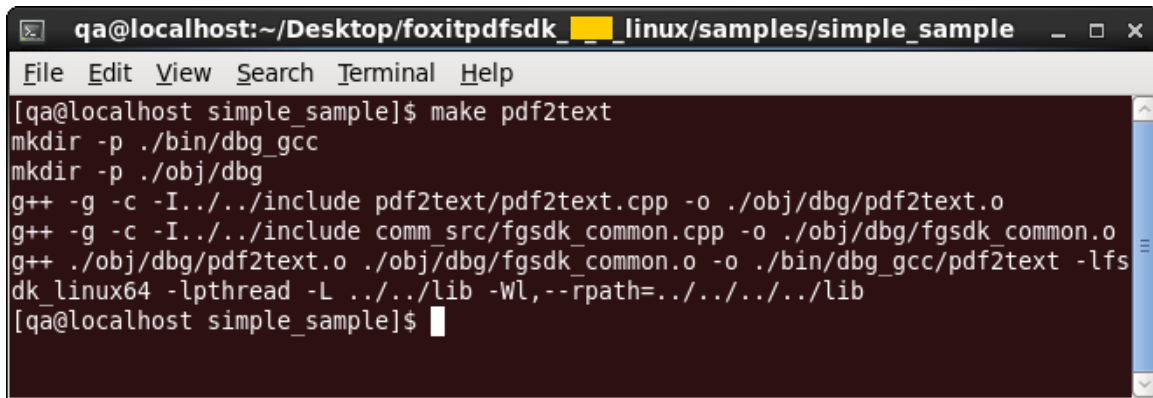
Foxit PDF SDK for Linux provides more than 20 demo projects that cover a wide range of PDF document application. These demos are in folder "samples/simple_samples" as shown in Figure 3-14.

Figure 3-14

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 3-15 shows the build process for pdf2text demo.

Figure 3-15

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. Navigate to the folder with the terminal, and run the binary file to get the demo running. Figure 3-16 shows a screen shot when running pdf2text demo.
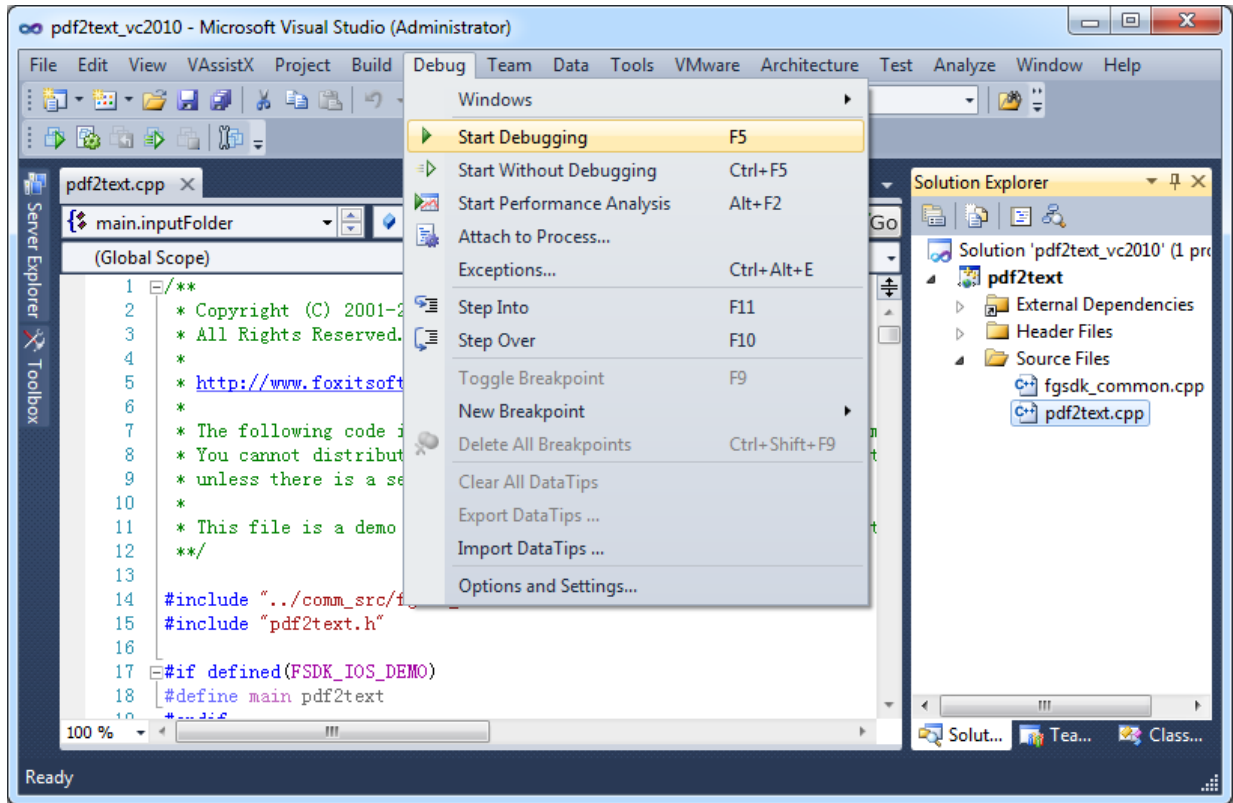

Figure 3-16

20

Some demos will generate output files (pdf, text or image files) to a folder named by the project name under "samples/pdfium_simple_sample/output_files/". In this example, output files are generated to "samples/pdfium_simple_sample/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 3-17.

Figure 3-17

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 3-18 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 (~/Desktop/test_linux) - gedit

File  Edit  View  Search  Tools  Documents  Help

  Open    Save         Undo              

 test.cpp 

#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;
}
```
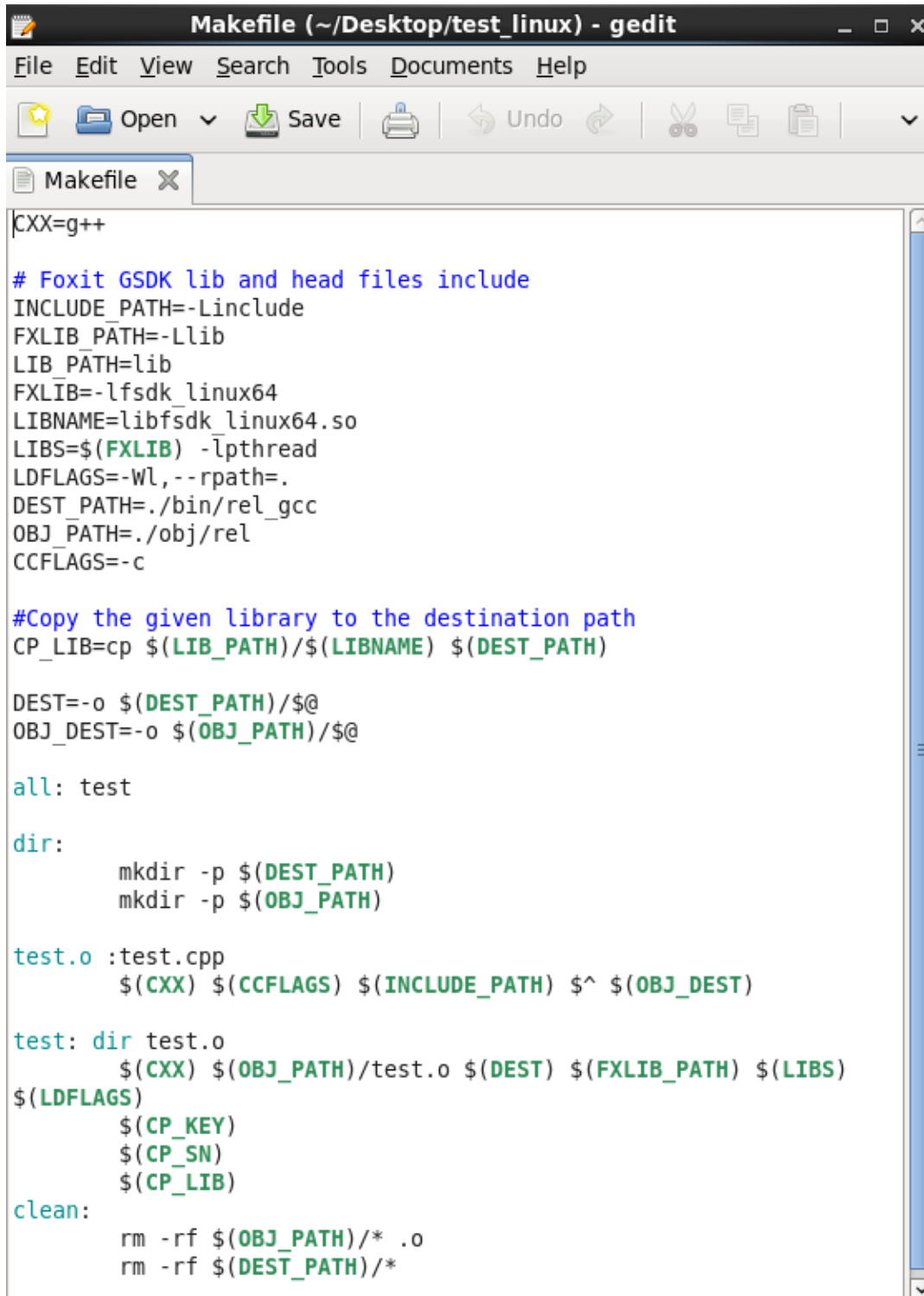
Figure 3-18

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 3-19.

22

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


```
CXX=g++

# Foxit GSDK lib and head files include
INCLUDE_PATH=-Linclude
FXLIB_PATH=-Llib
LIB_PATH=lib
FXLIB=-lfsdk_linux64
LIBNAME=libfsdk_linux64.so
LIBS=$(FXLIB) -lpthread
LDFLAGS=-Wl,--rpath=.
DEST_PATH=./bin/rel_gcc
OBJ_PATH=./obj/rel
CCFLAGS=-c

#Copy the given library to the destination path
CP_LIB=cp $(LIB_PATH)/$(LIBNAME) $(DEST_PATH)

DEST=-o $(DEST_PATH)/$@
OBJ_DEST=-o $(OBJ_PATH)/$@

all: test

dir:
        mkdir -p $(DEST_PATH)
        mkdir -p $(OBJ_PATH)

test.o :test.cpp
        $(CXX) $(CCFLAGS) $(INCLUDE_PATH) $^ $(OBJ_DEST)

test: dir test.o
        $(CXX) $(OBJ_PATH)/test.o $(DEST) $(FXLIB_PATH) $(LIBS)
$(LDFLAGS)
        $(CP_KEY)
        $(CP_SN)
        $(CP_LIB)
clean:
        rm -rf $(OBJ_PATH)/* .o
        rm -rf $(DEST_PATH)/*
```

Figure 3-19

23

3.4   Mac

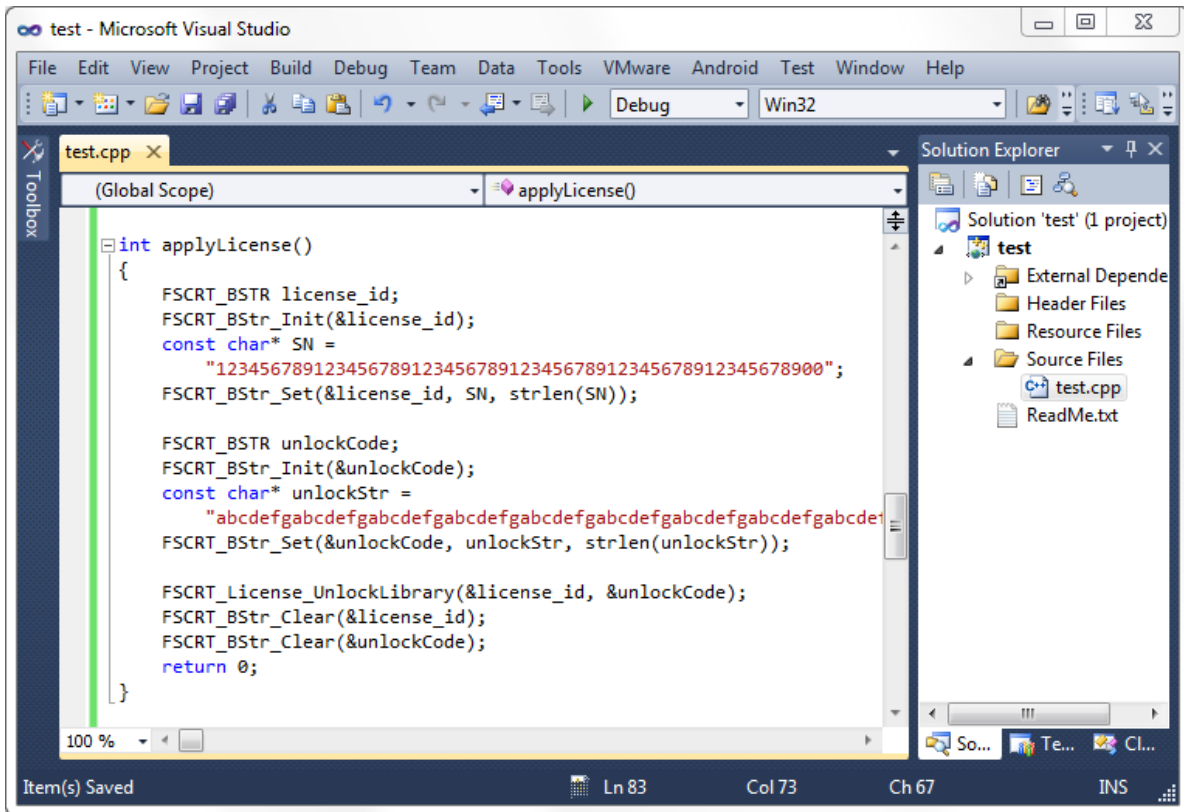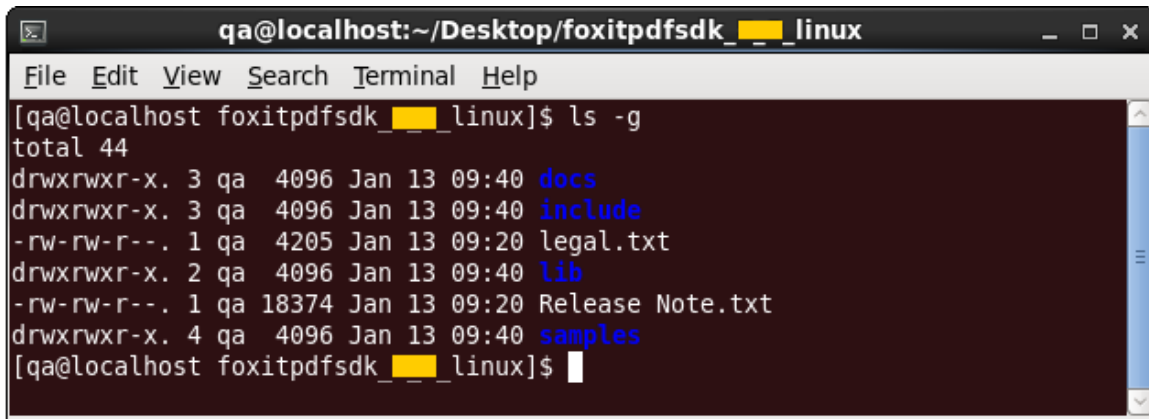### 3.4.1   What is in this package

Download Foxit PDF SDK zip for mac package and extract it to a new directory like
"foxitpdfsdk_5_3_mac". The structure of the release package is shown in Figure 3-20. This package
contains the following folders:

docs: API references, demo tutorial, developer guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos
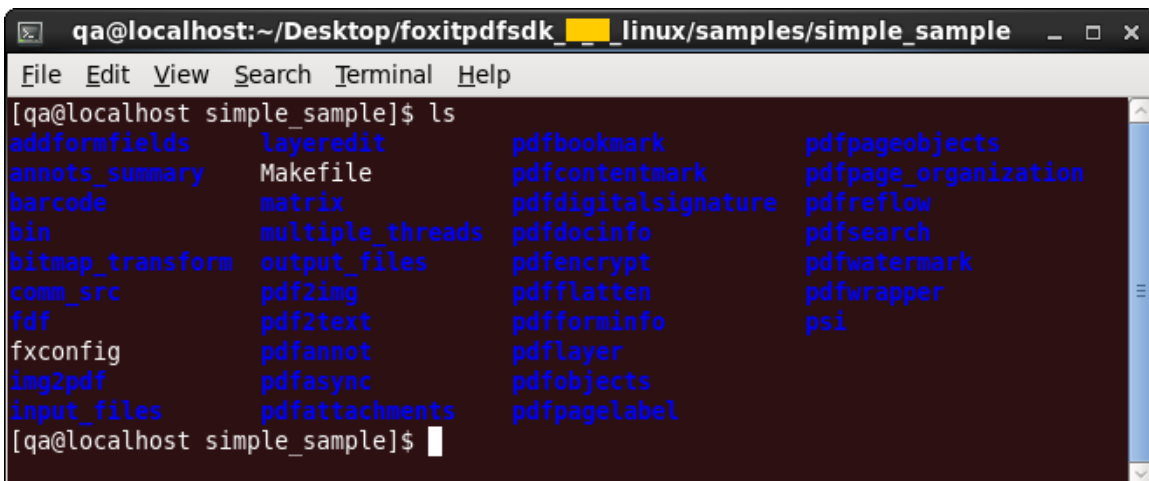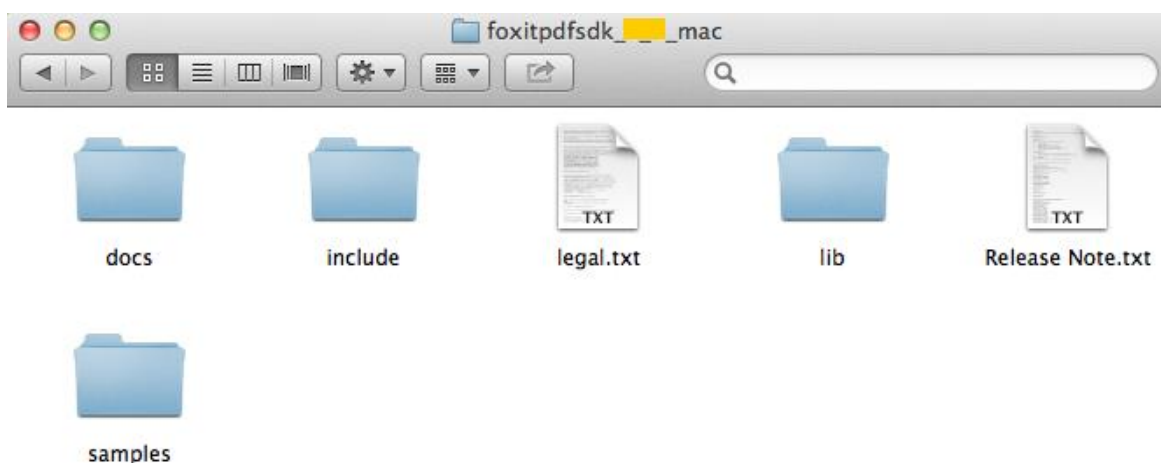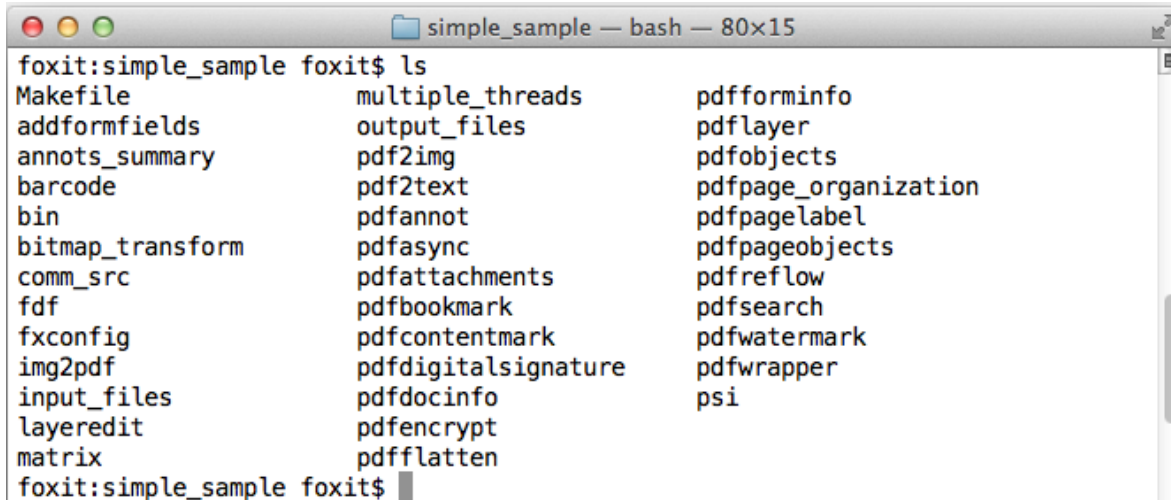

Figure 3-20

3.4.2   How to run a demo
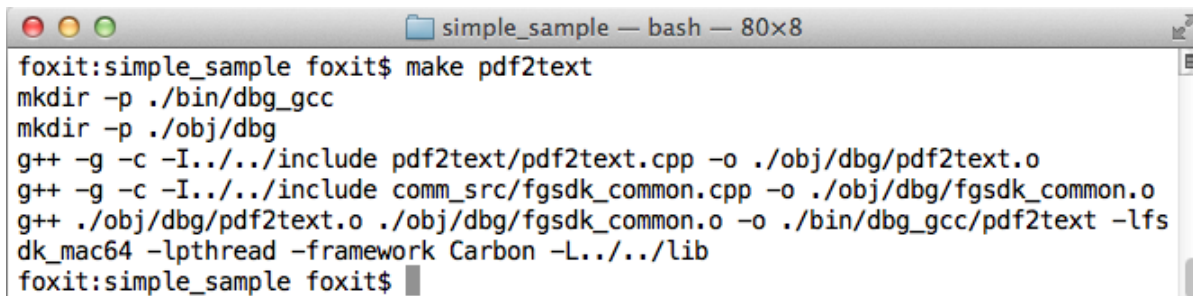
Foxit PDF SDK for mac provides more than 20 demo projects that cover a wide range of PDF document
application. These demos are in folder "samples/simple_samples" as shown in Figure 3-21.

Figure 3-21

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 3-22 shows the build process for pdf2text demo.

Figure 3-22

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. Navigate to the folder with the terminal, and run the binary file to get the demo running. Figure 3-23 shows the screen output when running pdf2text demo.

Figure 3-23

Some demos will generate output files (pdf, text or image files) to a folder named by the project name under "samples/pdfium_simple_sample/output_files/". In this example, output files are generated to "samples/pdfium_simple_sample/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 3-24.

Figure 3-24

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 3-25 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 3-25

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 3-26.

```
CXX=g++

# Foxit GSDK lib and head files include
INCLUDE_PATH=-Iinclude
FXLIB_PATH=-Llib
LIB_PATH=lib
FXLIB=-lfsdk_mac64
LIBS=$(FXLIB) -lpthread -framework CoreText -framework CoreGraphics -framework
    CoreFoundation

DEST_PATH=./bin/rel_gcc
OBJ_PATH=./obj/rel
CCFLAGS=-c



DEST=-o $(DEST_PATH)/$@
OBJ_DEST=-o $(OBJ_PATH)/$@

all: test

dir:
    mkdir -p $(DEST_PATH)
    mkdir -p $(OBJ_PATH)

test.o :test.cpp
    $(CXX) $(CCFLAGS) $(INCLUDE_PATH) $^ $(OBJ_DEST)

test: dir test.o
    $(CXX) $(OBJ_PATH)/test.o $(DEST) $(FXLIB_PATH) $(LIBS) $(LDFLAGS)
    $(CP_KEY)
    $(CP_SN)

clean:
    rm -rf $(OBJ_PATH)/* .o
```

Figure 3-26

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 is in the package

Download Foxit PDF SDK zip for iOS package and extract it to a new directory "foxitpdfsdk_5_3_ios".The structure of the release package is shown in Figure 3-27. This package contains the following folders:

Figure 3-27

docs: API references, demo tutorial, 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_sample" contains four demos that realize a simple PDF viewer, an OOM recovery mechanism, a form filling including importing/exporting FDF file, and a simple PDF viewer on Xamarin Studio 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 3-28 .

Figure 3-28

 "samples/view_sample" contains a viewer demo, an OOM demo, a form filling demo, and a Xamarin viewer demo which are shown in Figure 3-29.
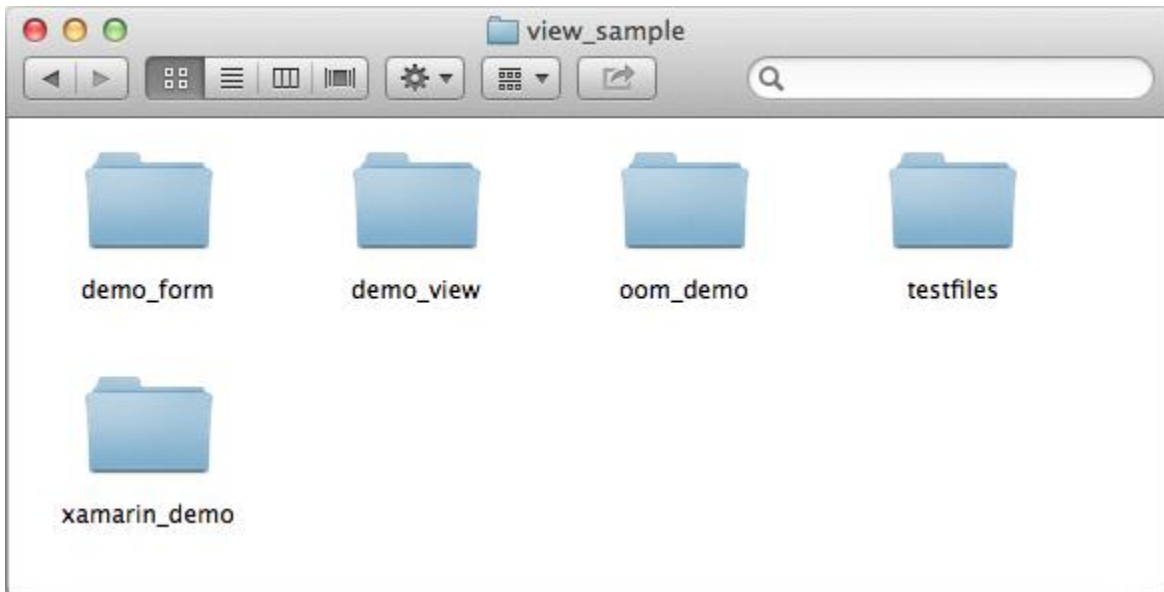
Figure 3-29

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 3-30. Here the Xcode version is 5.0.2.

Note: *The "pdfdigitalsignature" demo relies on the third-party library OpenSSL, so please make sure that you have already installed OpenSSL1.0.0q or later before running the simple samples.*

Here, you should put openssl package into "foxitpdfsdk_5_3_ios/lib" folder. The default paths of Xcode configure configuration are "$(SRCROOT)/../../../lib/ssl/lib" and "$(SRCROOT)/../../../lib/ssl/include". If you change the openssl package path, please update the Xcode configuration.
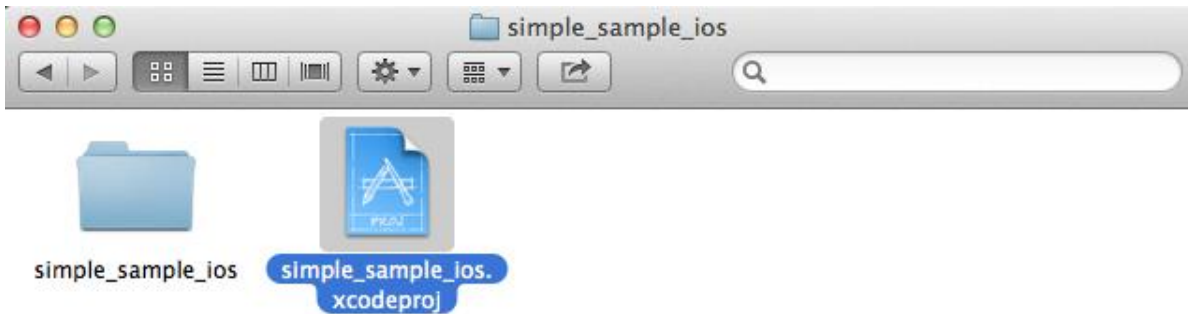
Figure 3-30

After loading the solution, click on "Run" on the menu bar to build the solution. A screenshot of the project is shown in Figure 3-31. The output files for all demos are put in "**sandbox**".
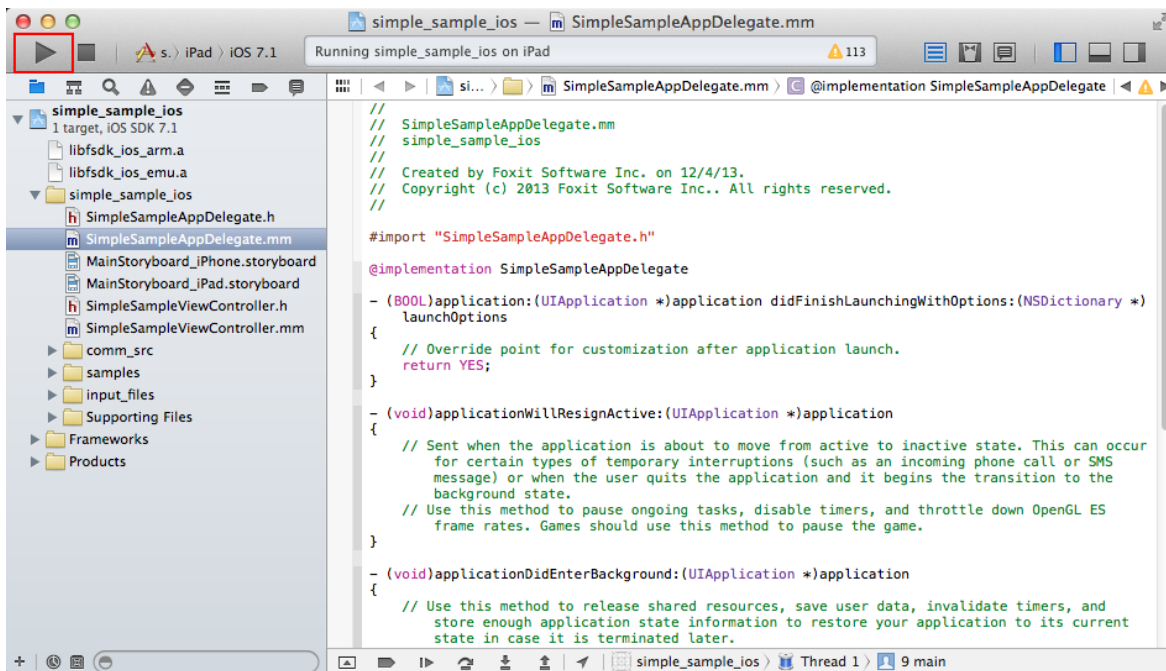
Figure 3-31

After Building the solution successfully, the iOS simulator will be started as shown in Figure 3-32.

Figure 3-32

For example, click on "pdf2text", the log information will be shown in Figure 3-33.

Figure 3-33

PDF View Samples

View samples contain a demo to realize a PDF viewer under "samples/view_sample/demo_view", a demo to show the OOM recovery mechanism under "samples/view_sample/oom_demo", a demo to fill a form including importing/exporting FDF file under "samples/view_sample/demo_form", and a demo to show how to run a simple PDF viewer in Xamarin Studio under "samples/view_sample/xamarin_demo".

1) demo_view

The "demo_view" under "samples/view_sample/demo_view" is a viewer demo illustrating how to realize a PDF viewer on Xcode. To run it in Xcode, follow the steps below:

a) Open "samples/view_sample/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 "AboutFoxit.pdf" under "samples/view_sample/testfiles" folder will be shown like Figure 3-34.

Figure 3-34

c) Click the "AboutFoxit.PDF", the first page of the PDF will be displayed. Click the small triangle menu to show the function buttons as shown in Figure 3-35.
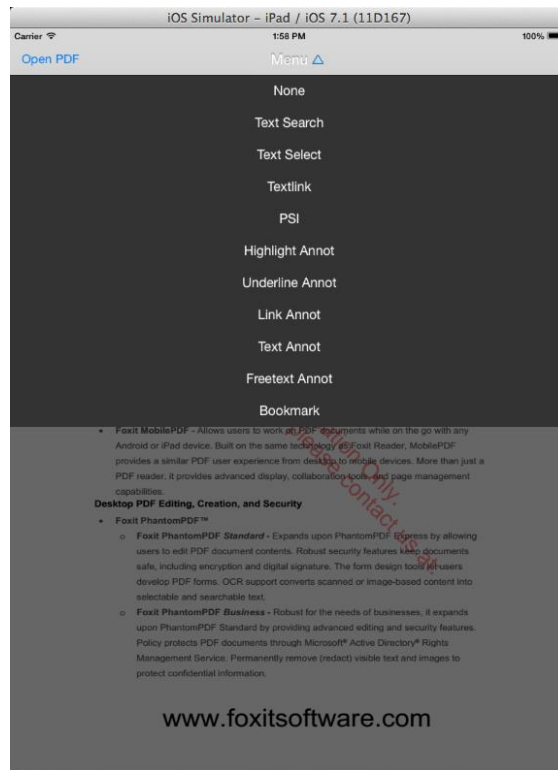
Figure 3-35

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 3-36.
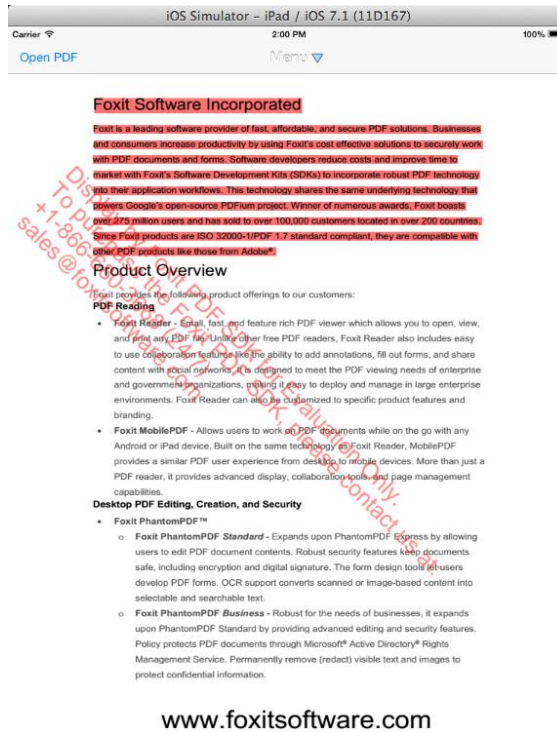
Figure 3-36

2) oom_demo

The "oom_demo" under "samples/view_sample/oom_demo" is used for showing the OOM recovery mechanism. To run it in Xcode, follow the steps below:

a) Open "samples/view_sample/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 shown like Figure 3-37.
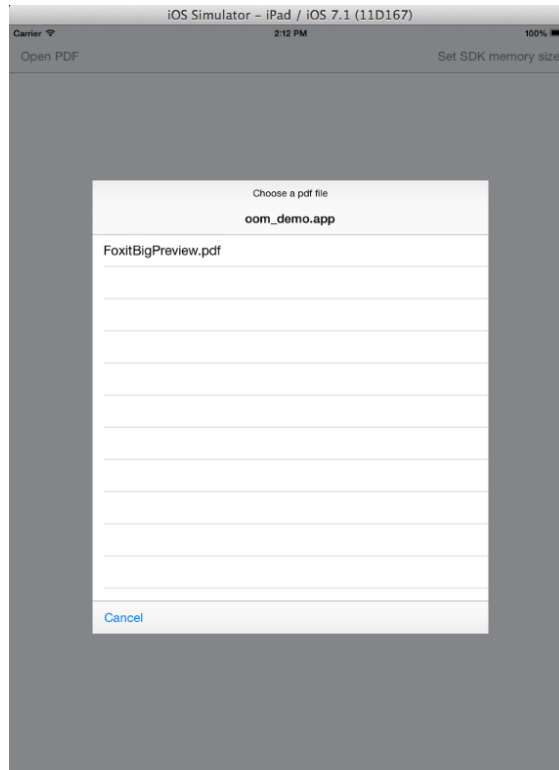
Figure 3-37

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 the memory with 4M bytes. This is shown in Figure 3-38.
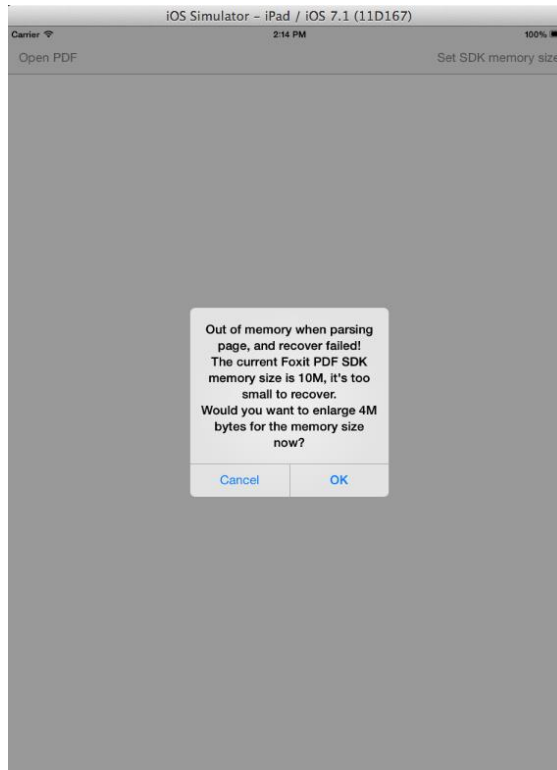
Figure 3-38

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 the memory with 4M bytes, which is shown in Figure 3-39.
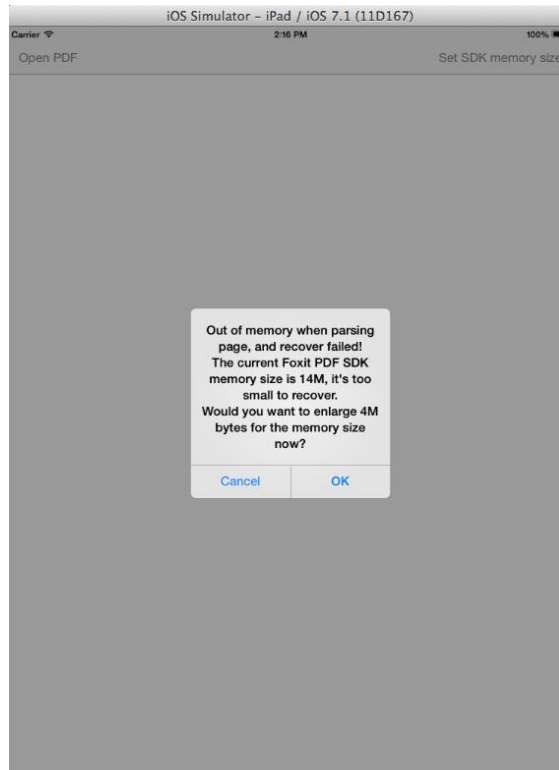
Figure 3-39

e) If you also click on "OK", the pages of the "FoxitBigPreview.pdf" file will be displayed as shown in Figure 3-40. 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 bytes.
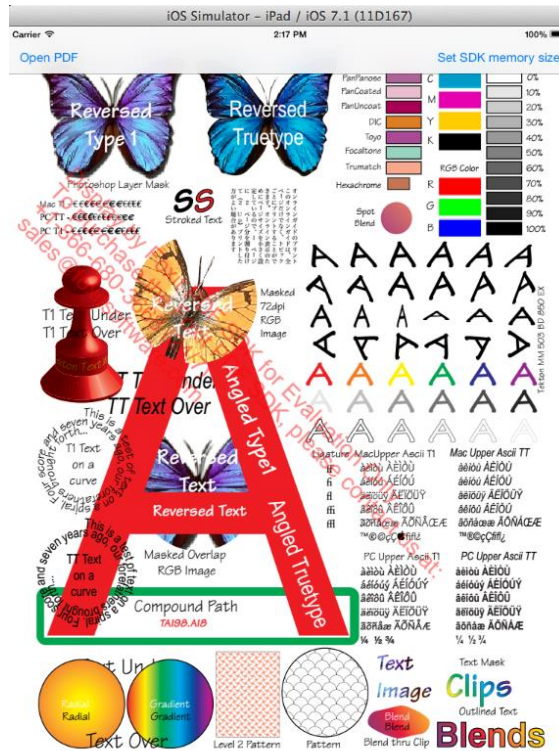
Figure 3-40

f) In Figure 3-38, 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 3-41 shows that setting the memory size to 20M. After clicking on "OK", you can see the same result with Figure 3-40.
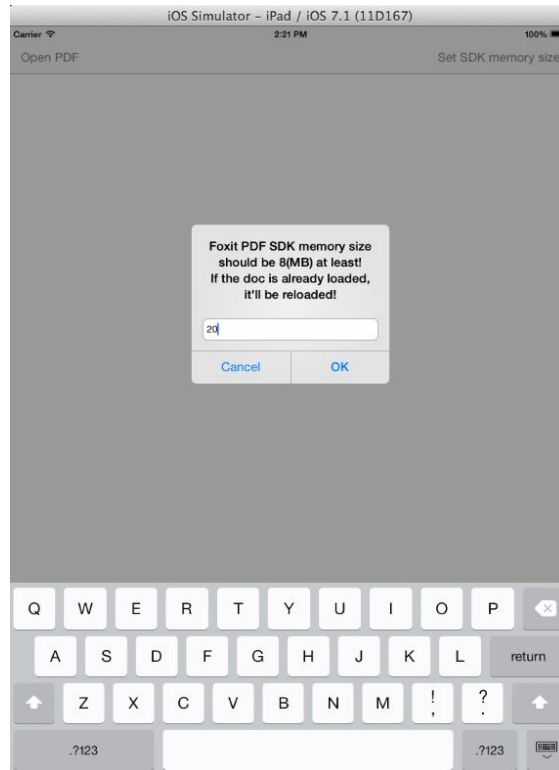
Figure 3-41

3) demo_form

The "demo_form" under "samples/view_sample/demo_form" is a form demo illustrating how to fill a form including importing/exporting FDF file. To run it in Xcode, follow the steps below:

a) Open "samples\view_sample\demo_form\demo_form.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 "FoxitForm.pdf" under "samples/view_demo/testfiles" folder will be shown as Figure 3-42.
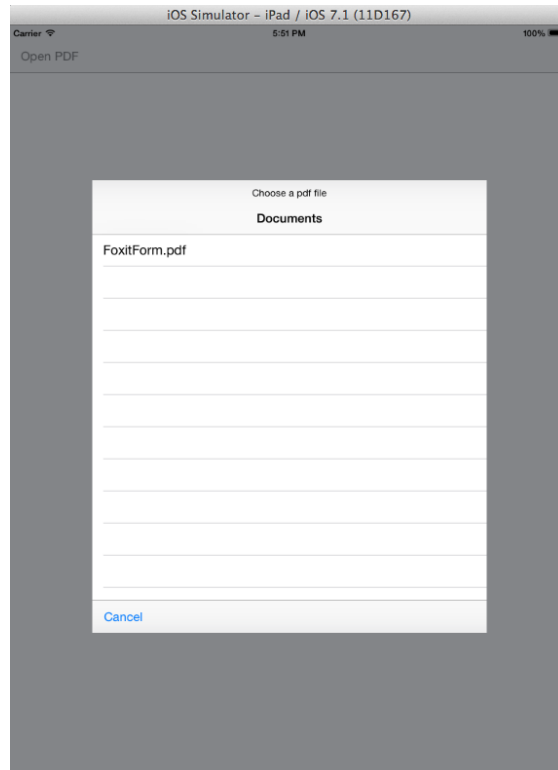
Figure 3-42

c) Click the "FoxitForm.pdf", and the PDF file will be shown as Figure 3-43
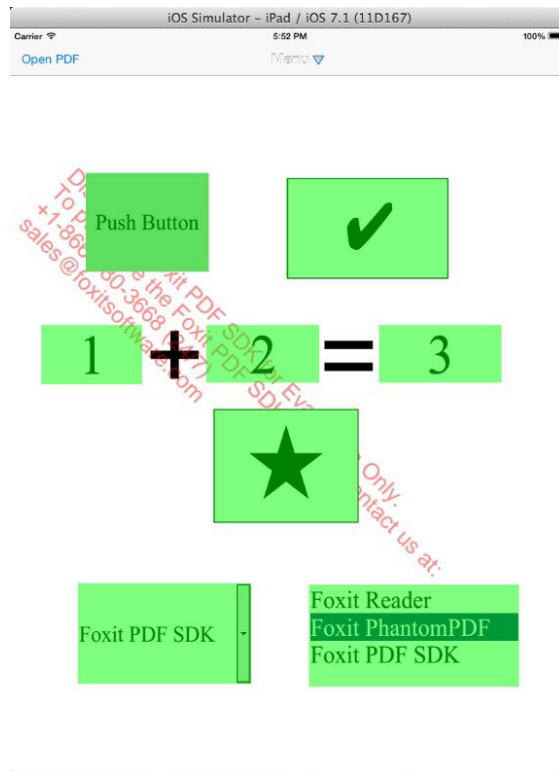
Figure 3-43

d) Fill the form, for example, like Figure 3-44. Press "menu" to show the operation menu as shown in Figure 3-45.
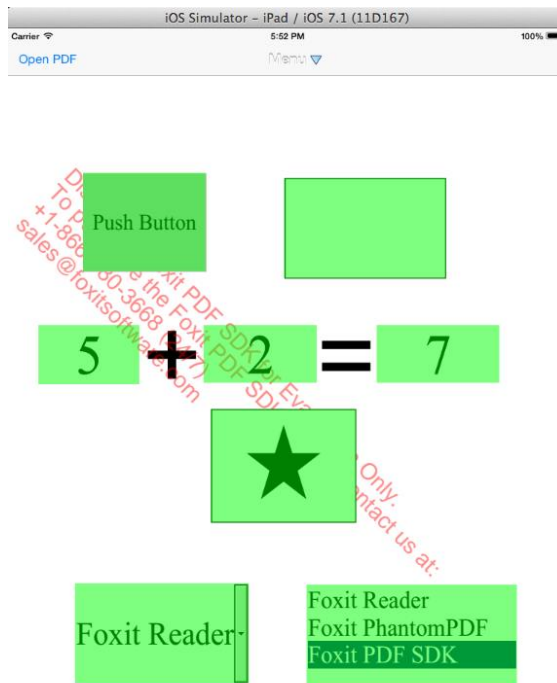
Figure 3-44
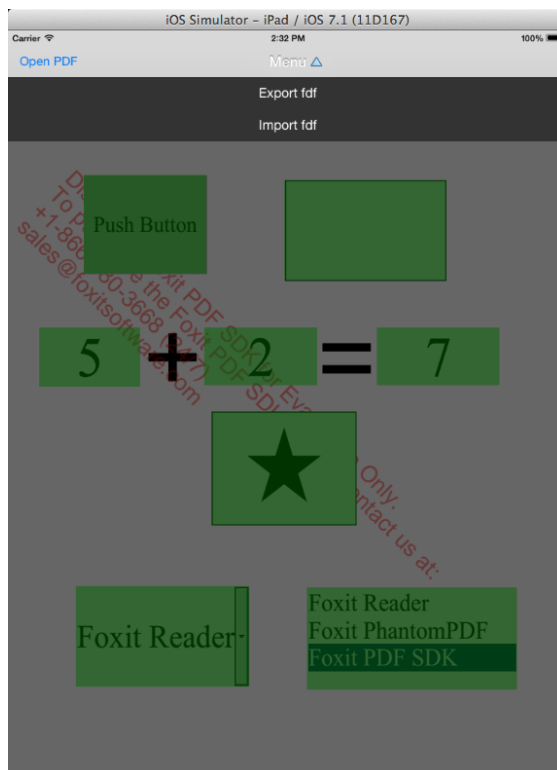
Figure 3-45

For example, select "Export fdf", click "New" like Figure 3-46. Input the exported name "formfill.fdf" as shown in Figure 3-47, and then click "OK" to export the form data to "formfill.fdf" file.
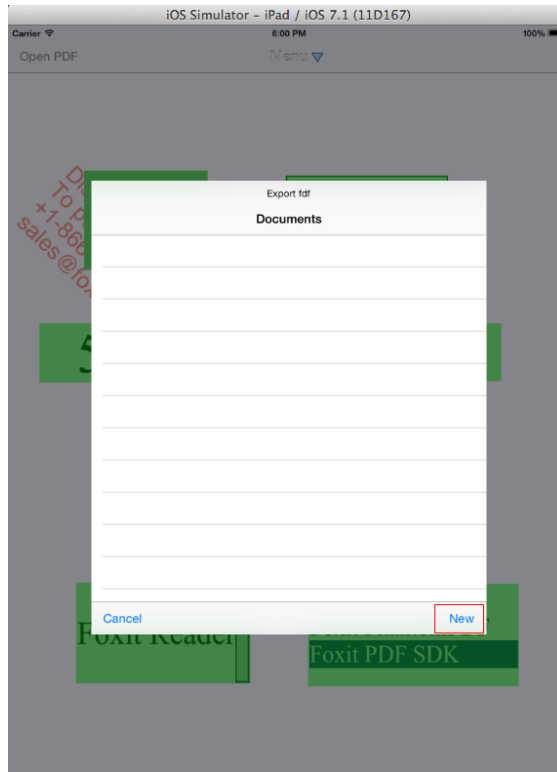
Figure 3-46

46
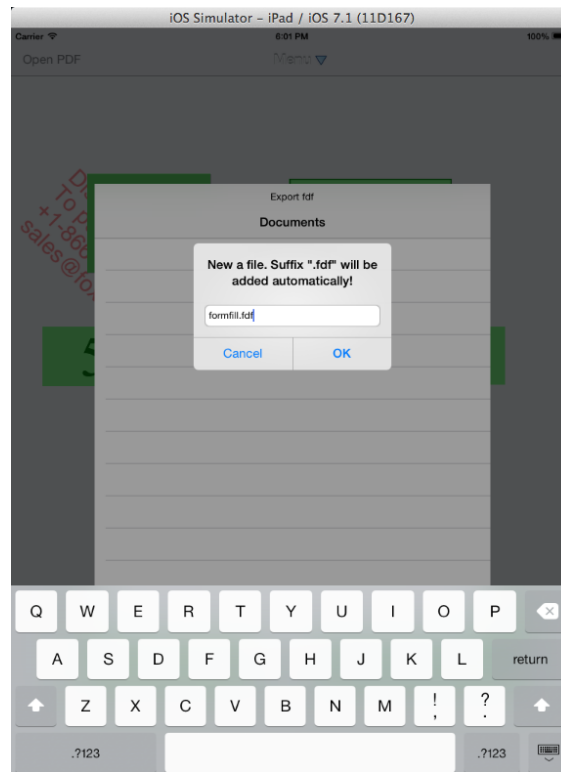
Figure 3-47

4) xamarin_demo

The "xamarin_demo" under "samples/view_sample/xamarin_demo" is a simple viewer demo illustrating how to run a simple PDF viewer in Xamarin Studio. To run it in Xamarin Studio, follow the steps below: (in this guide, we run the demo on iPhone 5 with iOS 8)

a) Import the project into Xamarin Studio following "File -> Open…", and choose the "sdk_demo.sln" under "samples\xamarin_demo" folder. The directory structure of the demo will be like Figure 3-48.
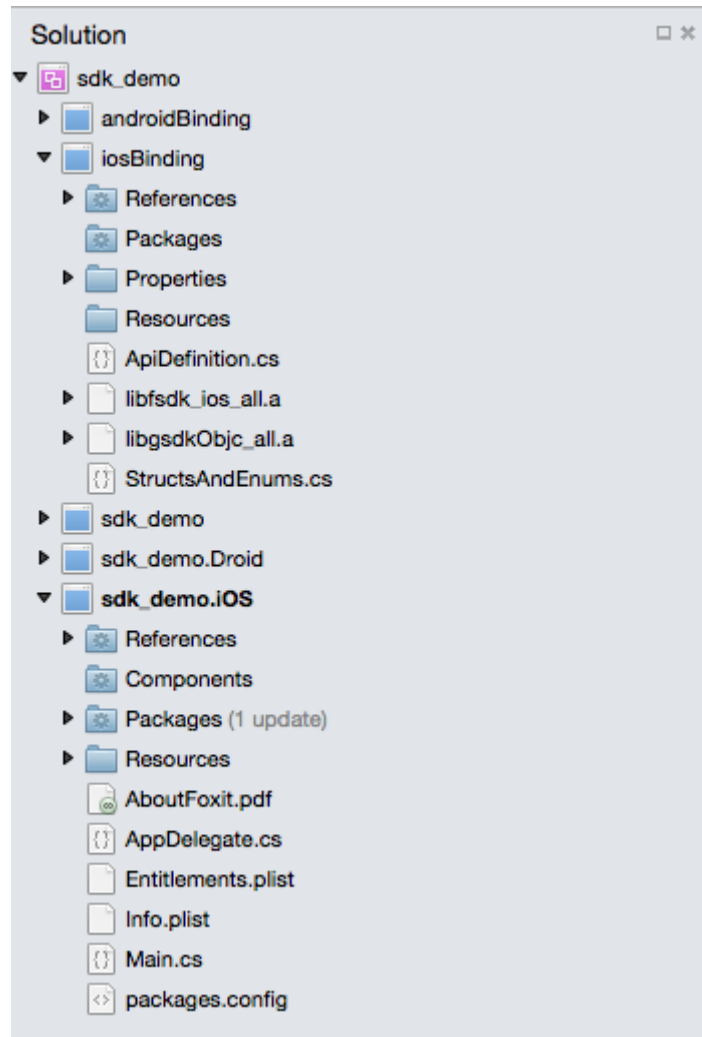
Figure 3-48

Note: *This demo includes the project codes of both iOS platform and Android platform, but the downloaded Foxit PDF SDK for iOS package only contains the SDK library for iOS to run the Xamarin iOS project "sdk_demo.iOS" on Xamarin Studio. If you want to run the Xamarin Android project "sdk_demo.Droid", please put the SDK library for Android to the corresponding directory.*

b) Right click the project "sdk_demo.iOS" and select "Set As Startup Project" as shown in Figure 3-49.
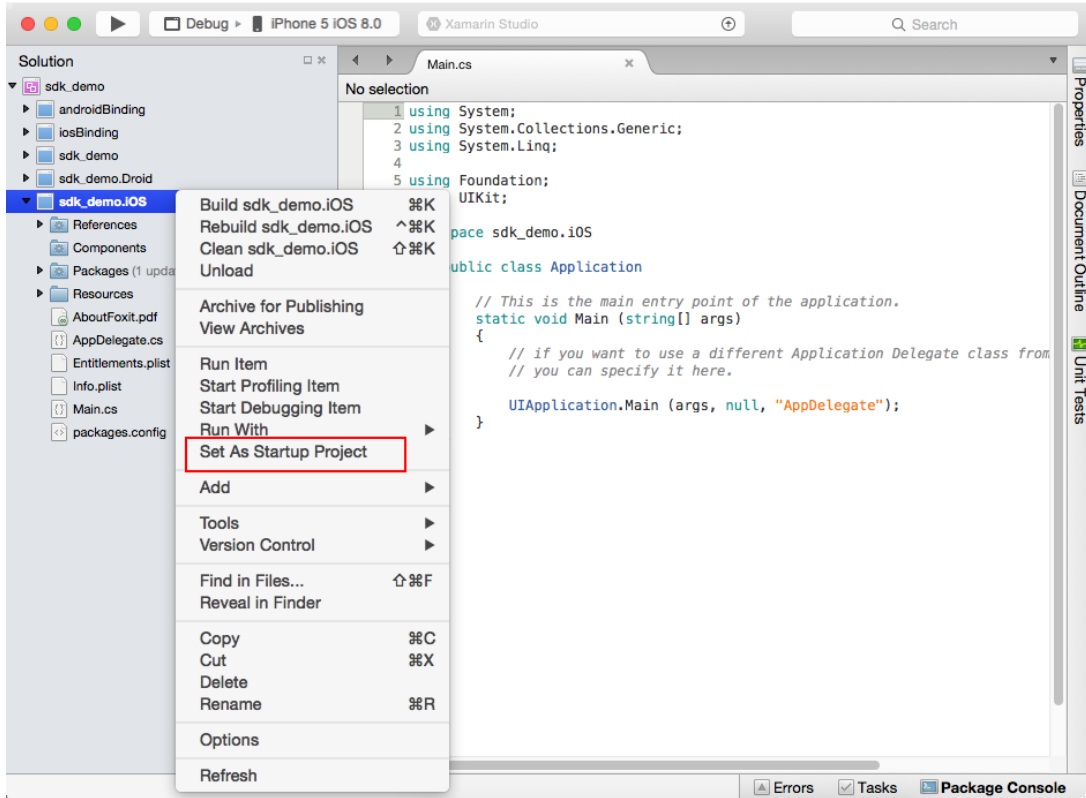
Figure 3-49

c) Click on ▶ button or "Run -> Start Debugging" to run the demo. After the iOS simulator starts, the "AboutFoxit.pdf" file under "samples/view_sample/testfiles" folder will be displayed as shown in Figure 3-50.

Note: *The "iosBinding" project used for binding "libfsdk_ios_all.a" and "libgsdkObjc_all.a" libraries will be compiled automatically first when running the "sdk_demo.iOS" project.*

This demo combines the "libfsdk_ios_arm.a" (for devices) and "libfsdk_ios_emu.a" (for emulators) static libraries under "lib" folder of the released package into one library "libfsdk_ios_all.a".

The "libgsdkObjc_all.a" library is a wrapped library which includes the devices and emulators libraries generated by the project "gsdkObjc" under "samples/view_sample/xamarin_demo" folder. The "gsdkObjc" project is used for wrapping the SDK C APIs to Object-C APIs and generating the static library which can be used in Xamarin Environment. It only wraps the APIs required by the demo. If you need more APIs, you can wrap them by yourself referring to the "gsdkObjc.mm" file under "samples/view_sample/xamarin_demo/gsdkObjc/gsdkObjc" folder.
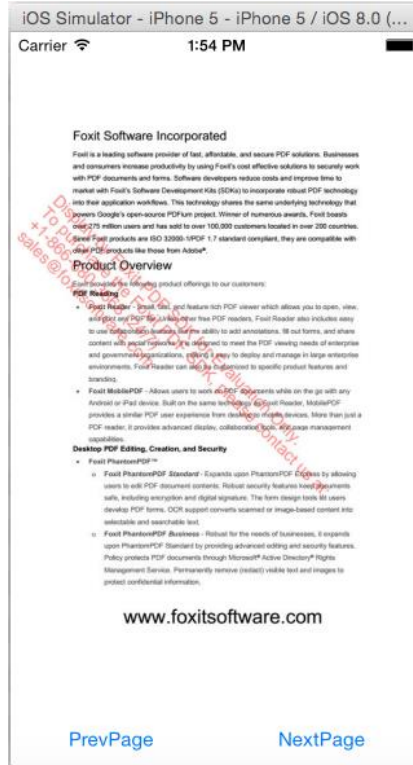
Figure 3-50

d) The simple viewer demo only provides page turning feature. Click on "NextPage" button to turn to the next page as shown in Figure 3-51.

Figure 3-51

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 3-52.

Figure 3-52

51
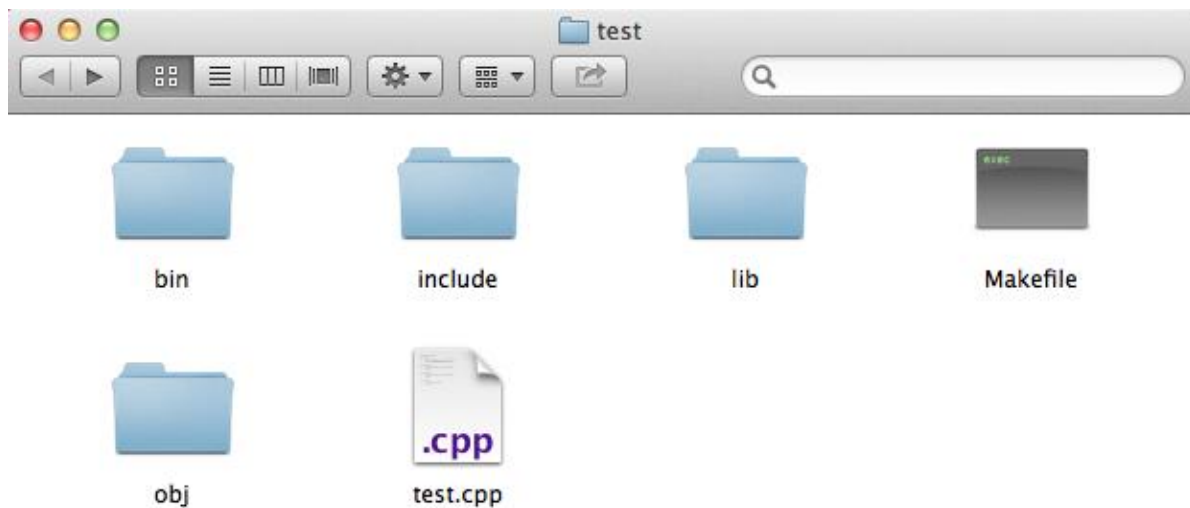
a) Copy "include" and "lib" folders from the PDF SDK package to "test".

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 3-53.

d) The right part of the Figure 3-53 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!
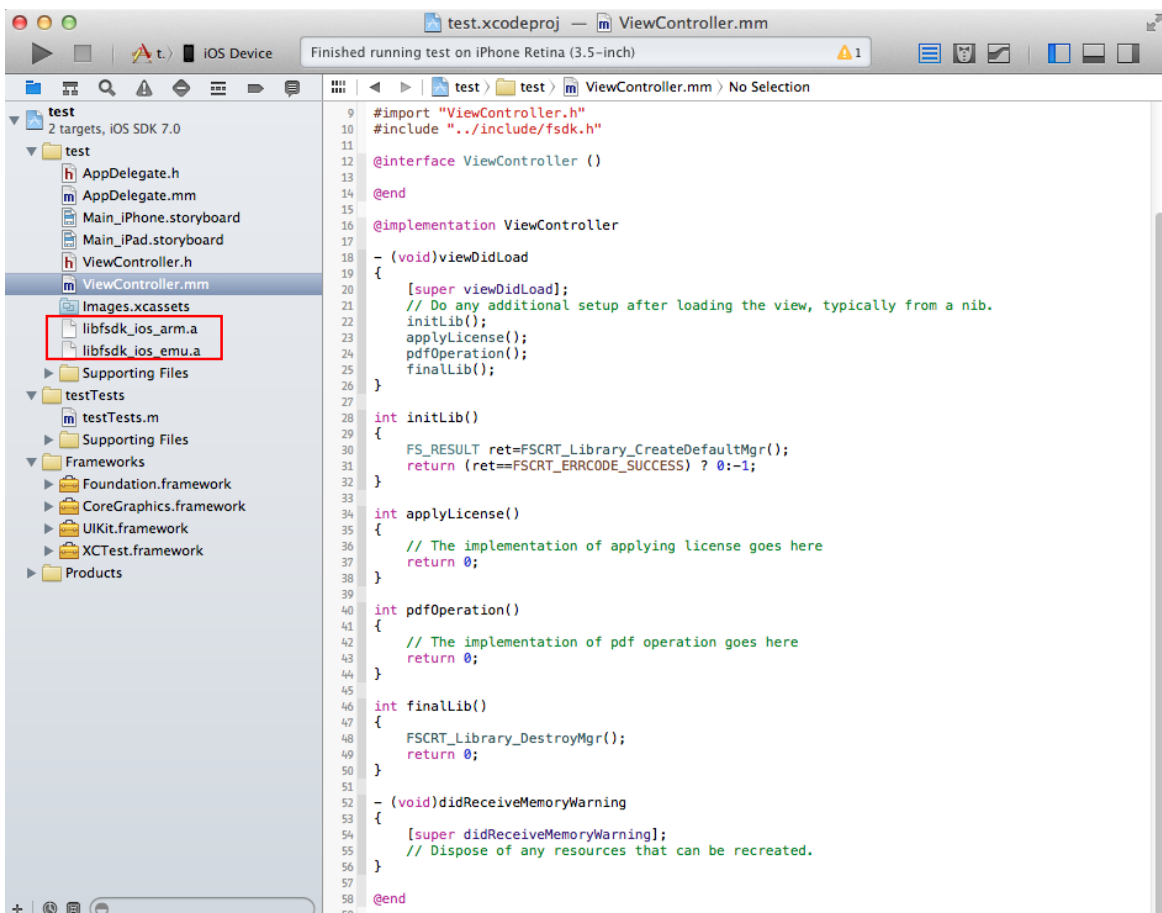
Figure 3-53

3.6 Android

3.6.1 What is in the package

Download Foxit PDF SDK zip for Android C API package and extract it to a new directory "foxitpdfsdk_5_3_android_c".The structure of the release package is shown in Figure 3-54. This package contains the following folders:
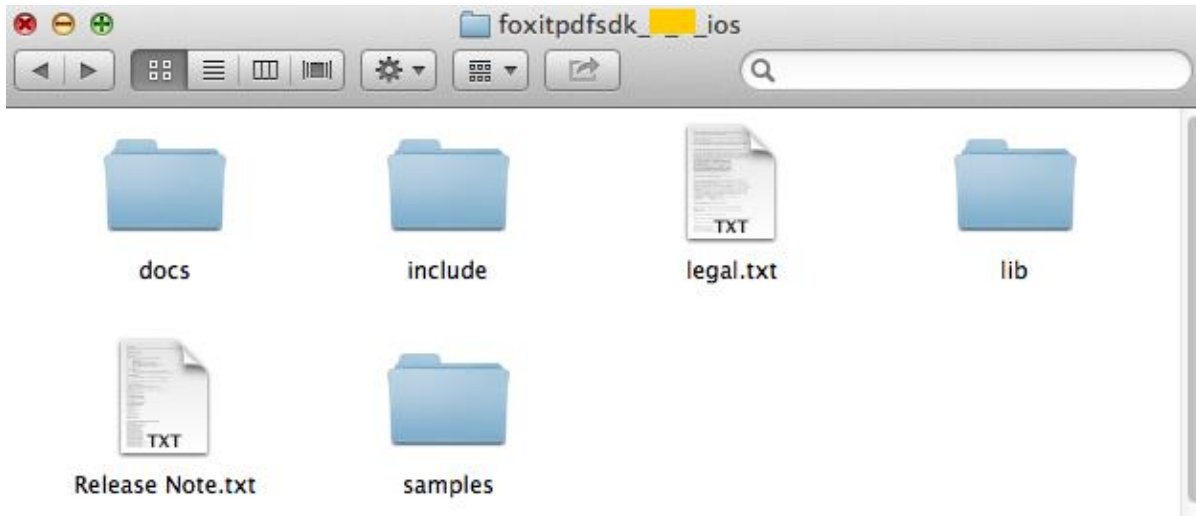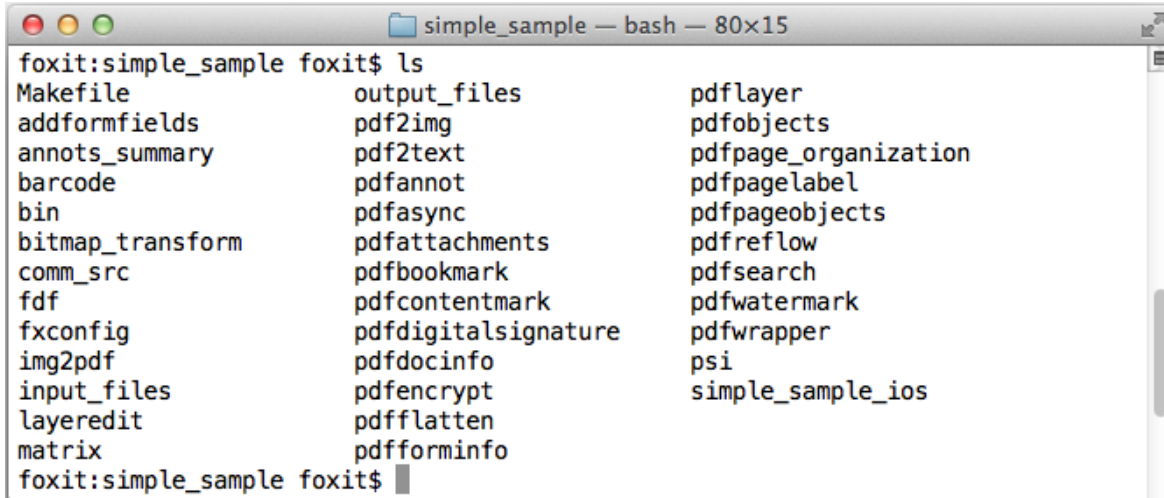
docs: API references, developer guide

include: header files for foxit pdf sdk API

libs: libraries and license files

samples: sample projects and demos
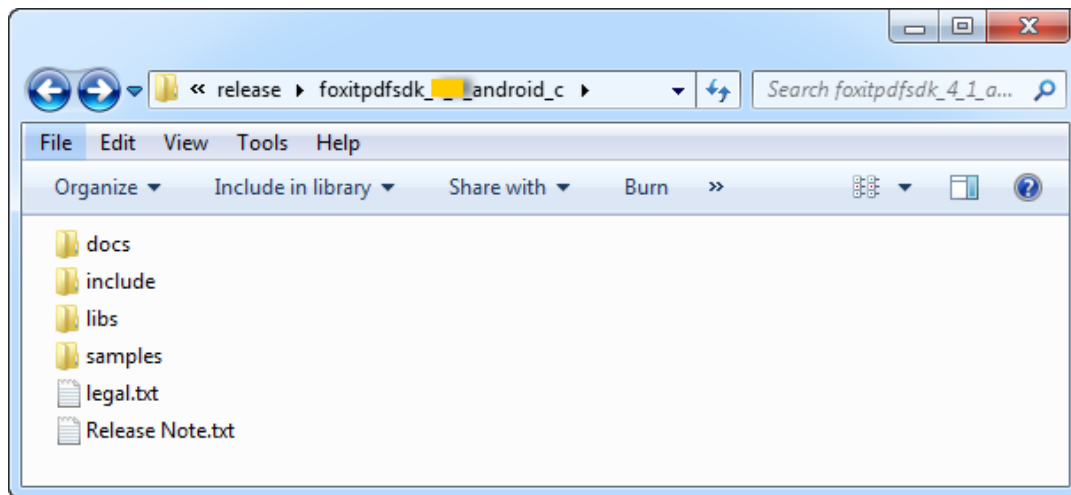
Figure 3-54

3.6.2 How to run a demo

Foxit PDF SDK for Android C API provides one viewer demo in folder "samples". The resources and files of this demo are put under "samples/ViewerDemo" as shown in Figure 3-55.
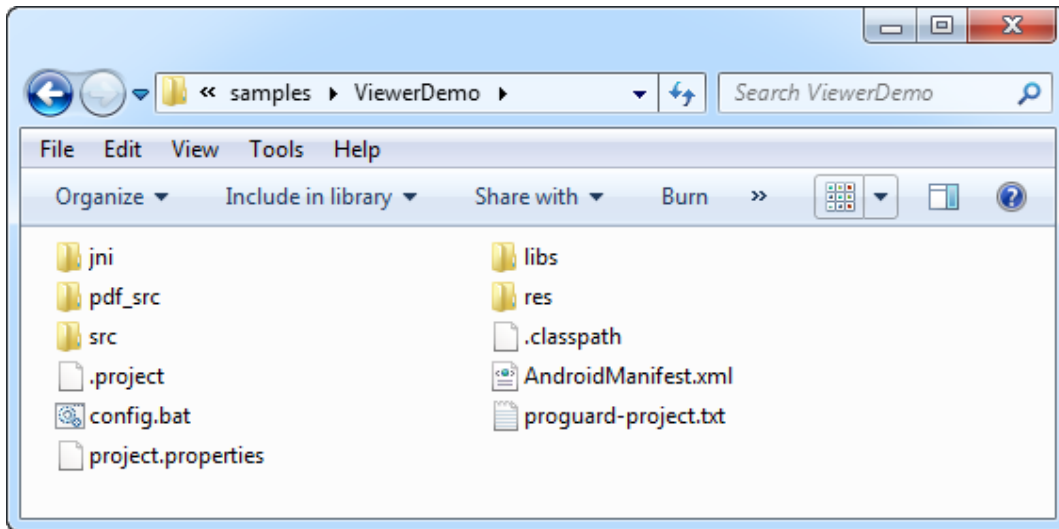
Figure 3-55
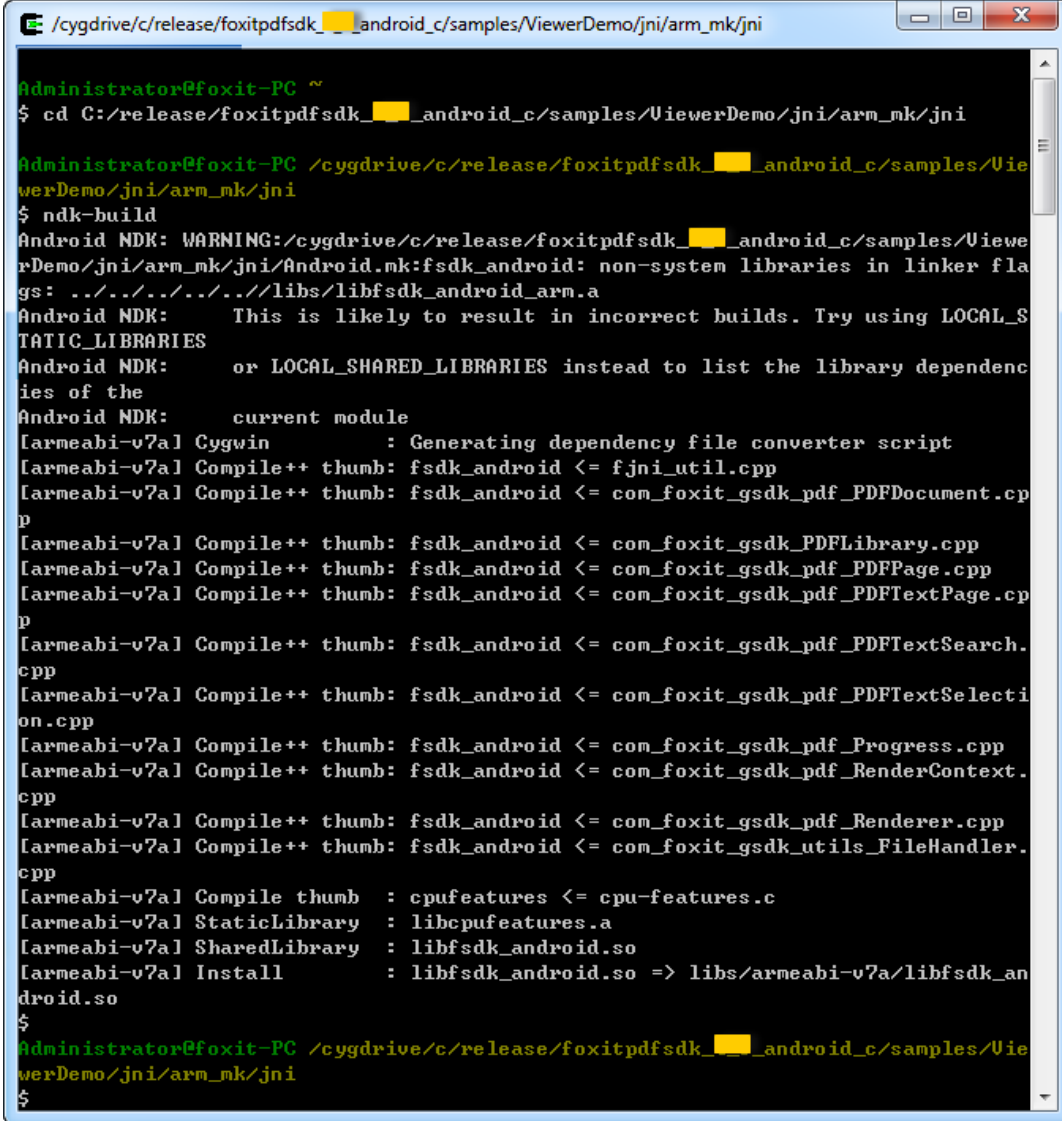
To run the demo of C API on Android platform, the JNI (Java Native Interface) is playing an important role. JNI can facilitate users to develop applications on Android platform with native-code languages such as C and C++. To use JNI, please download and install Android NDK (Native Development Kit) at 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_5_3_android_c/libs" folder and some other ".cpp" files under "/samples/ViewerDemo/jni" folder. If your operating system is x86 or arm64, you can build the "libfsdk_android_x86.a" or "libfsdk_android_arm_64.a" library under "/foxitpdfsdk_5_3_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 3-56.

Figure 3-56

d) Launch Eclipse, import the "ViewerDemo" project. Click on "Run->Run as->Android Application" to run the demo, here we assume that we have already created an AVD targeting Android 4.4.2 and pushed a PDF file "AboutFoxit.pdf" on this device. Figure 3-57 shows the demo.
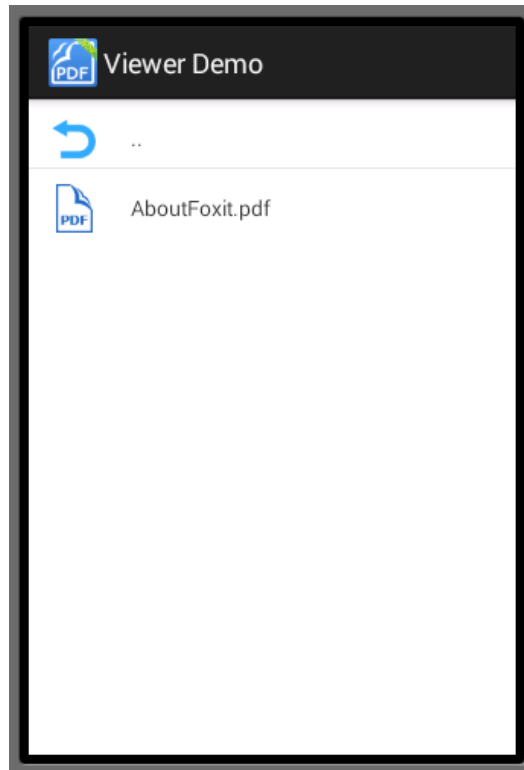
Figure 3-57

e) Click the "AboutFoxit.pdf", the PDF file will be displayed as shown in Figure 3-58.

Figure 3-58

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 3-59.
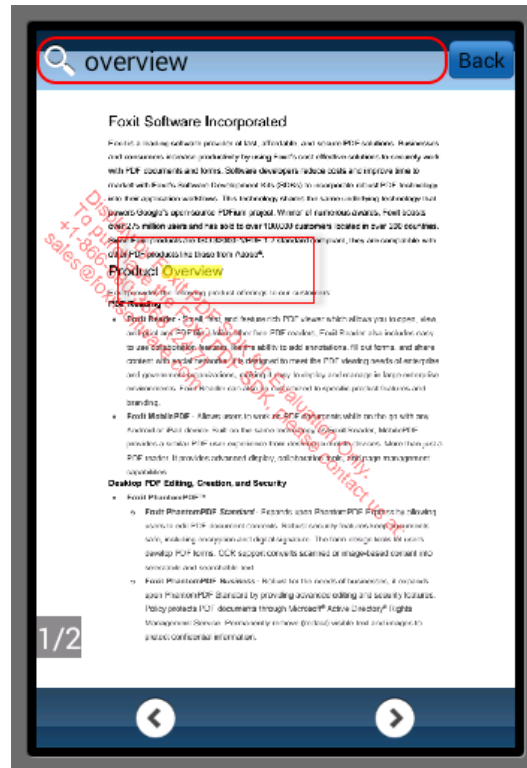
Figure 3-59

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 3-60.
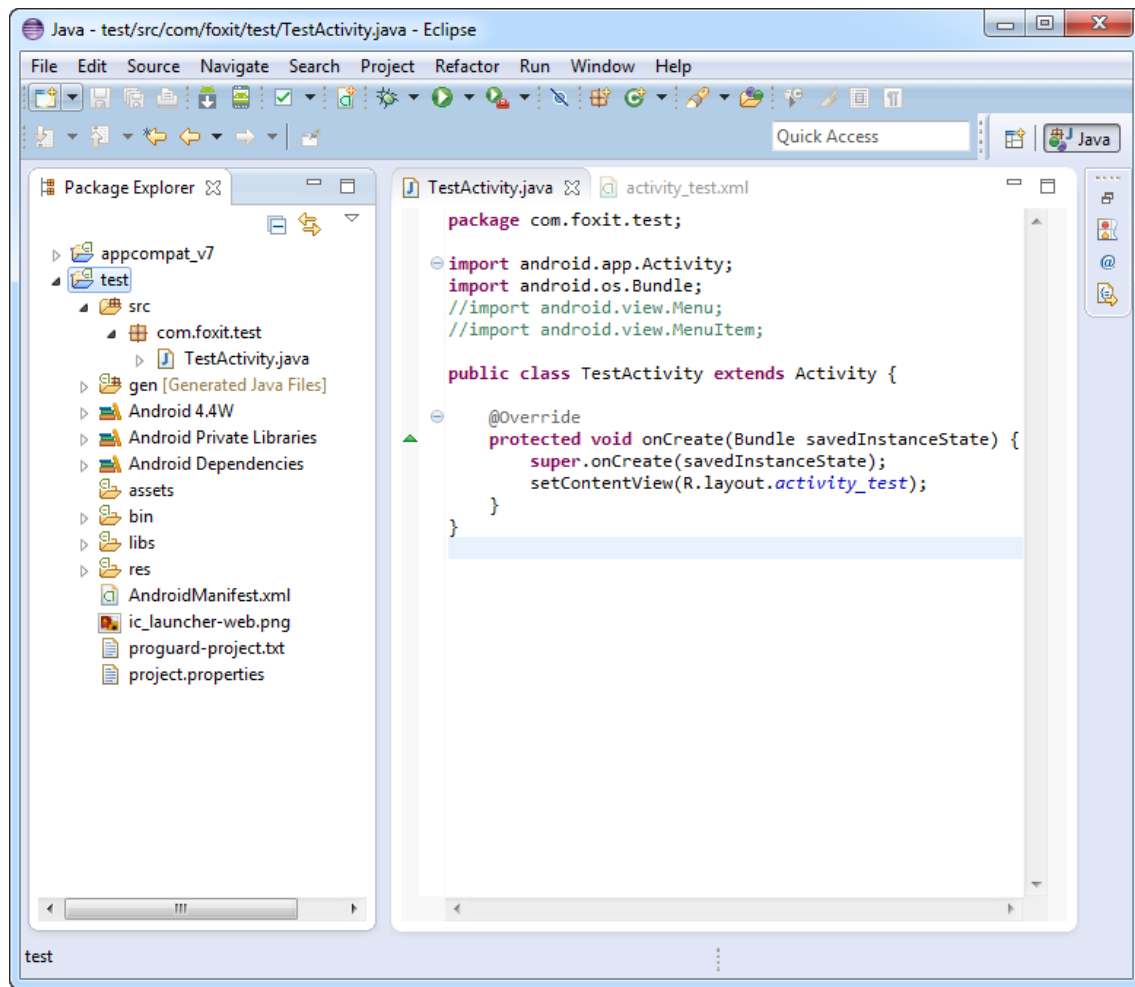
Figure 3-60

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 build 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 3-61.
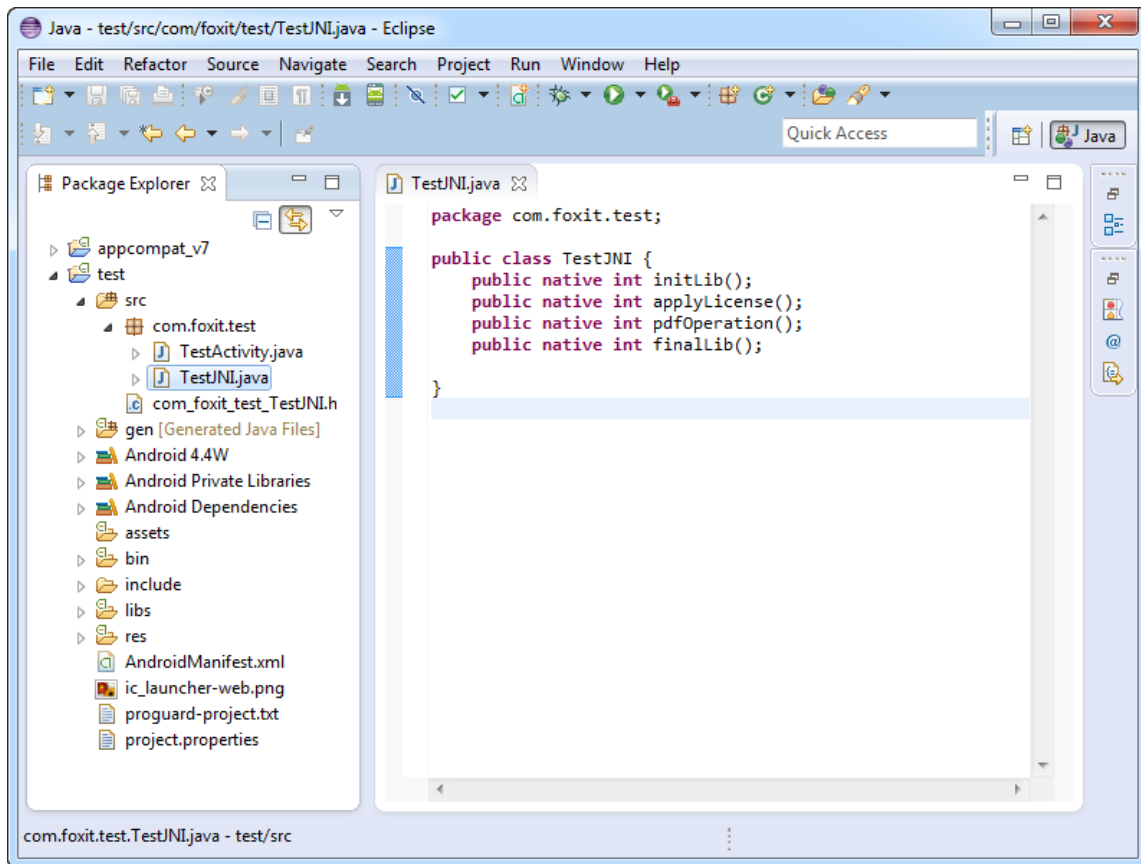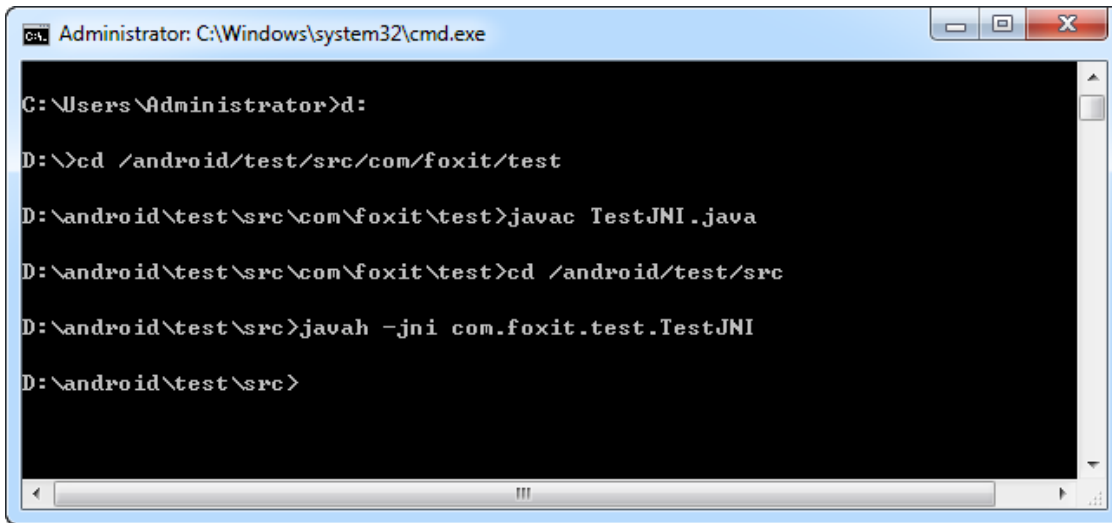
Figure 3-61

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 3-62. Now, delete the "TestJNI.class" file in "test/src/com/foxit/test".

Figure 3-62

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 3-63.

Figure 3-63

61

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 3-64 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.
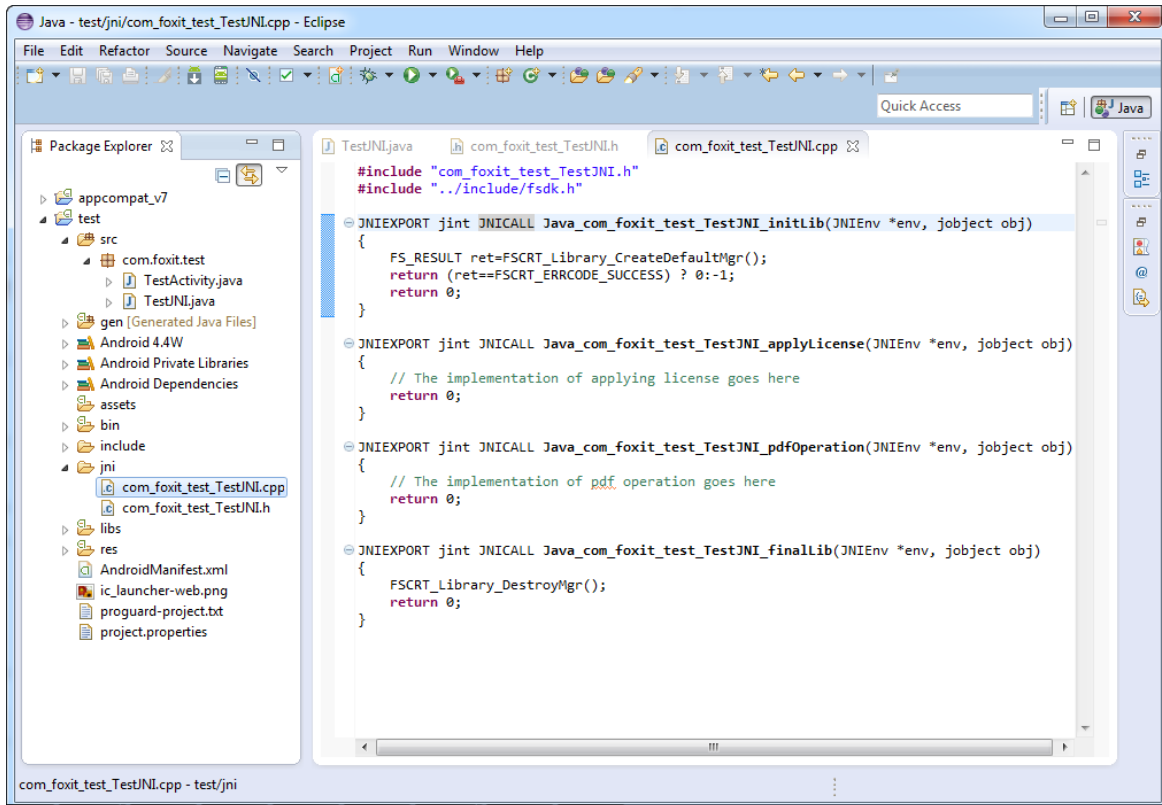
Figure 3-64

f) Create two ".mk" files called Android and Application in "test/jni", which are used for compiling and building ".so" library. Open the "Android.mk" and "Application.mk" files, input the contents referring to the Figure 3-65 and Figure 3-66. Here, we assume that the mobile or virtual device is armv7-a (neon), so compile the "libfsdk_android_arm.a" library in "test/libs".
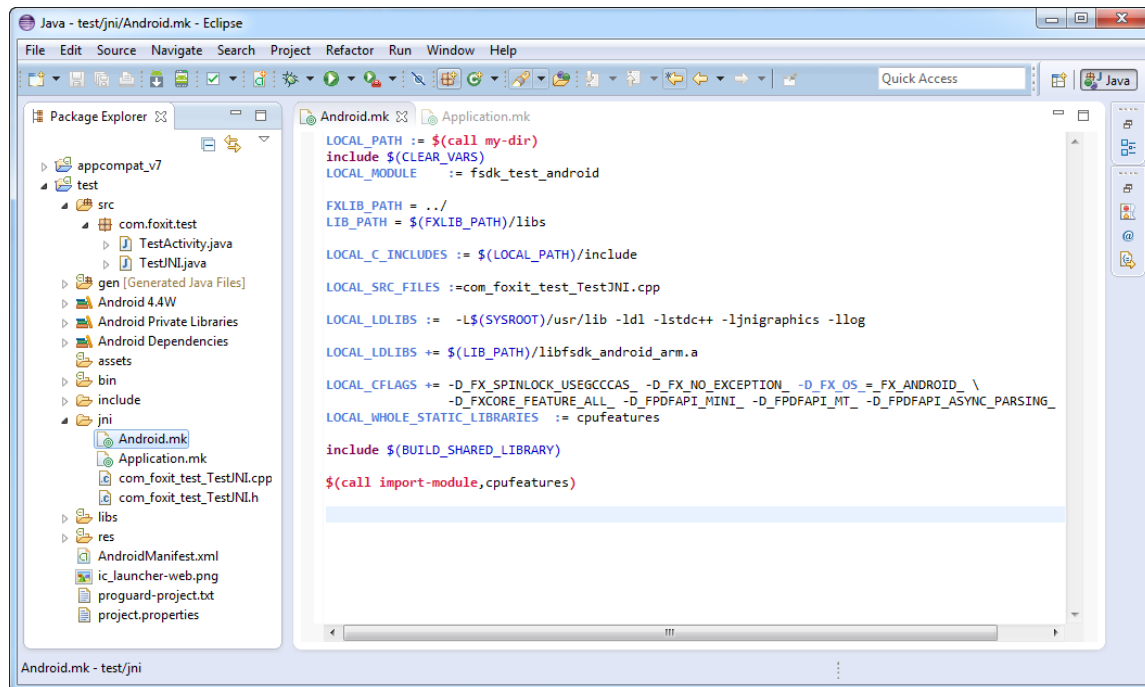
Figure 3-65
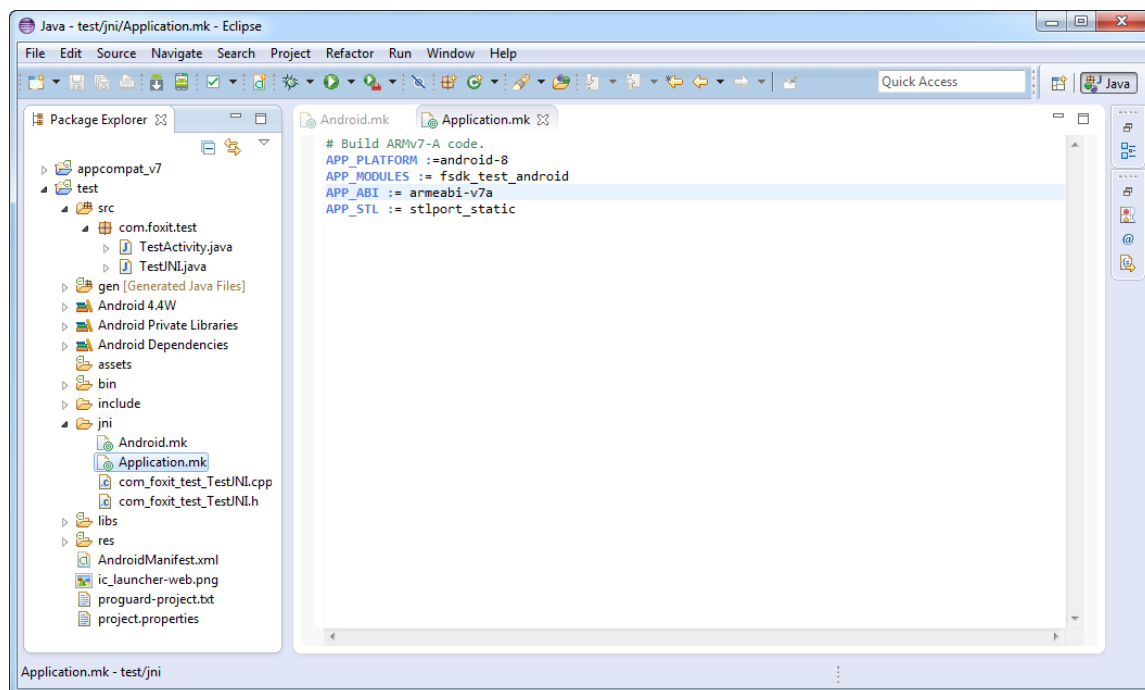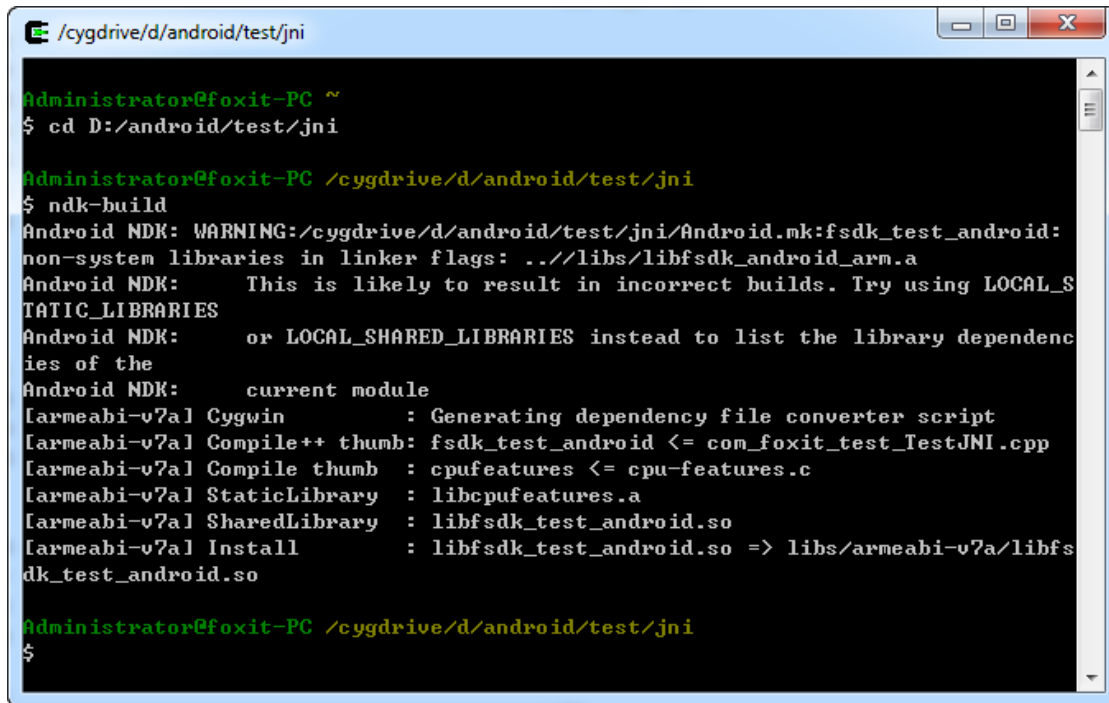
Figure 3-66

g) Open Cygwin, go to "test/jni", run "ndk-build", and the "libfsdk_test_android.so" library will be generated in "test/libs/ armeabi-v7a". This is shown in Figure 3-67.

63

Figure 3-67

h) Call the methods in "libfsdk_test_android.so" library. Open the "TestActivity.java", input the contents referring to Figure 3-68. 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.
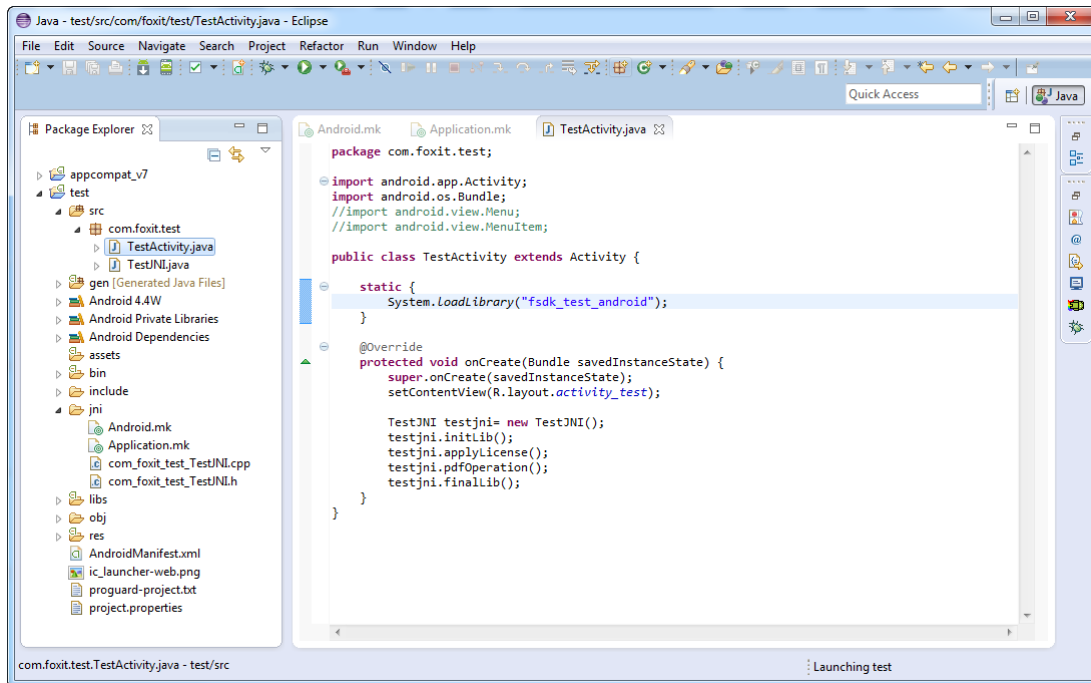
Figure 3-68

i) Click on "Run->Run as->Android Application" to run the test project. The running result is shown in Figure 3-69.

Figure 3-69

65

3.7   Windows Phone 8.1

3.7.1   What is in the package

Download Foxit PDF SDK zip for Windows Phone 8.1 C API package and extract it to a new directory "foxitpdfsdk_5_0_wp_c".The structure of the release package is shown in Figure 3-70. This package contains the following folders:

docs:	API references, developer guide
include:	header files for foxit pdf sdk API
lib	libraries and license files
samples:	sample projects and demos
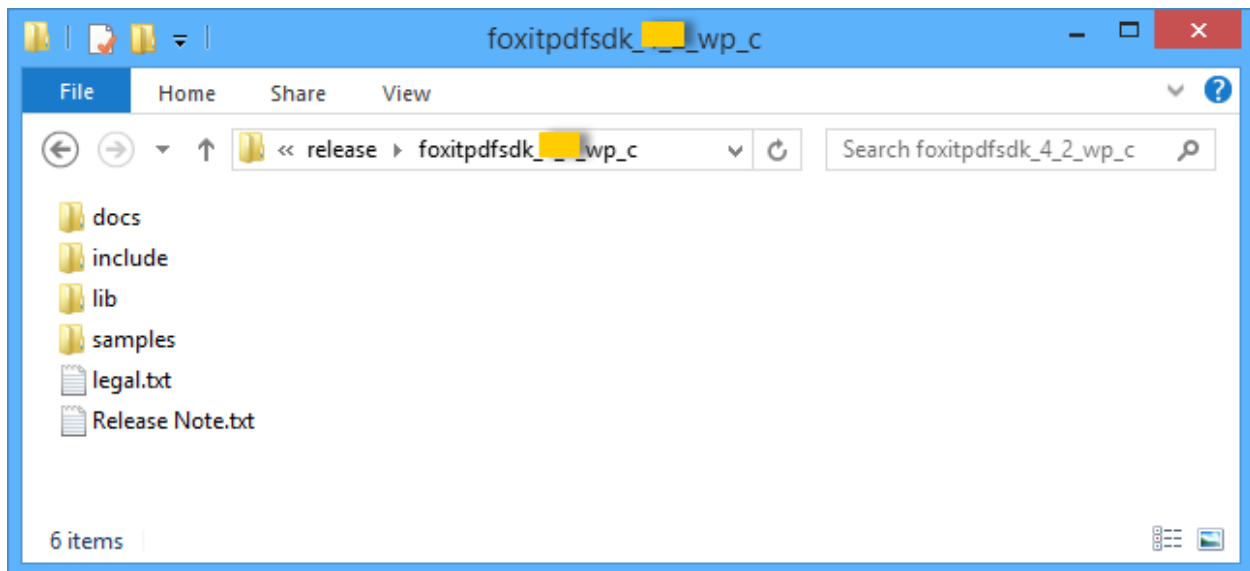
Figure 3-70

Foxit PDF SDK for Windows Phone 8.1 C API provides two programming language demos in folder "samples". The first one is C++, and the other one is CSharp(C#), which are shown in Figure 3-71.
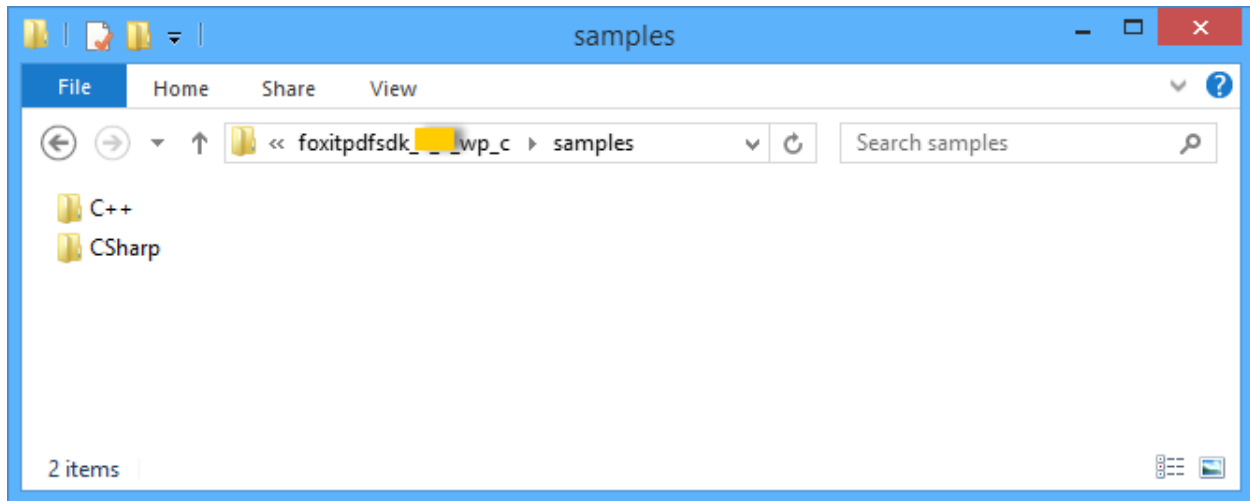
Figure 3-71

For each programming language, there are two demos such as demo_view and demo_annotations, which are provided to illustrate how to implement a simple PDF viewer and how to add annotations to a PDF document.

In "samples\C++", the directory structure of C++ demos is shown in Figure 3-72.
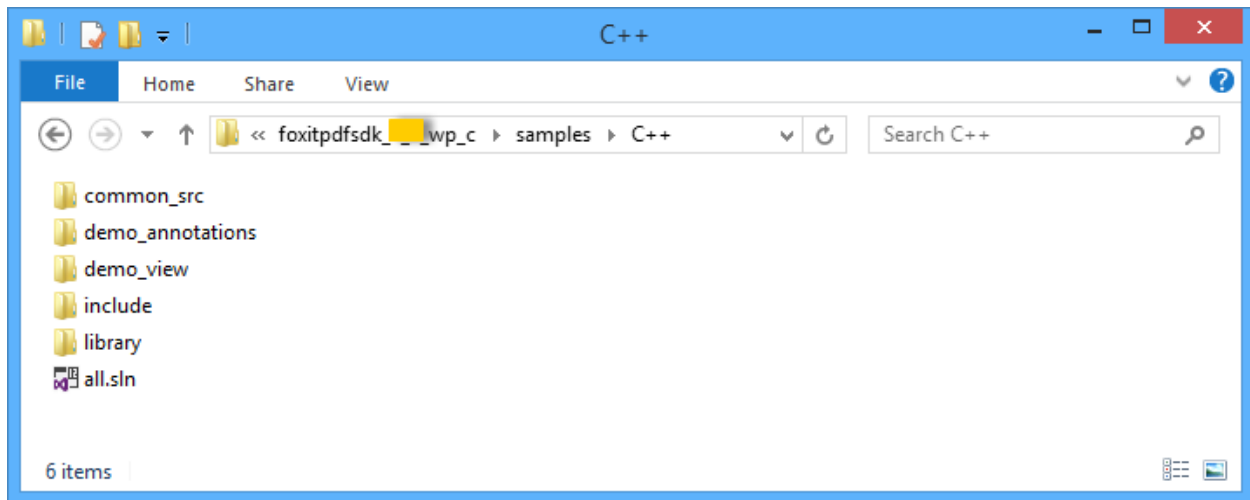
Figure 3-72

"**common_src**" is used for placing common codes that needed by all C++ demos.

"**demo_annotations**" is a demo that adds annotations to a PDF document.

"**demo_view**" is a simple PDF viewer demo.

"**include**" and "**library**" are used for placing header files and libraries files of Foxit PDF SDK (C API) respectively.

"**all.sln**" is a solution file that contains demo_annotations and demo_view project.

In "samples\CSharp", the directory structure of CSharp demos is shown in Figure 3-73.
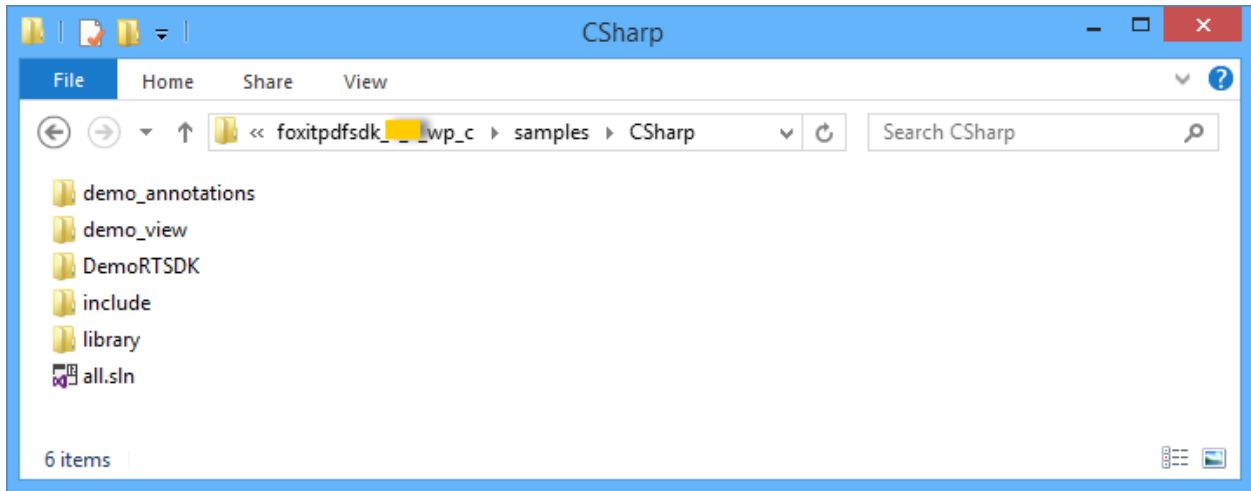
Figure 3-73

"**demo_annotations**" is a demo that adds annotations to a PDF document.

"**demo_view**" is a simple PDF viewer demo.

"**DemoRTSDK**" is a component project which gives a simple example on how to wrap PDF SDK C API. Here, it only includes the wrapped APIs that demos need.

"**include**" and "**library**" are used for placing header files and libraries files of Foxit PDF SDK (C API) respectively.

"**all.sln**" is a solution file that contains demo_annotations and demo_view project.

3.7.2   How to run a demo

demo_view

demo_view project provides an example for developers on how to implement a simple PDF viewer using Foxit PDF SDK APIs. To run the demo in Visual Studio 2013, follow the steps below: (in this guide, we use an emulator as an example to run the project)

    a) Load the visual studio solution ".sln" file. Here are two ways to load the ".sln" file.

        i.   Load the "all.sln" solution file. For C++ demo, the "all.sln" file is in directory "samples\C++", and for CSharp demo, it is in "samples\CSharp" directory.

ii. Load the "demo_view.sln" solution file. For C++ demo, the "demo_view.sln" file is in "samples\C++\demo_view" directory, and for CSharp demo, in "samples\CSharp\demo_view" directory.

In this case, we choose to load the "all.sln" file.

b) Change the build architecture of the project. Click on "Build->Configuration Manager" and choose Win32 for the active solution platform as shown in Figure 3-74.

Note: *If you will run the demo in a windows phone device, please choose the arm platform.*
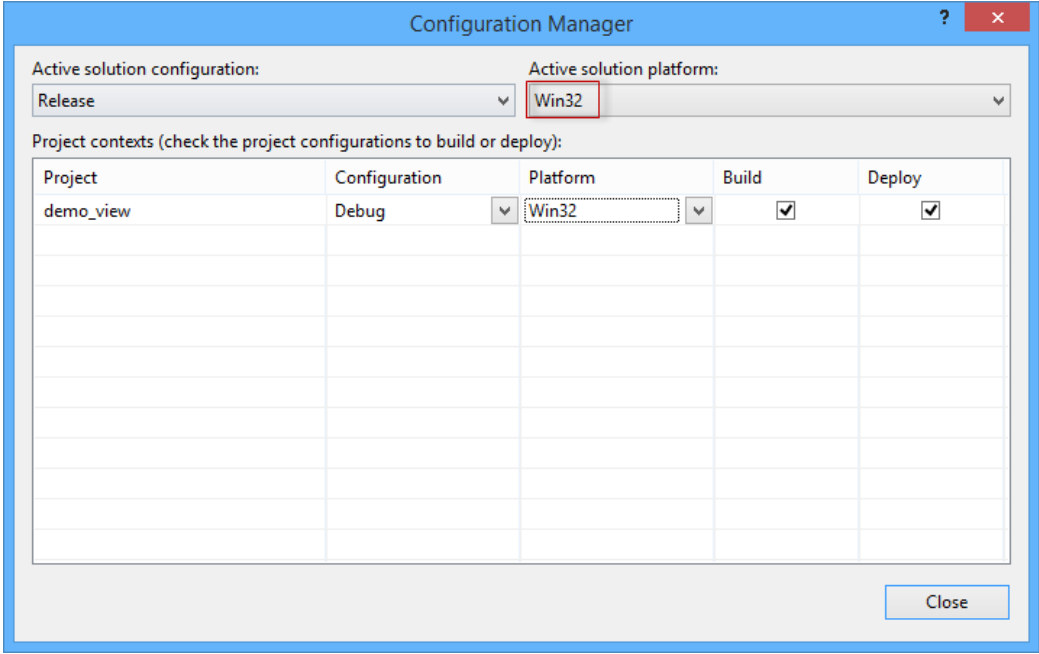
Figure 3-74

Here, we assume that the windows phone emulators have been installed in your computer. If not, please install it first.

i. Before installing, you should make sure that the CPU supports SLAT or virtualization (AMD-V or Inter-VT), which can be set in BIOS.

ii. Launch Hyper-v, go to "Control Panel", click on "Programs->Turn Windows feature on or off", check the Hyper-V checkbox in the list and click OK to install it as shown in Figure 3-75. Restart your computer when prompted.

iii. Now, you can install the windows phone emulators. Click the "Download Emulator 8.1…" as shown in Figure 3-76, then click "install" to go on installing the emulators.

69
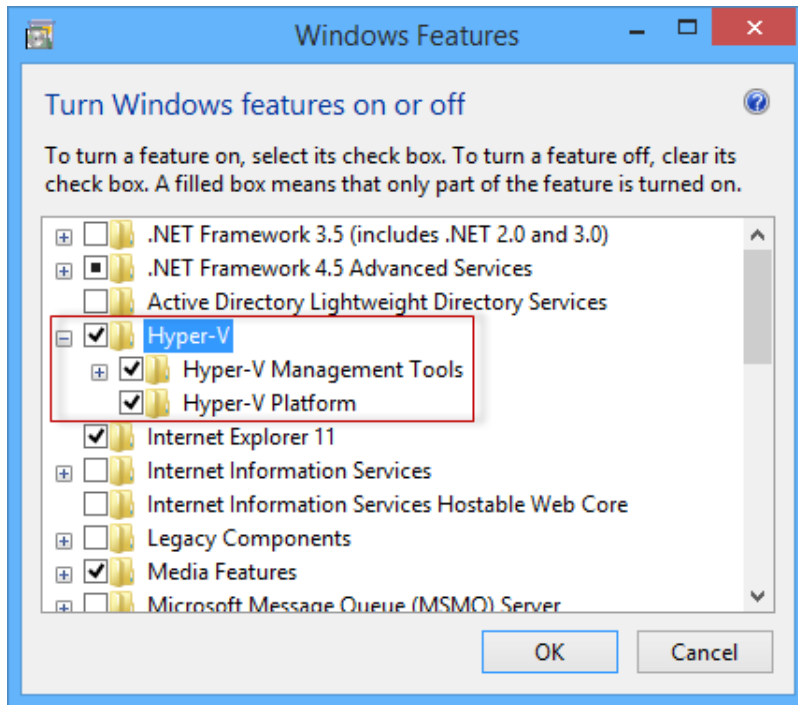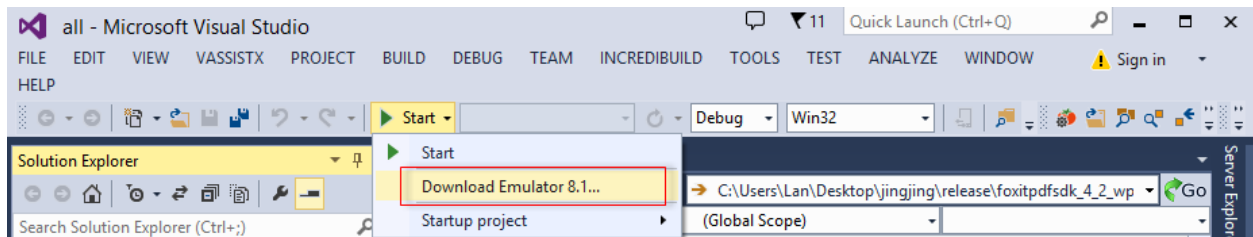
Figure 3-75

Figure 3-76

c) Set up demo_view as the startup project. Right click the project and select "Set as the Startup Project".

d) Click on "Emulator 8.1…" to build and run the demo. The screenshot of the demo is shown in Figure 3-77.

Figure 3-77

e) Put PDF files to emulator's SD card. In Figure 3-77, click on ">>" to see additional tools, click "SD Card" tab, set a local folder as SD card by "Browse", and then click the "Insert SD card" button, which are shown in Figure 3-78. Here, we put a PDF file named "AboutFoxit.pdf" to the folder "D:\testfiles". You can put any other PDF files to this folder.

Note: There is a known issue that the demo may cause crash when opening a file larger than 20M.
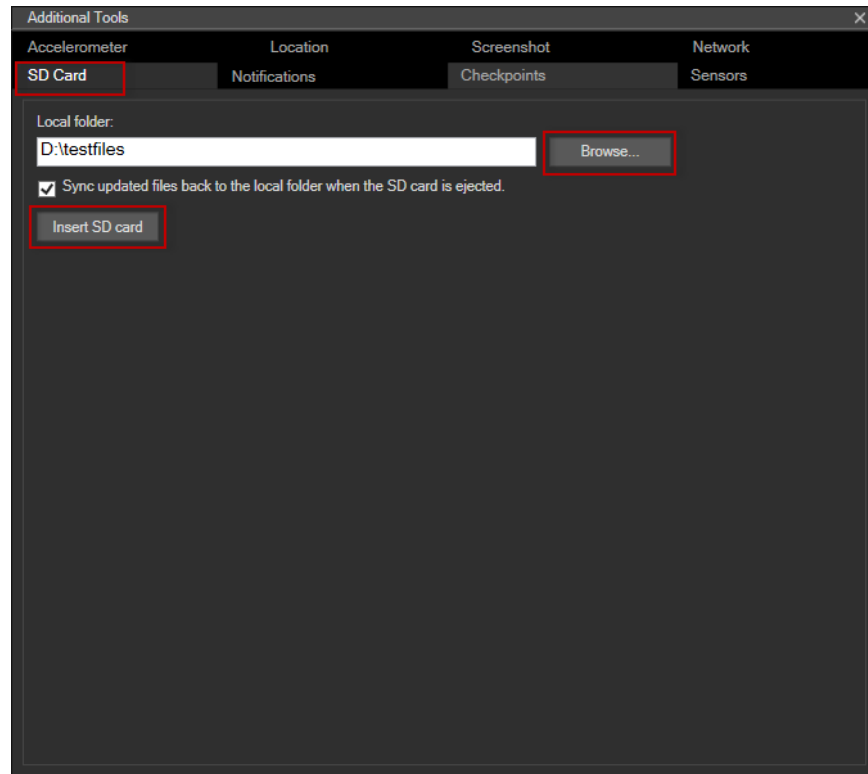
Figure 3-78

f) Open the "AboutFoxit.pdf" document. Click on "Render PDF" in Figure 3-77, choose "sd card->Documents", and then click the "AboutFoxit.pdf" to open it. This is shown in Figure 3-79.
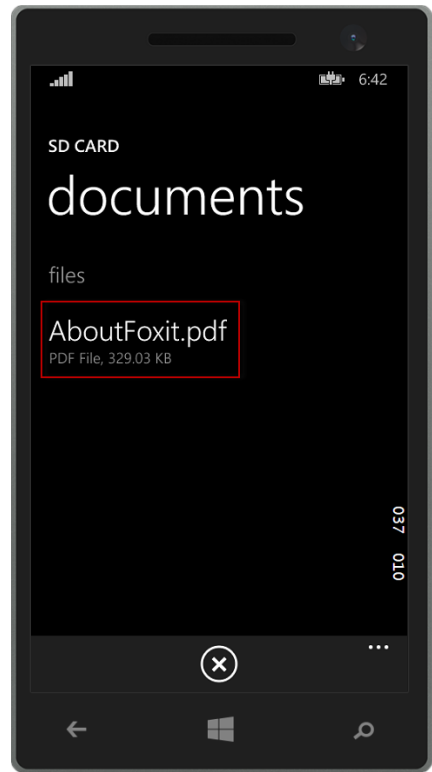
Figure 3-79

g) The "AboutFoxit.pdf" is displayed as shown in Figure 3-80. This demo provides the features like page turning, zooming. Click the "…" to see more features like rotating, actual Size, and fitting width/height as shown in Figure 3-81.

Figure 3-80

Figure 3-81

h) For example, click the "Rotate Left" button, the page of the document will be rotated 90 degree counterclockwise as shown in Figure 3-82.
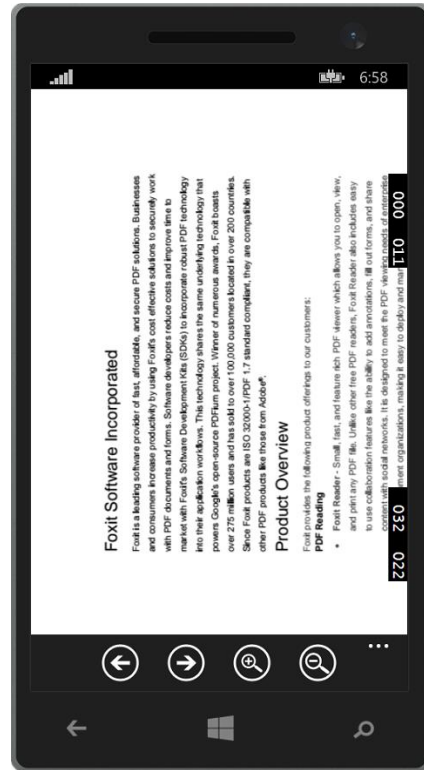
Figure 3-82

demo_annotations

Demo_annotations project provides an example for developers on how to add annotations to a PDF document using Foxit PDF SDK APIs. To run this demo in Visual Studio 2013, you can refer to the demo_view project.

a) Figure 3-83 shows the demo running in an emulator.

Figure 3-83

b) Click the "Render PDF" in Figure 3-83, choose "sd card->Documents", and then click the "AboutFoxit.pdf" to open it. The PDF document is displayed as shown in Figure 3-84.
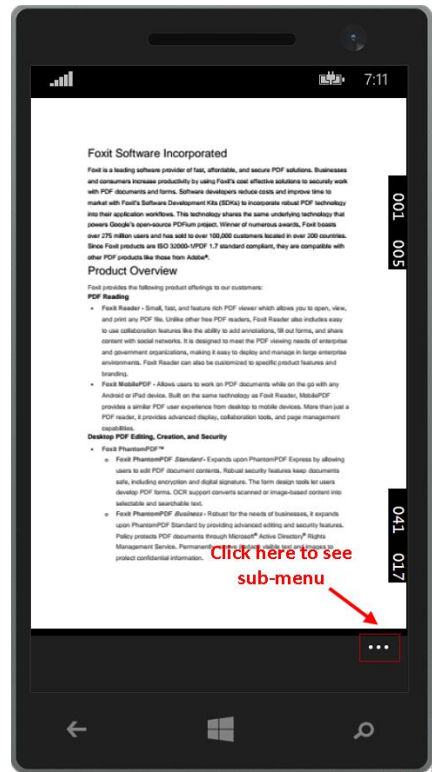
Figure 3-84

c) In Figure 3-84, click the "…" to see sub-menu including "Add Annots", "Remove Annots" and "Save As", which are shown in Figure 3-85.
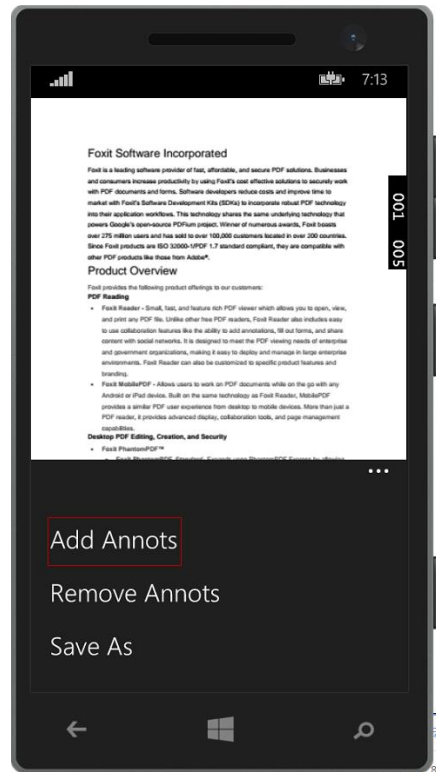
Figure 3-85

d) Click "Add Annots", some annotations will be added to the document as shown in Figure 3-86. In this demo, it supports annotations like link, line, underline, highlight, polygon, square, freetext and text.
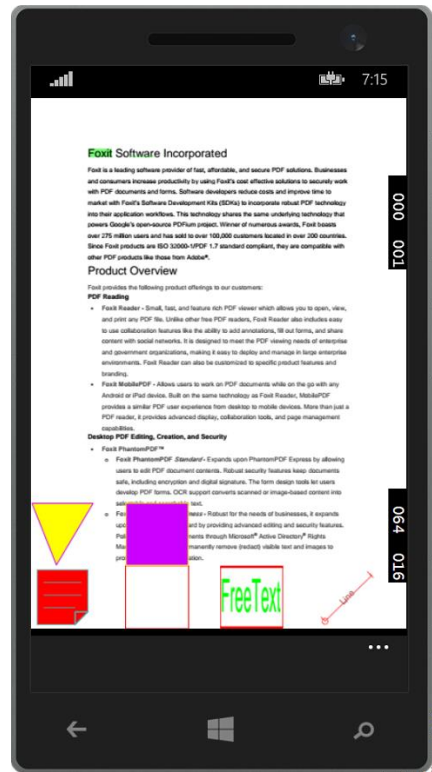
Figure 3-86

e) Click "Remove Annots", all annotations in this document will be removed as shown in Figure 3-87.
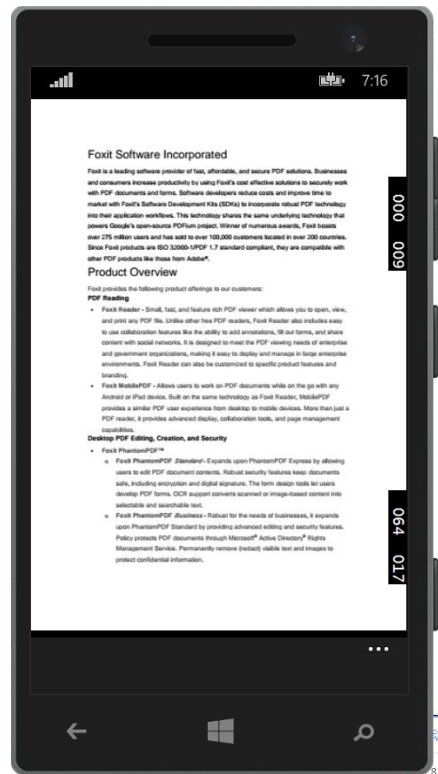
Figure 3-87

f) Click "Save As", you can save the PDF as a new file. Assume you save it to the "Documents" folder of the sd card, locate to "sd card->Documents", and click the "save" button as shown in Figure 3-88. A popup message dialog will be shown in Figure 3-89. Figure 3-90 shows the default saved name "New Document.pdf" in "sd card->Documents" folder.

Figure 3-88
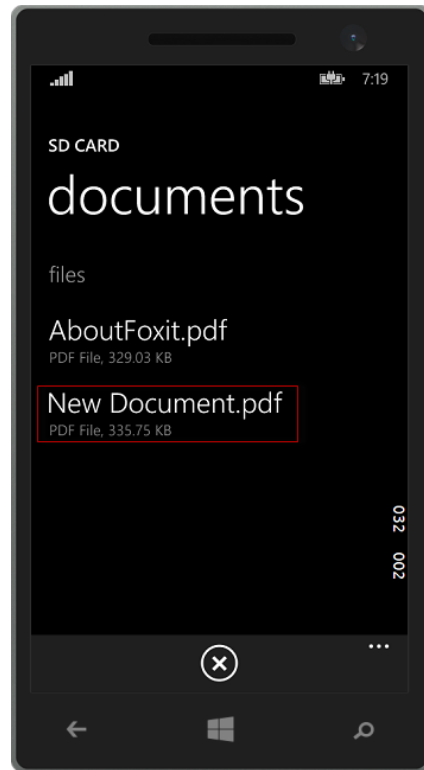
Figure 3-89

Figure 3-90

If you want to rename the saved file by yourself, please click the "…" to see sub-menu as shown in Figure 3-88. Then, Click the "rename file" button in Figure 3-91.
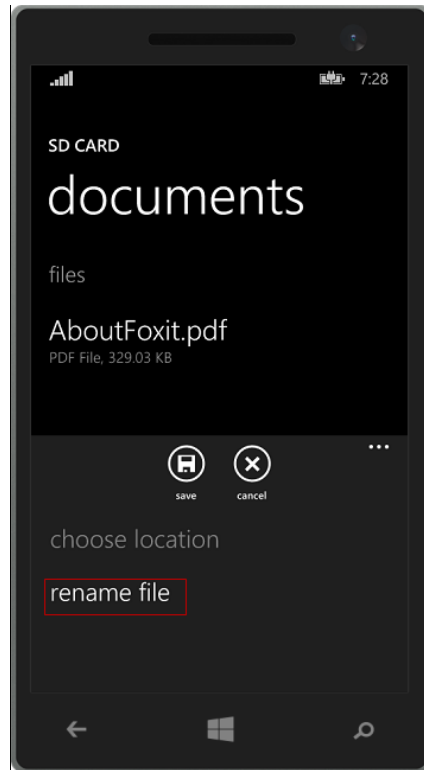
Figure 3-91

Type the file name, for example, "savefile", click the "save" button as shown in Figure 3-92. A popup message dialog will also be shown in Figure 3-93. Then, you will find the "savefile.pdf" in "sd card->Documents" folder as shown in Figure 3-94.
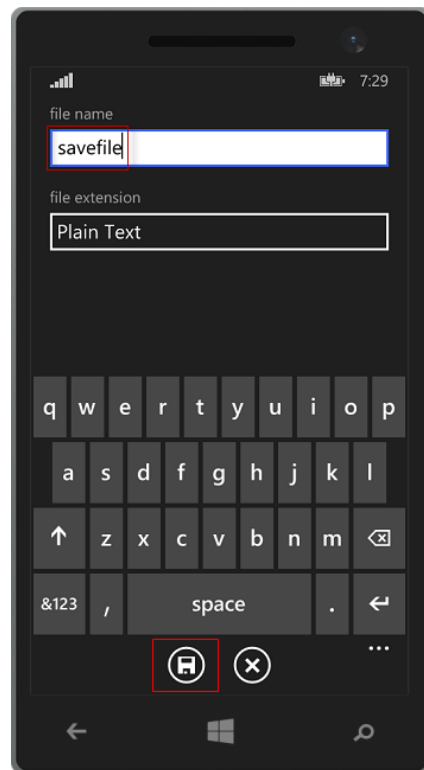
Figure 3-92
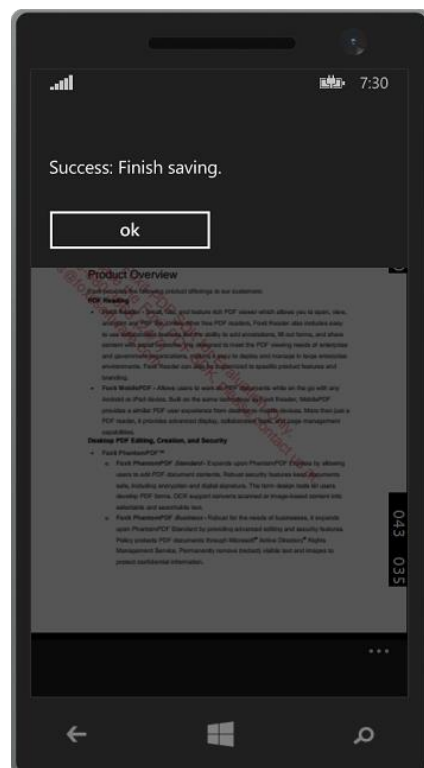
Figure 3-93
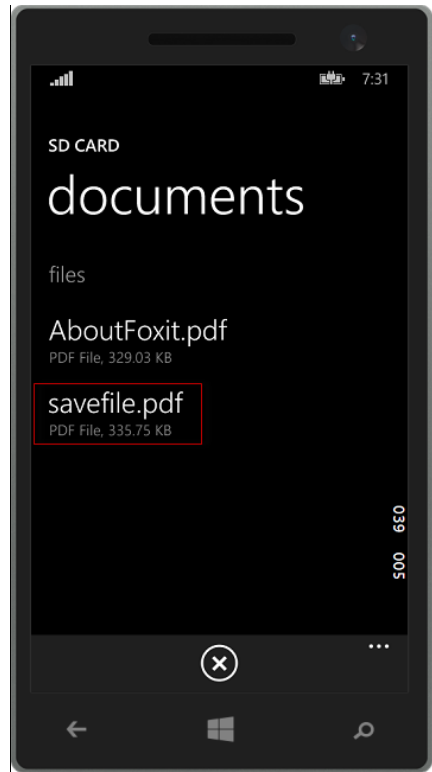
Figure 3-94

Note: Now, the "savefile.pdf" cannot be found in "D:\testfiles" folder. If you want to get this file, please click the "Eject SD card" as shown in Figure 3-95.
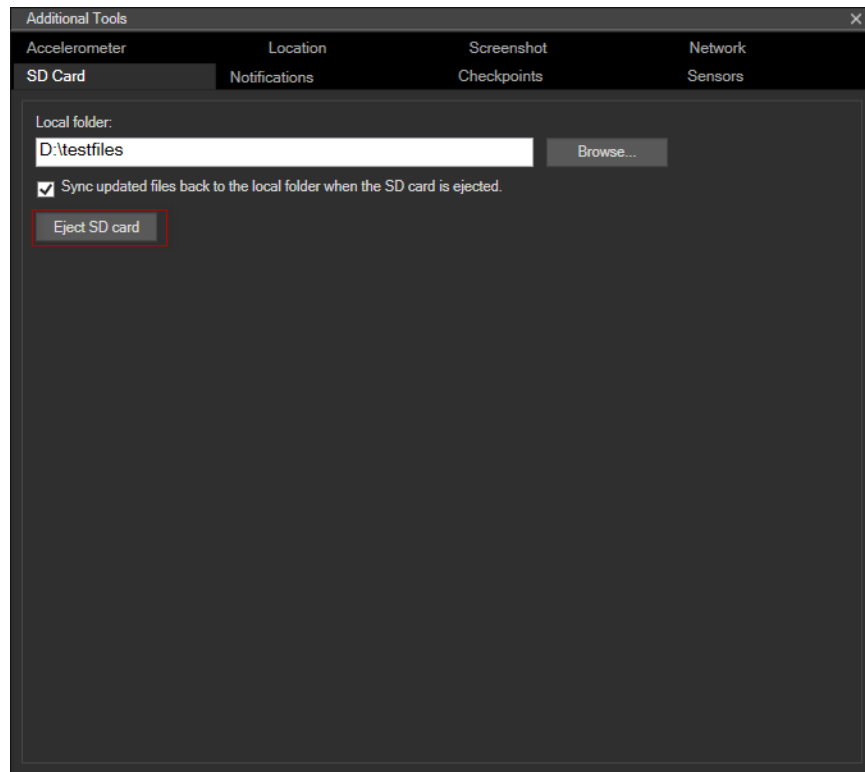
Figure 3-95

Then go to "D:\testfiles", you will find this file as shown in Figure 3-96.
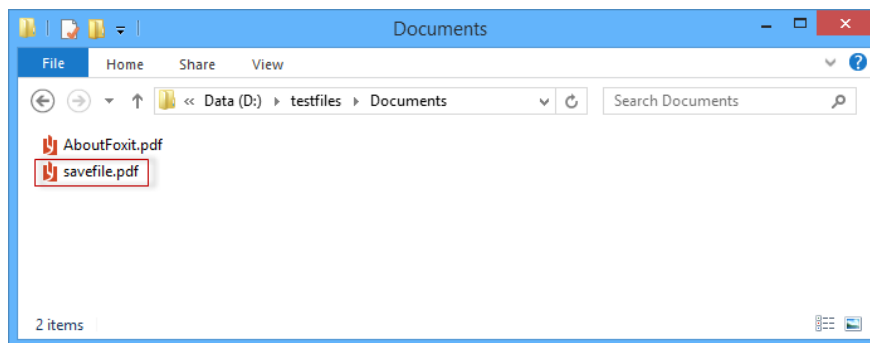
Figure 3-96

3.7.3 How to create your own project

In this section, we will show you how to create your own project by using Foxit PDF SDK APIs. Create a Windows Phone Apps project in Visual Studio 2013 called "test_wp" with C++ programming language as shown in Figure 3-97.
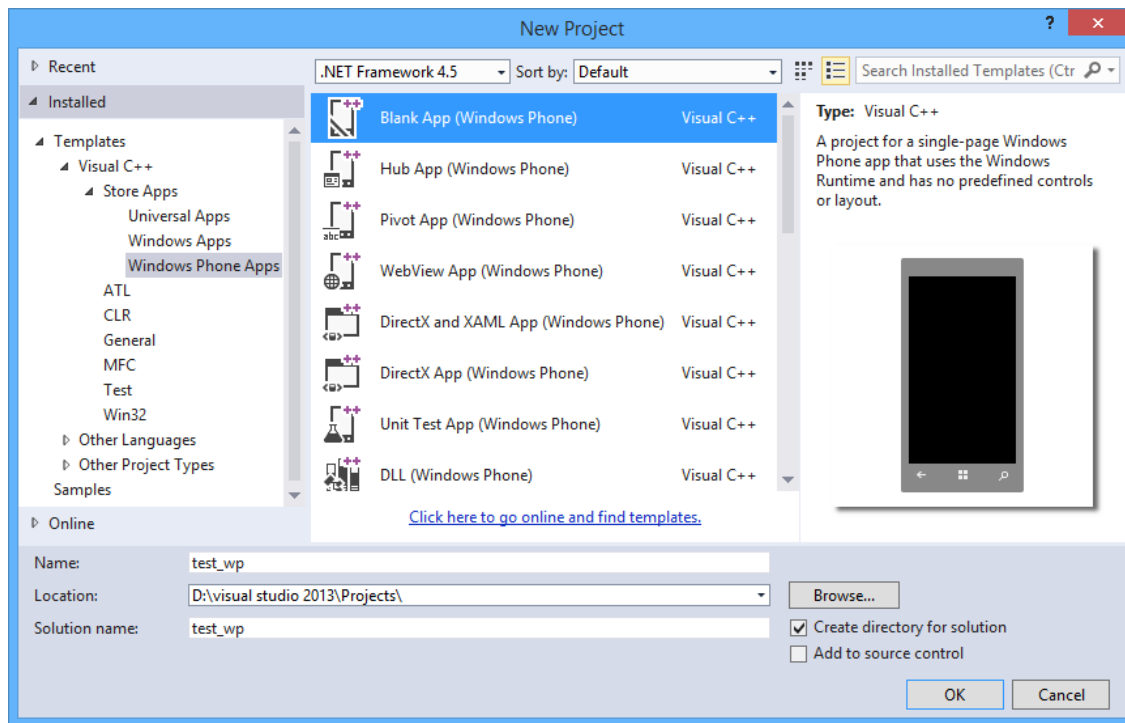
Figure 3-97

To run this project in Visual Studio 2013, please follow the steps below: (In this case, we also use an emulator as an example to run the project)

a) Copy "include" and "lib" folders from the download package to the project folder. Then the directory structure of the test_wp project is shown in Figure 3-98.

Figure 3-98

b) Change the build architecture of the project. Click on "Build->Configuration Manager" and choose Win32 for the active solution platform.

c) Add the libraries in "test_wp\lib" folders to the project in Visual Studio 2013. First, right click the "test_wp" project, choose "Add->New Filter" to create a new folder named "lib". Then, add the libraries by right clicking the "lib" folder, choosing "Add->Existing Item…" as shown in Figure 3-99, and select the "fsdk_wp_x86.dll" and "fsdk_wp_x86.lib" libraries in "test_wp\lib".

Figure 3-99

d) Set the properties of the libraries. The "Content" property of "fsdk_wp_x86.dll" should be set to "Yes" as shown in Figure 3-100, which deploys the dll library to the emulator along with the ".exe" file to make sure the project can run successfully.
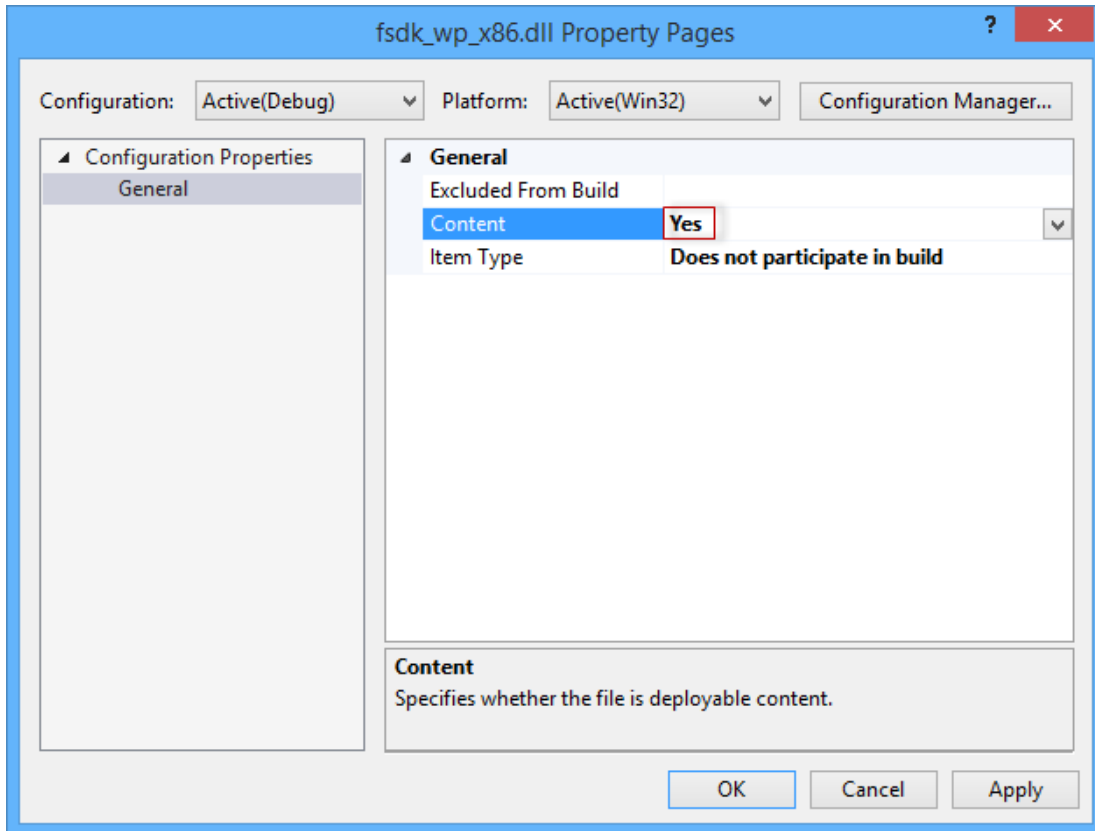
Figure 3-100

Note: If you also added the "fsdk_wp_arm.dll" and "fsdk_wp_arm.lib" to the project, please make sure the arm libraries should be excluded when building for Win32. Right click the "fsdk_wp_arm.dll", choose "properties", and set the "Excluded From Build" to "Yes" as shown in Figure 3-101. After setting the properties, the structure of the project will be like Figure 3-102.

If you want to run the project in a device, you should choose the arm libraries and do the similar setting as above. Here, we just introduce one way to configure the libraries, you can refer to this, or use other ways.

Figure 3-101

Figure 3-102

e) Construct the code to build a PDF application. Open the "MainPage.xaml", add a button, and set the content to "start your own project", which is shown in Figure 3-103. Here, we add a click event "Click_BTN_Start", but we did not implement it, you can do this by yourself.

```
<Page
    x:Class="test_wp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:test_wp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid>
        <Button x:Name="start" Content="start your own project" HorizontalAlignment="Left" Margin="24,45,0,0"
            VerticalAlignment="Top" Click="Click_BTN_Start"/>
    </Grid>

</Page>
```

Figure 3-103

f) Open the "MainPage.xaml.h", declare the methods as shown in Figure 3-104.

```
//
// MainPage.xaml.h
// Declaration of the MainPage class.
//

#pragma once

#include "MainPage.g.h"

namespace test_wp
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public ref class MainPage sealed
    {
    public:
        MainPage();
        int initLib();
        int applyLicense();
        int pdfOperation();
        int release();
    protected:
        virtual void OnNavigatedTo(Windows::UI::Xaml::Navigation::NavigationEventArgs^ e) override;
    private:
        void Click_BTN_Start(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e);
    };
}
```

Figure 3-104

g) Open the "MainPage.xaml.cpp" and implement the methods defined in "MainPage.xaml.h". The simple implement is shown in Figure 3-105 and you can go on your application in the "pdfOperation()" method. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.
Note: please include the "fsdk.h" first.

91

```
#include "pch.h"
#include "MainPage.xaml.h"
#include "../include/fsdk.h"

using namespace ...

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=234238

MainPage::MainPage()
{
    InitializeComponent();
    initLib();
    applyLicense();
    pdfOperation();
    release();
}

int MainPage::initLib()
{
    FS_RESULT ret = FSCRT_Library_CreateDefaultMgr();
    return (ret == FSCRT_ERRCODE_SUCCESS) ? 0 : -1;
    return 0;
}
int MainPage::applyLicense()
{
    //The implementation of applying license goes here
    return 0;
}
int MainPage::pdfOperation()
{
    // The implementation of pdf operation goes here
    return 0;
}
int MainPage::release()
{
    FSCRT_Library_DestroyMgr();
    return 0;
}
```

Figure 3-105

h) Click on "Emulator 8.1…" to build and run the demo. The screenshot of the demo is shown in
    Figure 3-106.

Figure 3-106

3.8 Windows 8.1 Store App

3.8.1 What is in the package

Download Foxit PDF SDK zip for WinRT C API package and extract it to a new directory
"foxitpdfsdk_5_0_winrt_c".The structure of the release package is shown in Figure 3-107. This package
contains the following folders:

docs: API references, developer guide

include: header files for foxit pdf sdk API

lib: libraries and license files
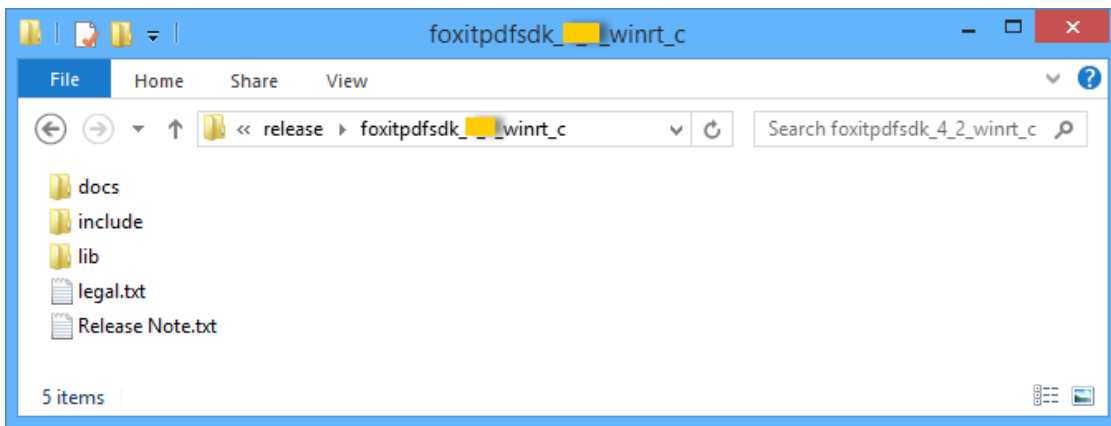
Figure 3-107

3.8.2 How to create your own project

In this section, we will show you how to create your own project by using Foxit PDF SDK APIs. Create a
Windows Apps project in Visual Studio 2013 called "test_winrt" with C++ programming language as
shown in Figure 3-108.
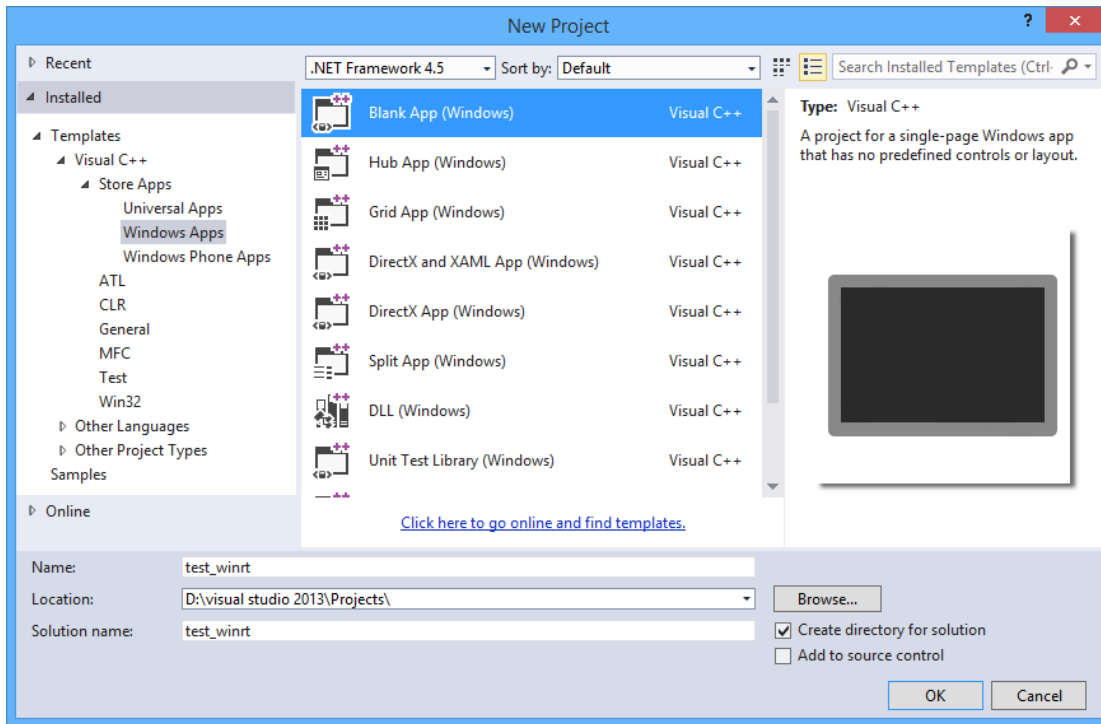
Figure 3-108

To run this project in Visual Studio 2013, please follow the steps below: (In this case, we use a Win32 simulator as an example to run the project)

a) Copy "include" and "lib" folders from the download package to the project folder. Then the directory structure of the test_winrt project is shown in Figure 3-109.
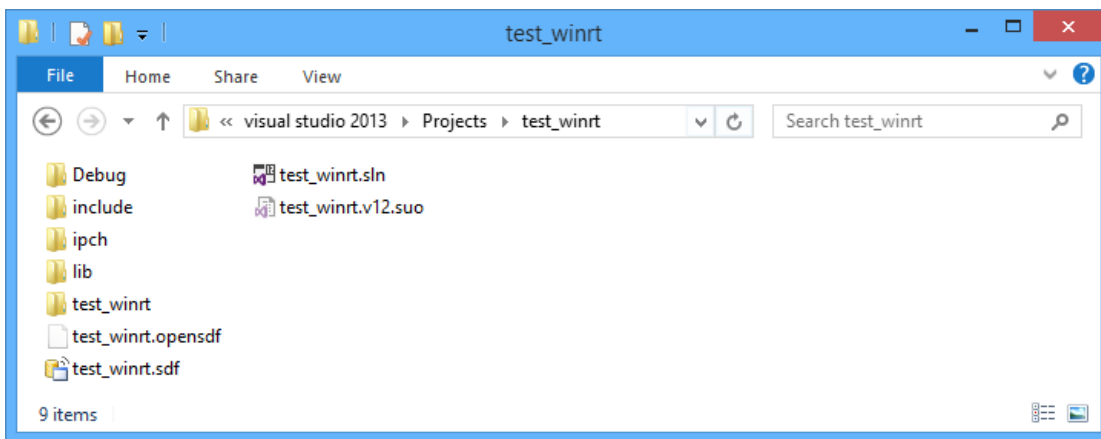
Figure 3-109

b) Change the build architecture of the project. Click on "Build->Configuration Manager" and choose Win32 for the active solution platform as shown in Figure 3-110.
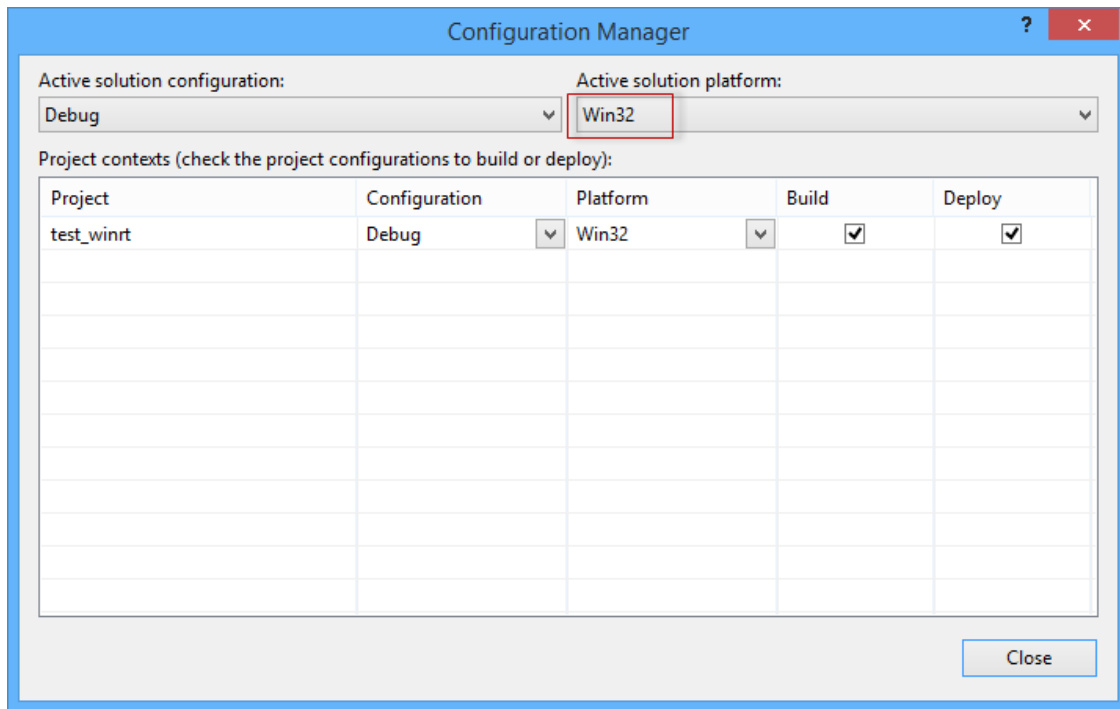
Figure 3-110

Note: *There are three active solution platforms: Win32, x64, and ARM. You should choose the proper platform for the build architecture according to the system you used to run the project.*

c) Add the libraries in "test_winrt\lib" folders to the project in Visual Studio 2013. First, right click the "test_winrt" project, choose "Add->New Filter" to create a new folder named "lib". Then, add the libraries by right clicking the "lib" folder, choosing "Add->Existing Item…" as shown in Figure 3-111, and select the "fsdk_win81_x86.dll" and "fsdk_win81_x86.lib" in "test_wp\lib". After adding, the struture of the project will be like Figure 3-112.
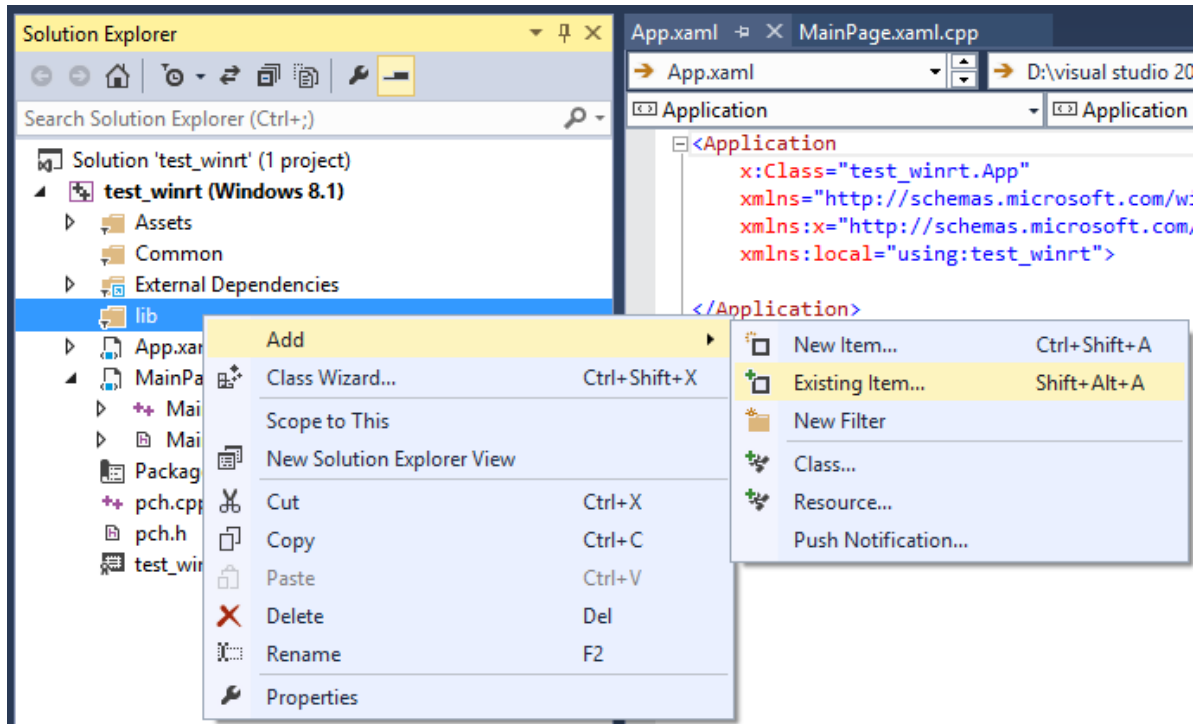
Figure 3-111
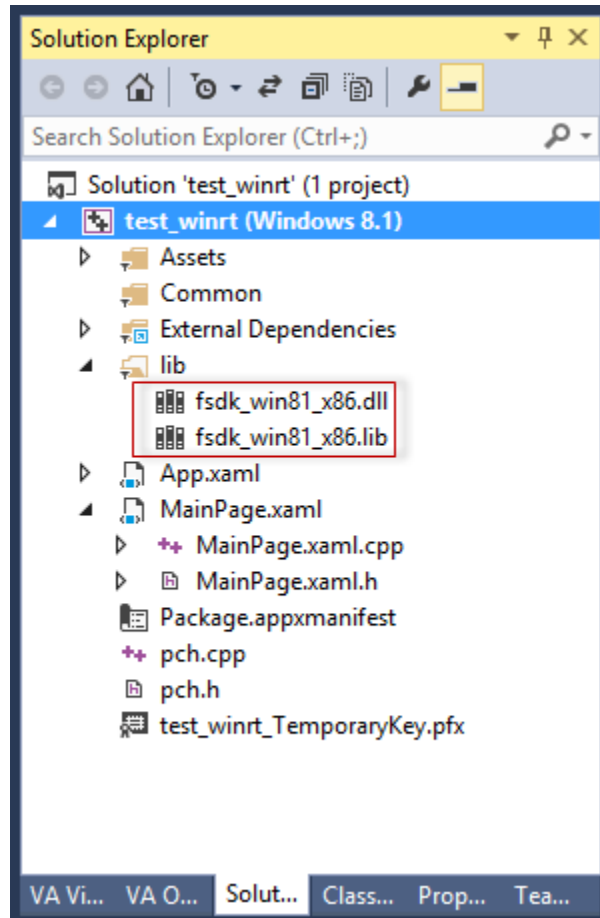
Figure 3-112

d) Set the properties of the libraries. The "Content" property of "fsdk_win81_x86.dll" should be set to "Yes" as shown in Figure 3-113, which deploys the dll library to the simulator along with the ".exe" file to make sure the project can run successfully.
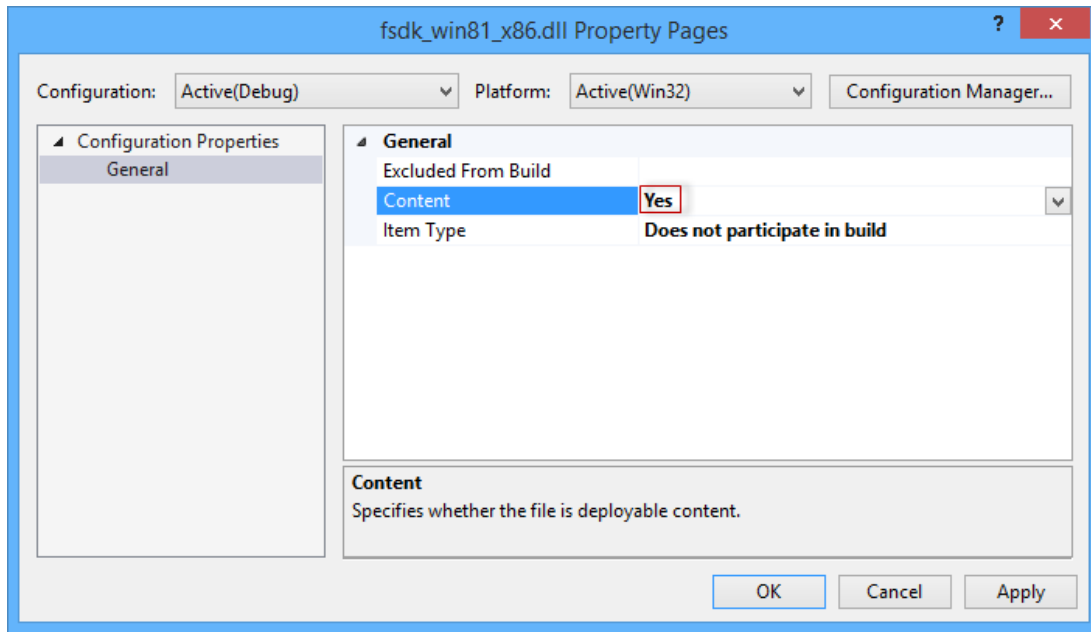
Figure 3-113

Note: You should also use the proper libraries according to the system you used to run the project. If you added other libraries to the project, please set the properties "Excluded From Build" of those libraries to "Yes". Here, we just introduce one way to configure the libraries, you can refer to this, or use other ways.

e) Construct the code to build a PDF application. Open the "MainPage.xaml", add a button, and set the content to "start your own project", which is shown in Figure 3-114. Here, we add a click event "Click_BTN_Start", but we did not implement it, you can do this by yourself.

```
<Page
    x:Class="test_winrt.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:test_winrt"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Button x:Name="start" Content="start your own project" HorizontalAlignment="Left" Margin="60,70,0,0" FontSize="30"
                VerticalAlignment="Top" Click="Click_BTN_Start"/>
    </Grid>
</Page>
```

Figure 3-114

f) Open the "MainPage.xaml.h", declare the methods as shown in Figure 3-115.

99

```
// 
// MainPage.xaml.h
// Declaration of the MainPage class.
// 

#pragma once

#include "MainPage.g.h"

namespace test_winrt
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public ref class MainPage sealed
    {
    public:
        MainPage();
        int initLib();
        int applyLicense();
        int pdfOperation();
        int release();

    private:
        void Click_BTN_Start(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e);
    };
}
```

Figure 3-115

g) Open the "MainPage.xaml.cpp" and implement the methods defined in "MainPage.xaml.h". The simple implement is shown in Figure 3-116 and you can go on your application in the "pdfOperation()" method. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.

Note: please include the "fsdk.h" first.

```
//
// MainPage.xaml.cpp
// Implementation of the MainPage class.
//
#include "pch.h"
#include "MainPage.xaml.h"
#include "../include/fsdk.h"

using namespace ...

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=234238

MainPage::MainPage()
{
    InitializeComponent();
    initLib();
    applyLicense();
    pdfOperation();
    release();
}
int MainPage::initLib()
{
    FS_RESULT ret = FSCRT_Library_CreateDefaultMgr();
    return (ret == FSCRT_ERRCODE_SUCCESS) ? 0 : -1;
    return 0;
}
int MainPage::applyLicense()
{
    //The implementation of applying license goes here
    return 0;
}
int MainPage::pdfOperation()
{
    // The implementation of pdf operation goes here
    return 0;
}
int MainPage::release()
{
    FSCRT_Library_DestroyMgr();
    return 0;
}
void test_winrt::MainPage::Click_BTN_Start(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e)
{

}
```

Figure 3-116

h) Click on "Simulator" to build and run the demo as shown in Figure 3-117. The screenshot of the demo is shown in Figure 3-118.

Figure 3-117

101
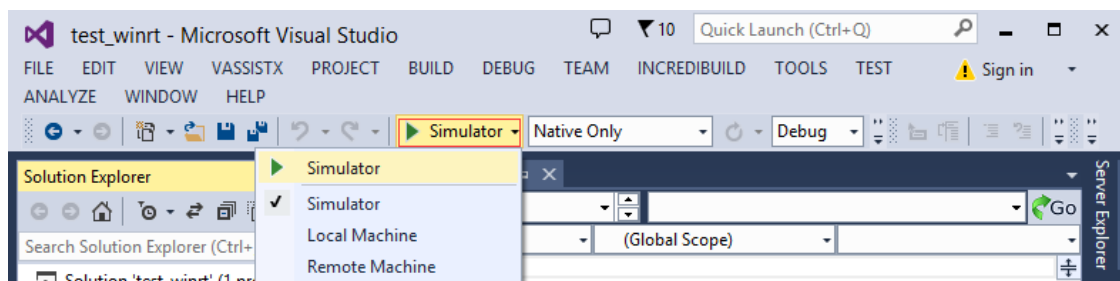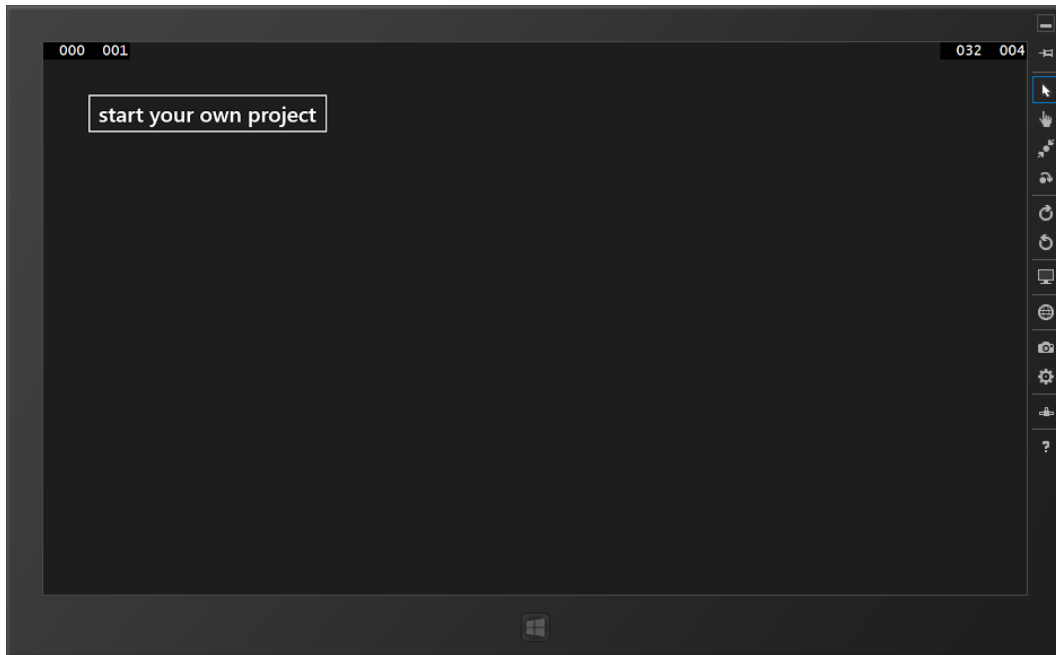
Figure 3-118

3.9　Windows 10 Universal App

3.9.1　What is in the package

Download Foxit PDF SDK zip for Windows 10 C API package and extract it to a new directory
"foxitpdfsdk_5_3_win10_c".The structure of the release package is shown in Figure 3-119. This package
contains the following folders:

docs:　　　　　API references, developer guide

include:　　　　header files for foxit pdf sdk API

lib　　　　　　libraries and license files

samples:　　　sample projects and demos
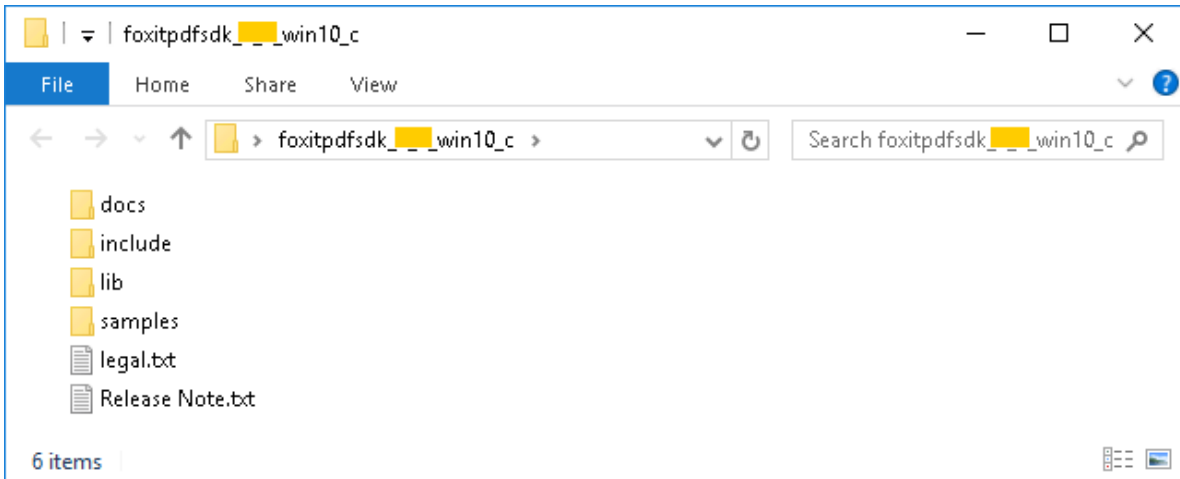
Figure 3-119

Foxit PDF SDK for Windows 10 C API provides two programming language demos in folder "samples".
The first one is C++, and the other one is CSharp(C#), which are shown in Figure 3-120.

Figure 3-120

103
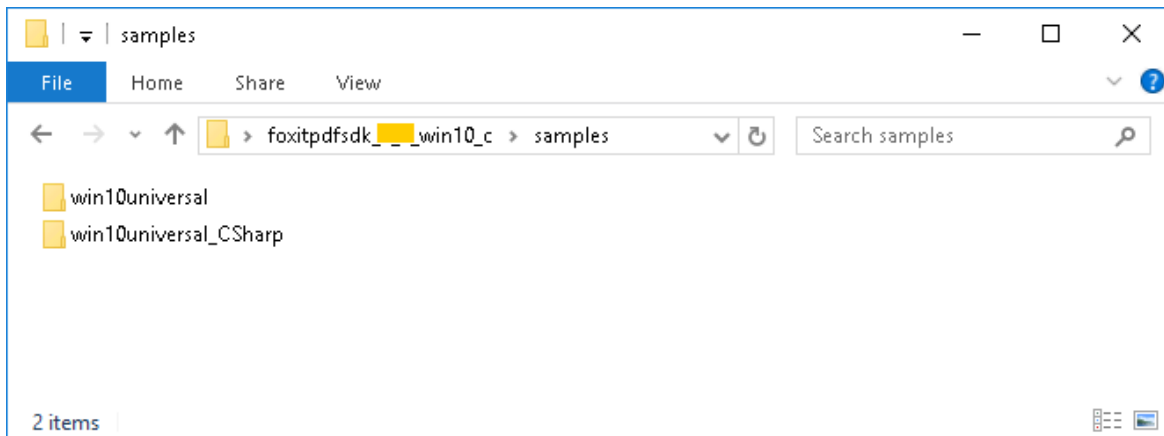
3.9.2 How to run a demo

demo_view (C++)

This view demo provides an example for C++ developers on how to implement a simple PDF viewer in Windows 10 platform using Foxit PDF SDK APIs. To run the demo in Visual Studio, follow the steps below: (in this guide, we use an x86 simulator as an example to run the project in Visual Studio 2015)

a) Load the "demo_view.sln" file under "samples\win10universal" folder in Visual Studio 2015.

b) Change the build architecture of the project. Click on "Build->Configuration Manager" and choose x86 for the active solution platform as shown in Figure 3-121.
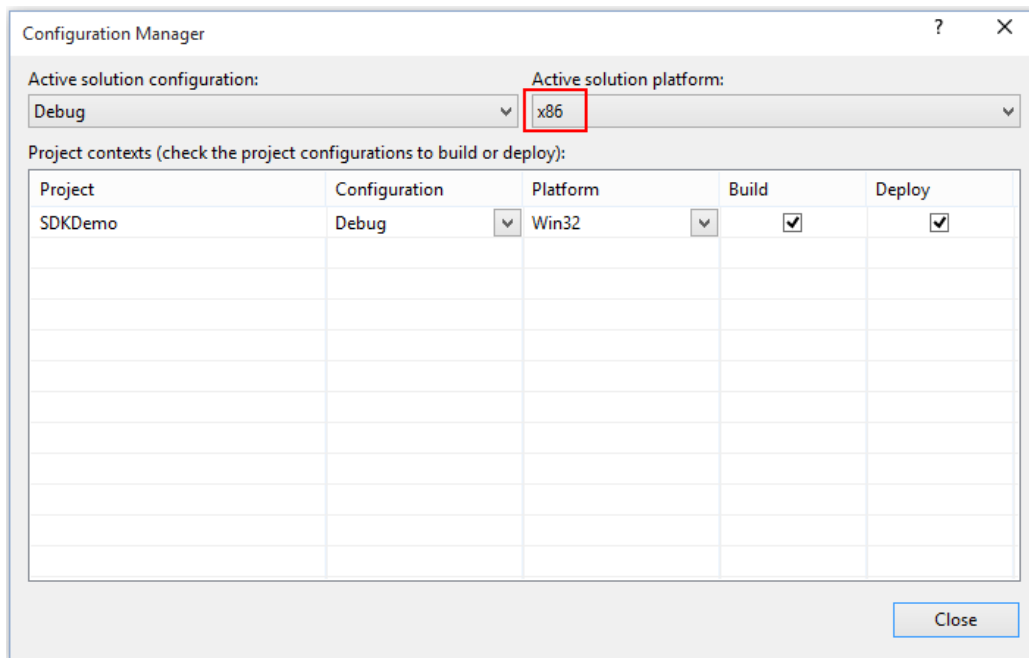
Figure 3-121

Note: *There are three active solution platforms: x86, x64, and arm. You should choose the proper platform for the build architecture according to the system you used to run the project. For example, if you will run the demo in an arm device, please choose ARM for the "Active solution platform".*

c) Click on "Simulator" as shown in Figure 3-122 to build and run the demo. The screenshot of the demo is shown in Figure 3-123.
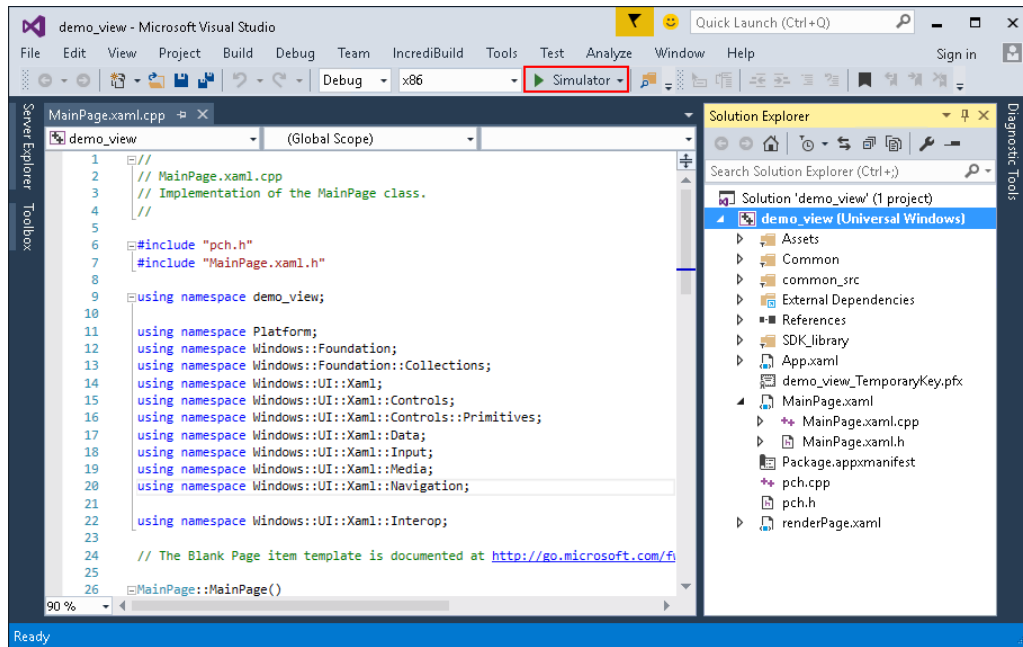
Figure 3-122
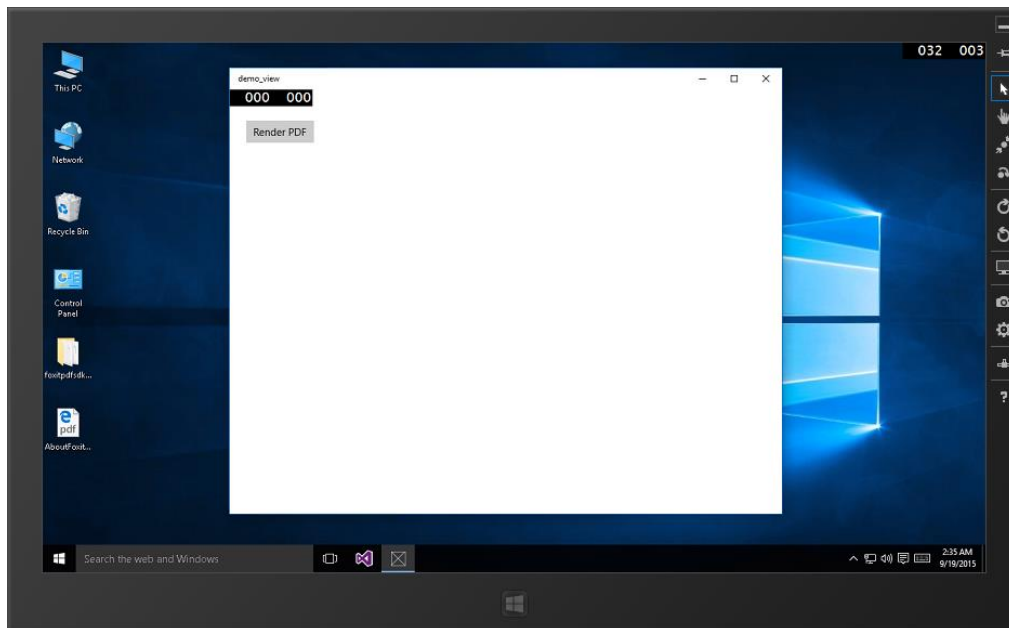
Figure 3-123

d) Click on "Render PDF" to open a PDF file. Here, we open a PDF document named "AboutFoxit.pdf". The "AboutFoxit.pdf" will be displayed as shown in Figure 3-124.
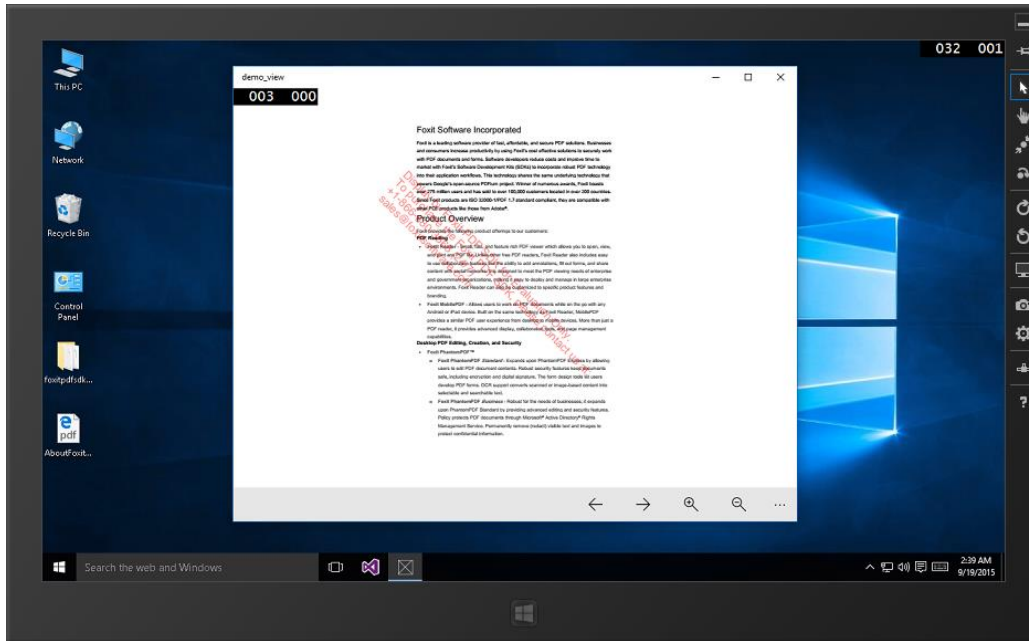
Figure 3-124

e) This demo provides the features like page turning, zooming. Click the "…" to see more features like rotating left/right, actual Size, and fitting width/height as shown in Figure 3-125.
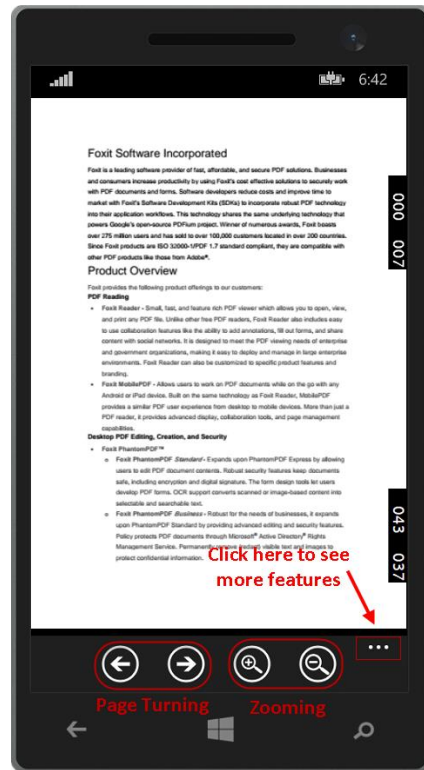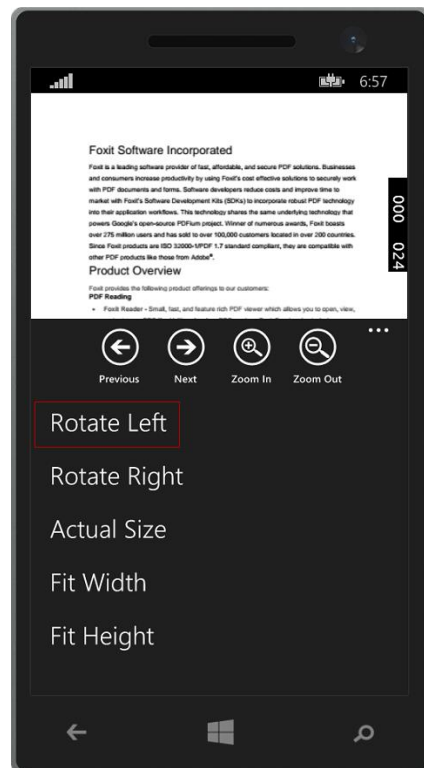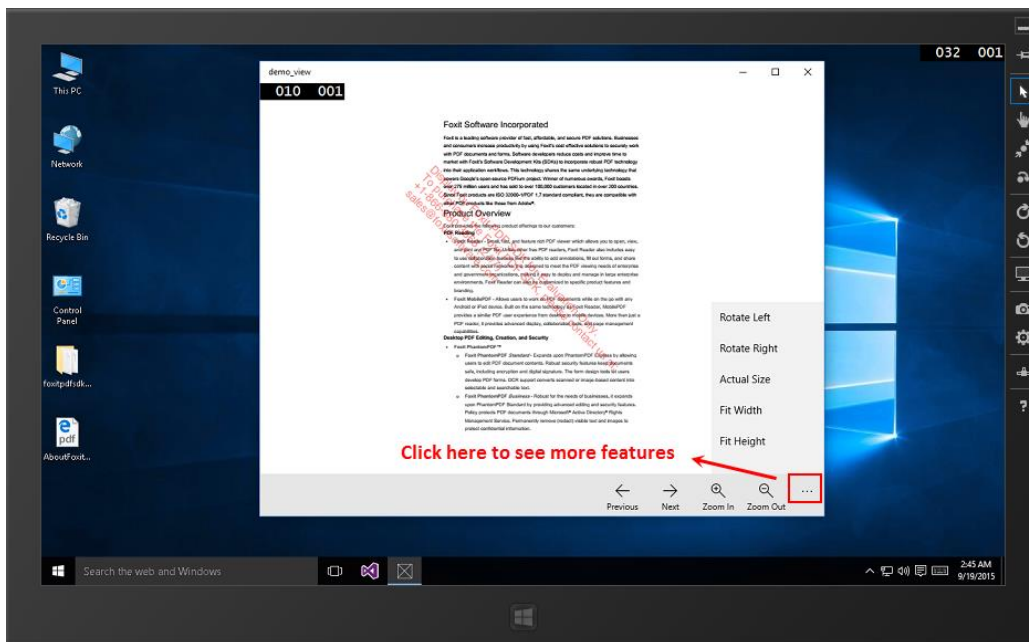

Figure 3-125

f) For example, click the "Rotate Left" button, the page of the document will be rotated 90 degree counterclockwise as shown in Figure 3-126.
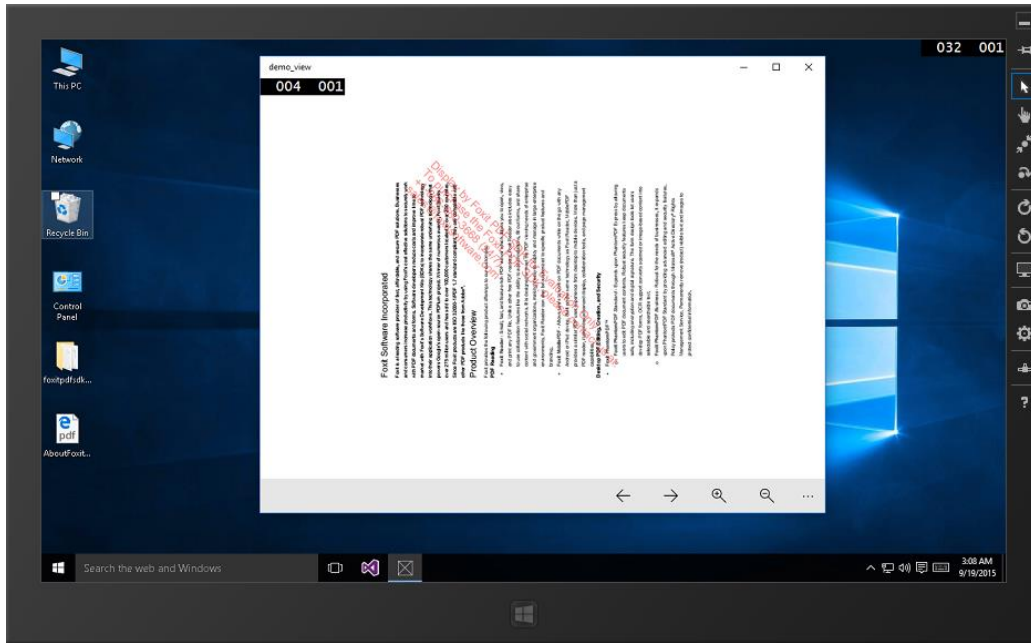
Figure 3-126

FSDKWin10CSharpDemo(C#)

This view demo provides an example for C# developers on how to implement a simple PDF viewer in Windows 10 platform using Foxit PDF SDK APIs. To run the demo in Visual Studio, follow the steps below: (in this guide, we use an x86 simulator as an example to run the project in Visual Studio 2015)

a) Load the "FSDKWin10CSharpDemo.sln" file under "samples\win10universal_CSharp" folder in Visual Studio 2015.

b) Change the build architecture of the project. Click on "Build->Configuration Manager" and choose x86 for the active solution platform as shown in Figure 3-127.
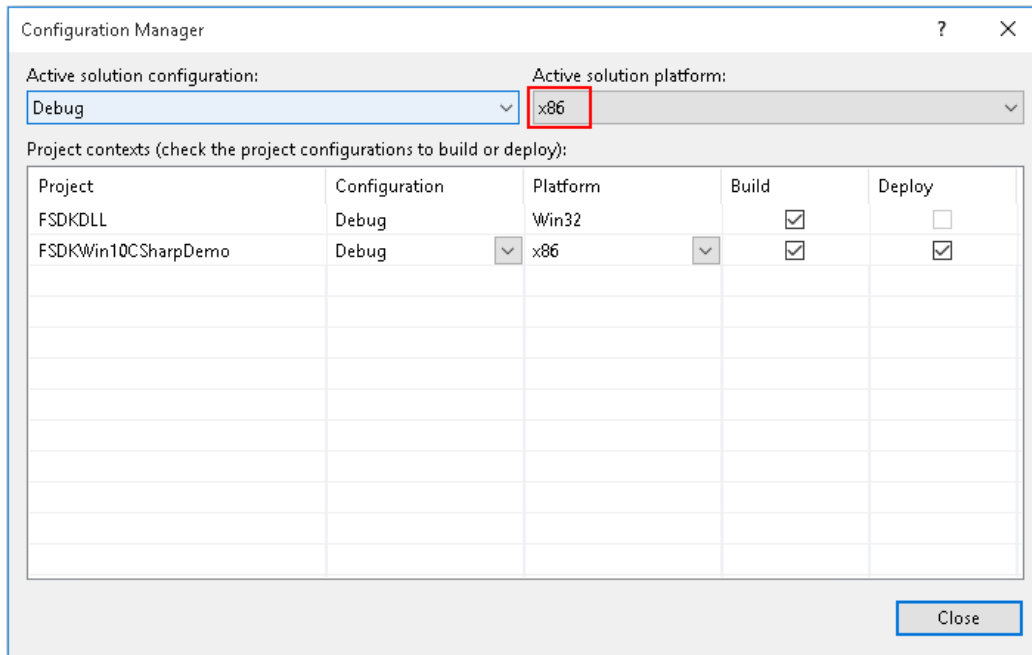
Figure 3-127

Note: *There are three active solution platforms: x86, x64, and arm. You should choose the proper platform for the build architecture according to the system you used to run the project. For example, if you will run the demo in an arm device, please choose ARM for the "Active solution platform".*

c) Build the project "FSDKDLL" by right clicking the project "FSDKDLL" and select "Build". After successfully building, the "FSDKDLL.dll" dynamic library will be generated in the "samples\win10universal_CSharp\Debug\FSDKDLL" folder as shown in Figure 3-128.
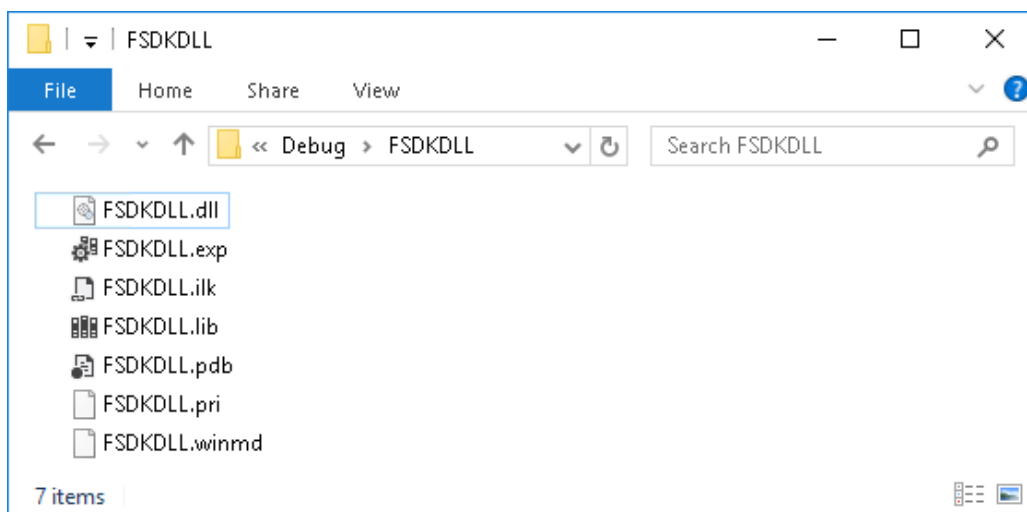
Figure 3-128

FSDKDLL project is used for wrapping the Foxit PDF SDK Windows 10 C library to class library with C++/CX which can be called in C# project. However, you must first add a reference to the C# project before using the classes and functions in FSDKDLL.dll library.

d) Add "FSDKDLL.dll" reference to the FSDKWin10CSharpDemo project.

i.   In **Solution Explorer**, right-click the **FSDKWin10CSharpDemo** project node and click **Add Reference** as shown in Figure 3-129.
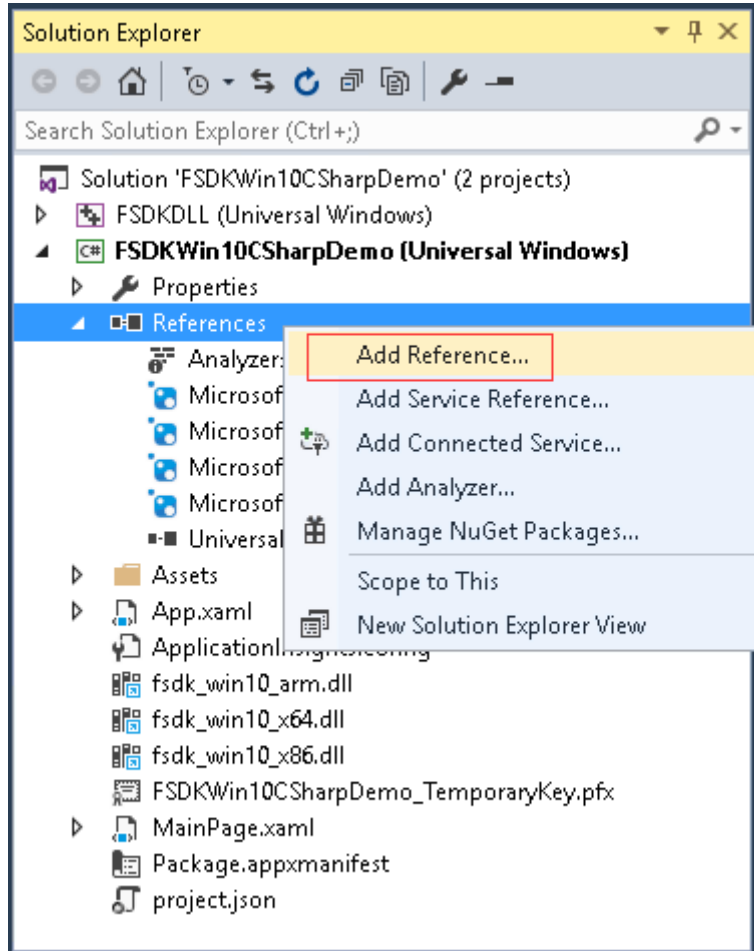
Figure 3-129

ii.   In the **Reference Manager** dialog, select the **Project** -> **Solution** tab, check the box next to "FSDKDLL", and then click **OK**. It is shown in Figure 3-130.
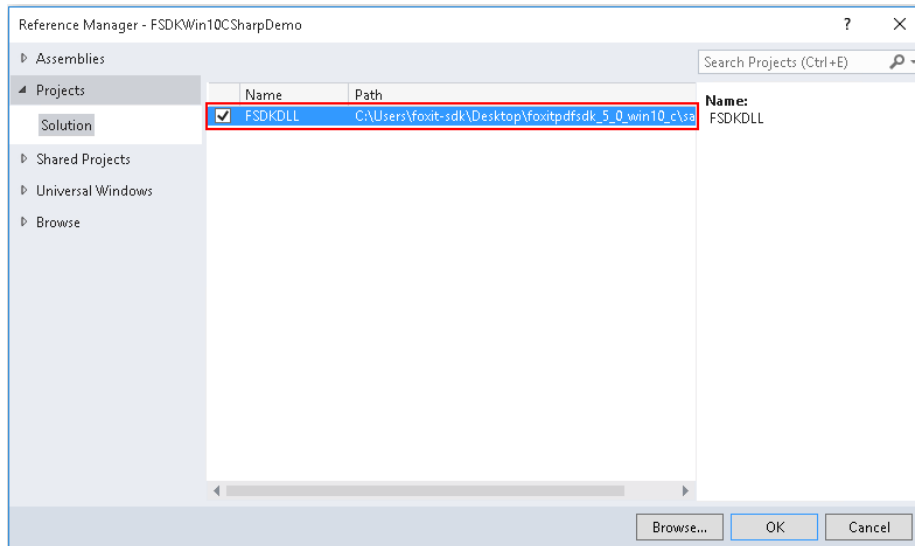
Figure 3-130

e) Right click the project "FSDKWin10CSharpDemo" and select "Set as StartUp Project". Click on "Simulator" as shown in Figure 3-131 to build and run the demo. The screenshot of the demo is shown in Figure 3-132.
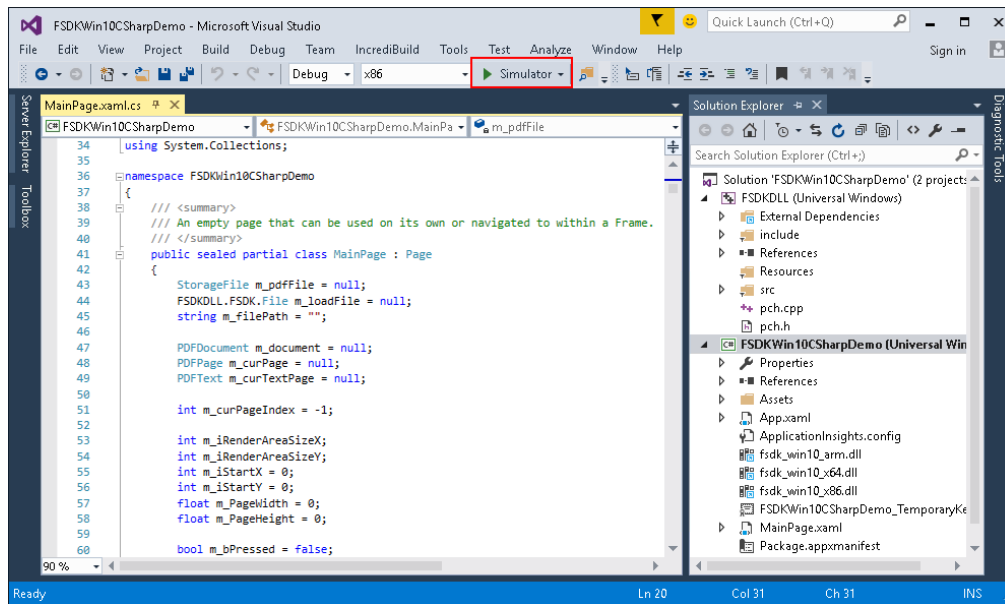
Figure 3-131
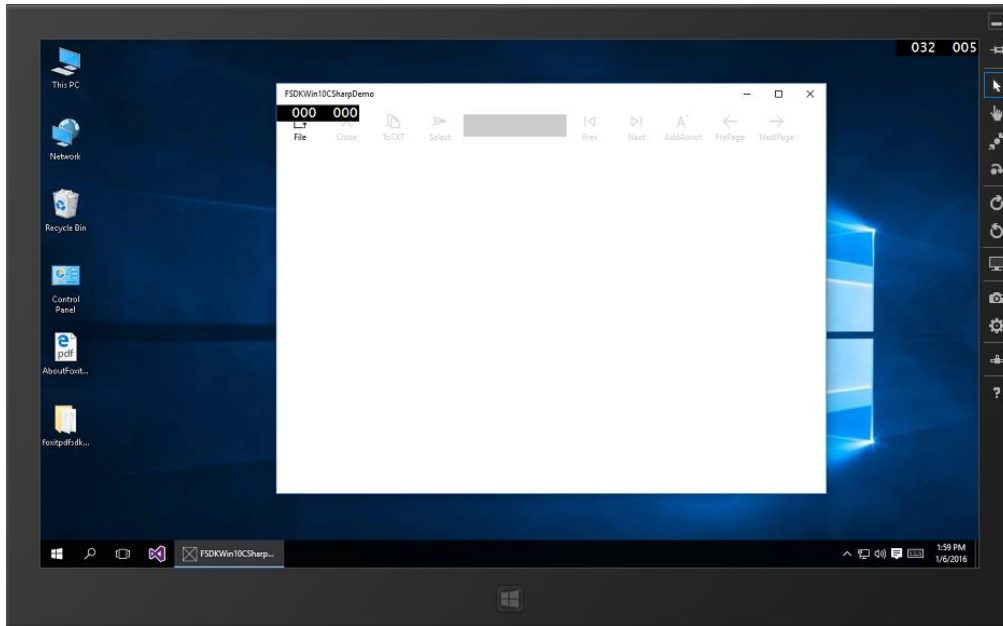
Figure 3-132

g) Click on "File" to open a PDF file. Here, we open a PDF document named "AboutFoxit.pdf". The "AboutFoxit.pdf" will be displayed as shown in Figure 3-133.
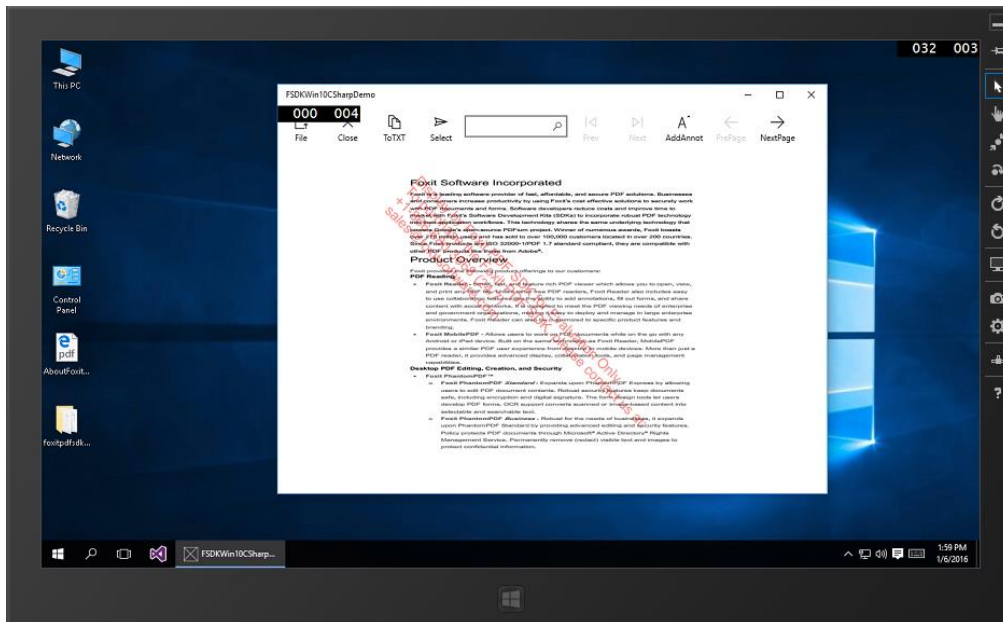
Figure 3-133

h) This demo provides the features like converting to txt, text manipulation including search, selection and copy (the selected text will be copied to clipboard automatically), adding annotations, and page turning.

For example, click on "AddAnnot", the current page will be added some annotations like Figure 3-134.
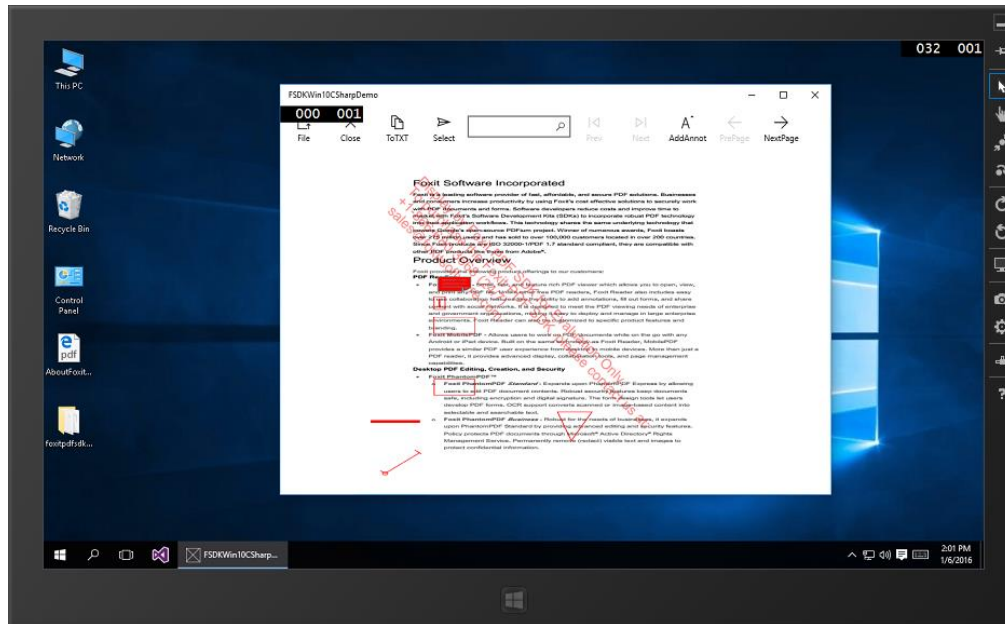
Figure 3-134

Click on "ToTxT", input the txt file name and choose the location to save the generated txt file. Here, we input "aboutfoxit" and save it to the "Desktop" as shown in Figure 3-135.

Figure 3-135

112
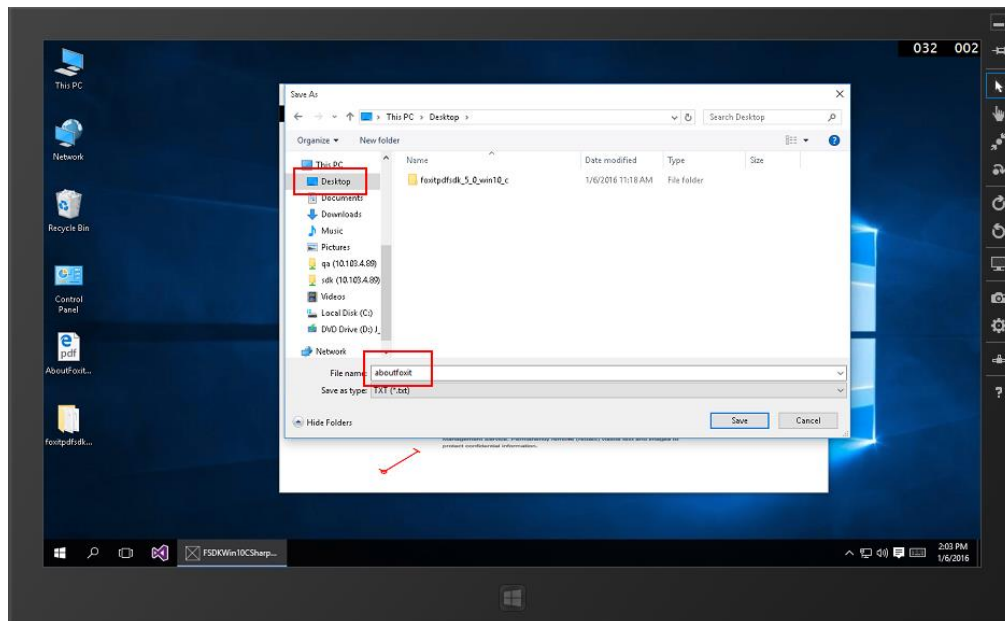
After clicking on "Save", the "aboutfoxit.txt" file will be generated on the "Desktop", and then open it to browse the contents as shown in Figure 3-136.
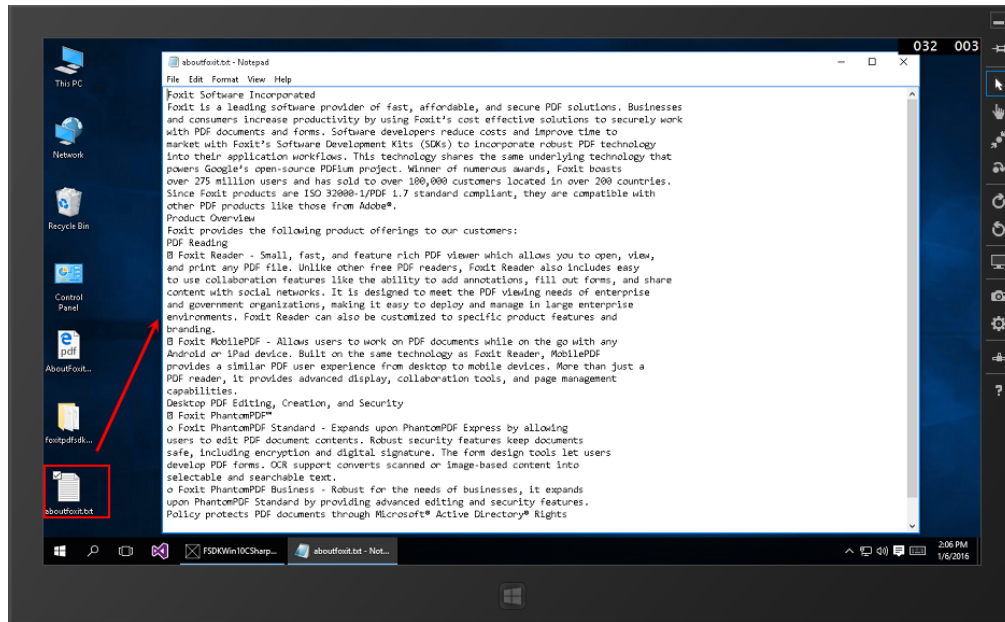
Figure 3-136

3.9.3   How to create your own project

In this section, we will take a C++ project as an example to show you how to create your own project by using Foxit PDF SDK APIs. Create a Windows Universal project in Visual Studio 2015 called "test_win10universal" with C++ programming language as shown in Figure 3-137.
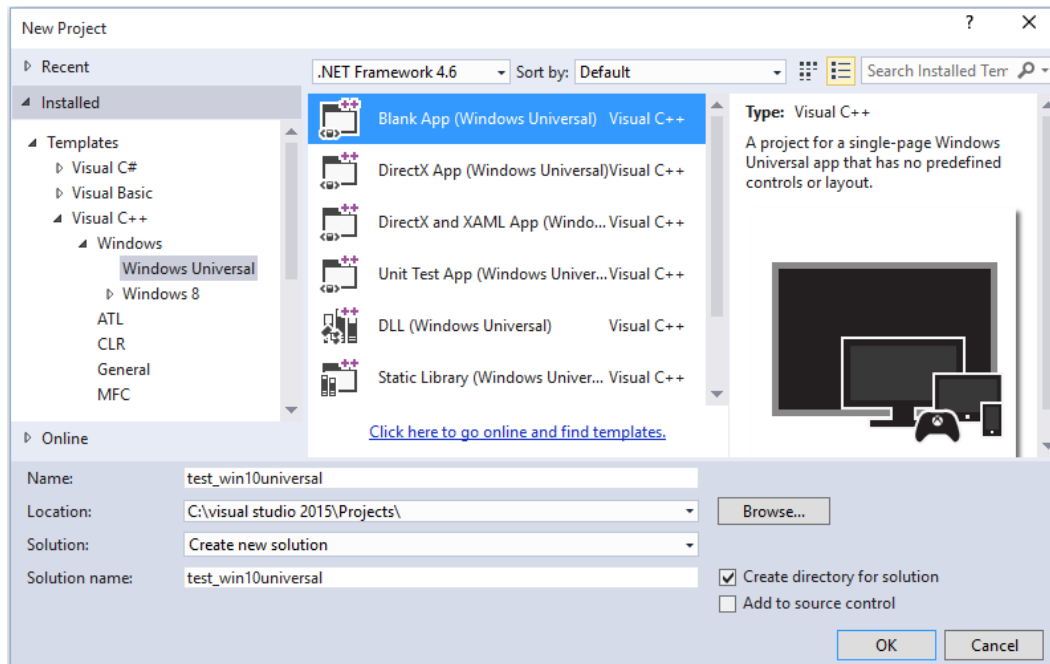
Figure 3-137

To run this project in Visual Studio 2015, please follow the steps below: (In this case, we also use an x86 simulator as an example to run the project)

a) Copy "include" and "lib" folders from the download package to the project folder. Then the directory structure of the test_win10universal project is shown in Figure 3-138.
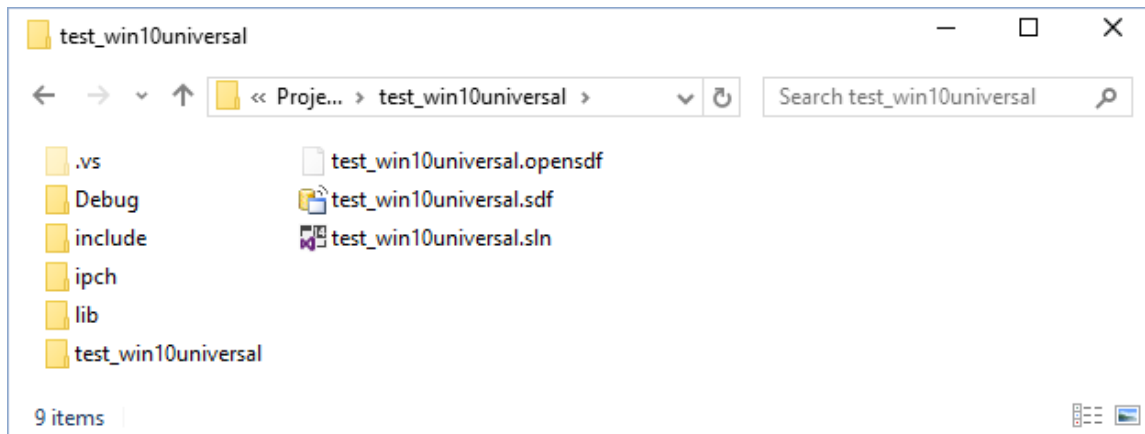
Figure 3-138

b) Change the build architecture of the project. Click on "Build->Configuration Manager" and choose x86 for the active solution platform.

c) Add the libraries in "test_win10universal \lib" folders to the project in Visual Studio 2015. First, right click the "test_win10universal" project, choose "Add->New Filter" to create a new folder

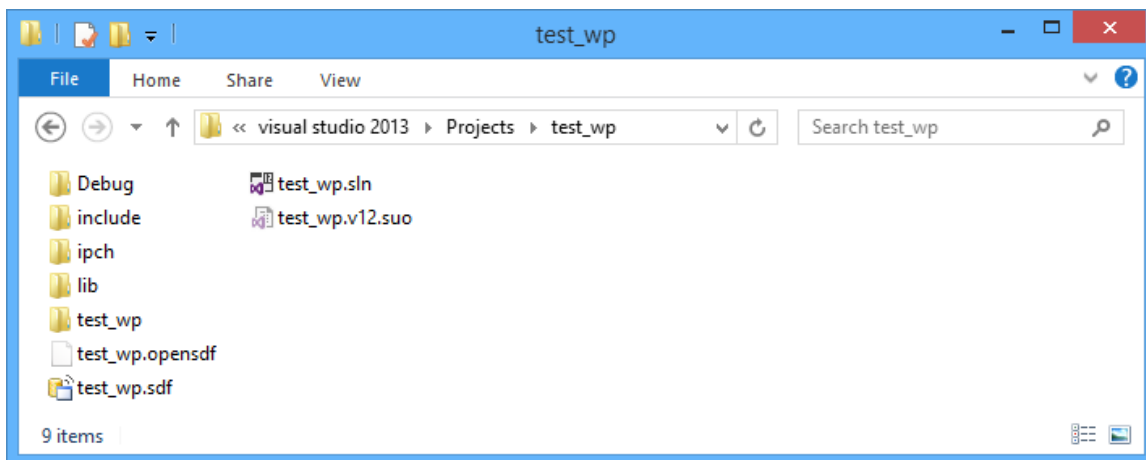named "lib". Then, add the libraries by right clicking the "lib" folder, choosing "Add->Existing Item…" as shown in Figure 3-139, and select the "fsdk_win10_x86.dll" and "fsdk_win10_x86.lib" libraries in "test_win10universal \lib".
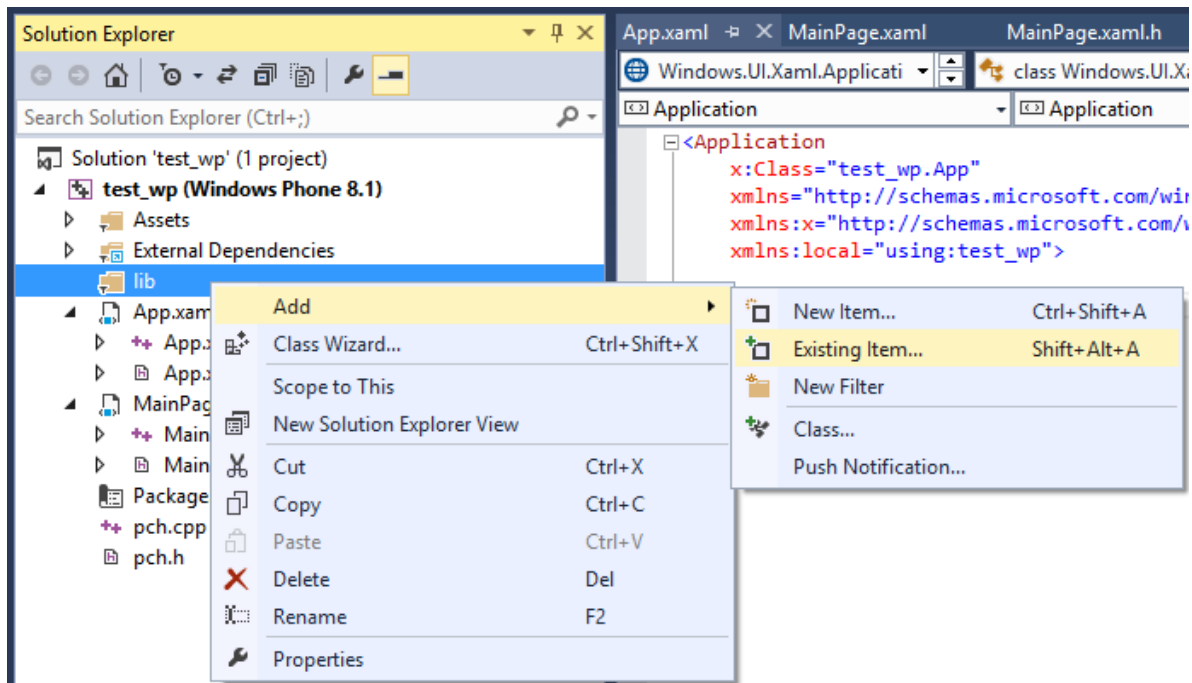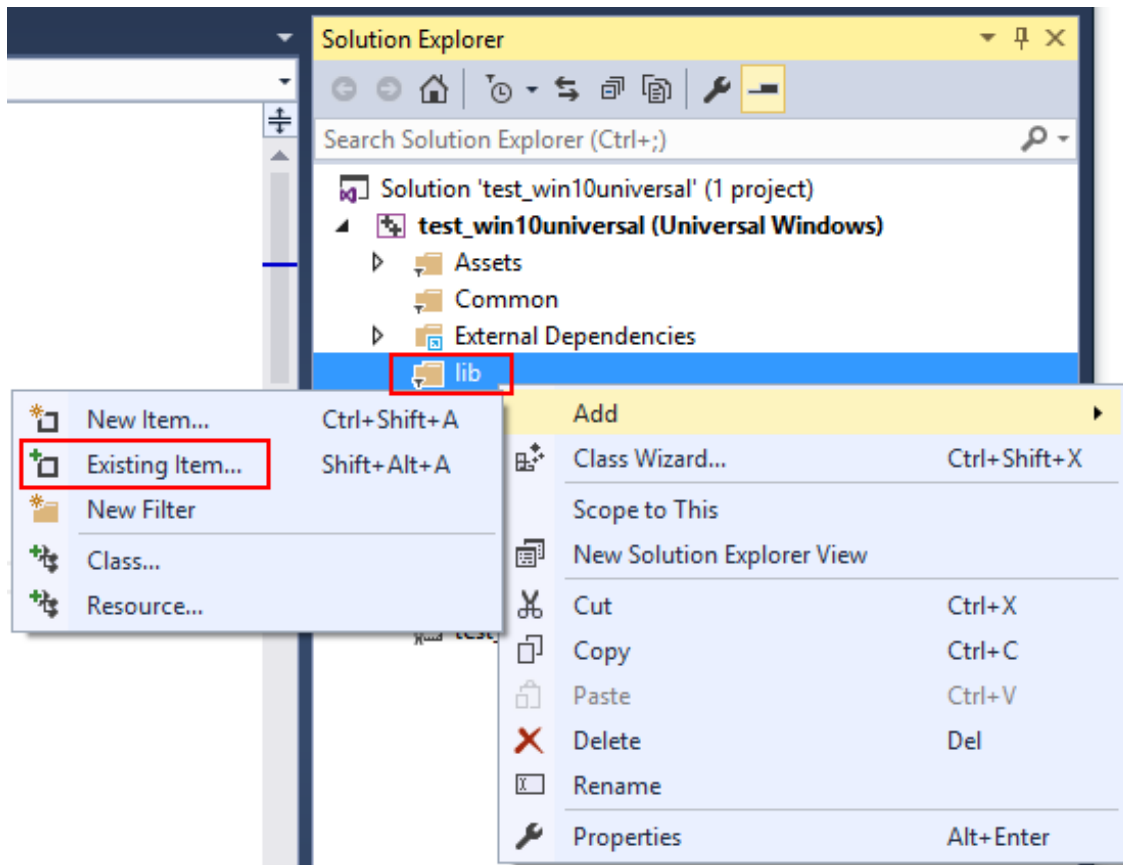

Figure 3-139

d) Set the properties of the libraries. The "Content" property of "fsdk_win10_x86.dll" should be set to "Yes" as shown in Figure 3-140, which deploys the dll library to the simulator along with the ".exe" file to make sure the project can run successfully.
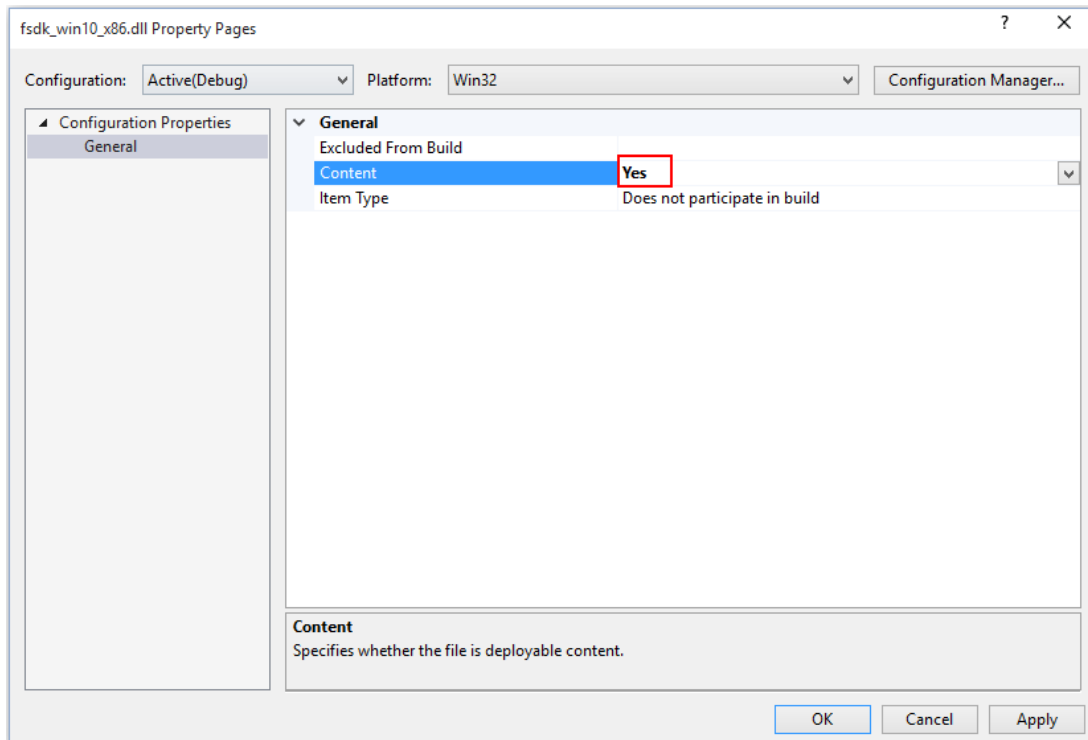
Figure 3-140

Note: If you also added the "fsdk_win10_arm.dll", "fsdk_win10_arm.lib", "fsdk_win10_x64.lib" and "fsdk_win10_x64.dll" to the project, please had better to set the arm and x64 libraries to be excluded when building for Win32. Right click the "fsdk_win10_arm.dll", choose "properties", and set the "Excluded From Build" to "Yes" as shown in Figure 3-141. Do the same settings for the "fsdk_win10_arm.lib", "fsdk_win10_x64.dll" and "fsdk_win10_x64.lib" libraries. After setting the properties, the structure of the project will be like Figure 3-142 .

If you want to run the project in an arm device, you should choose the arm libraries and do the similar settings as above**.** Here, we just introduce one way to configure the libraries, you can refer to this, or use other ways.
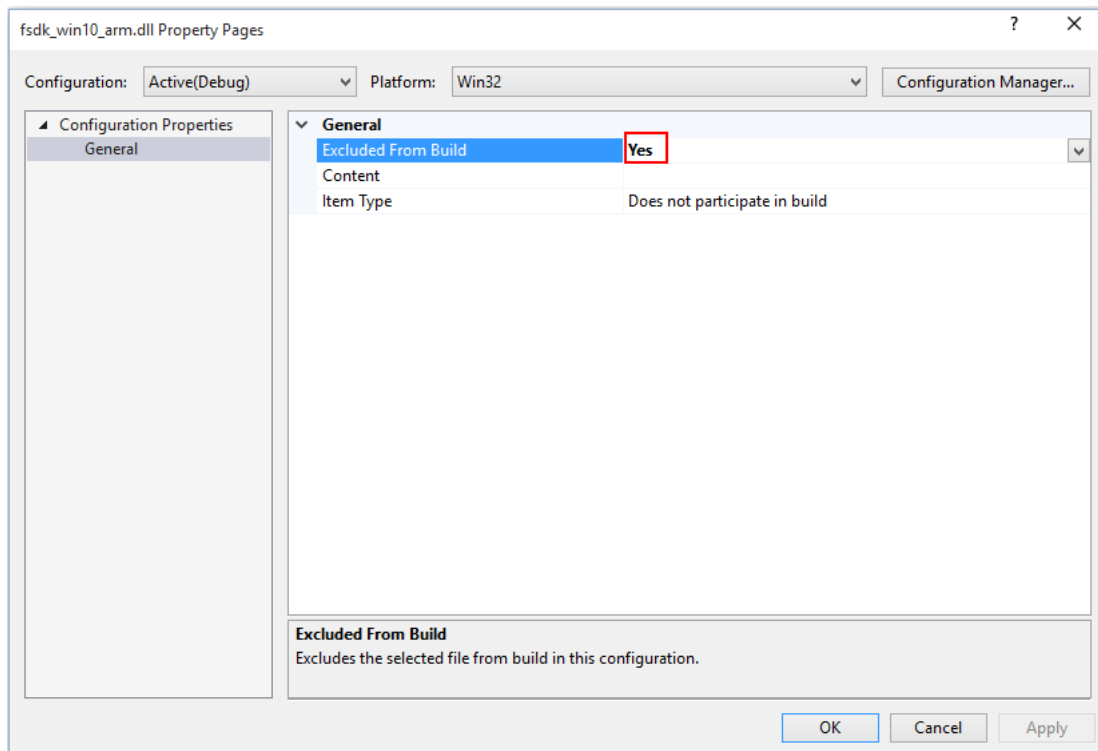
Figure 3-141
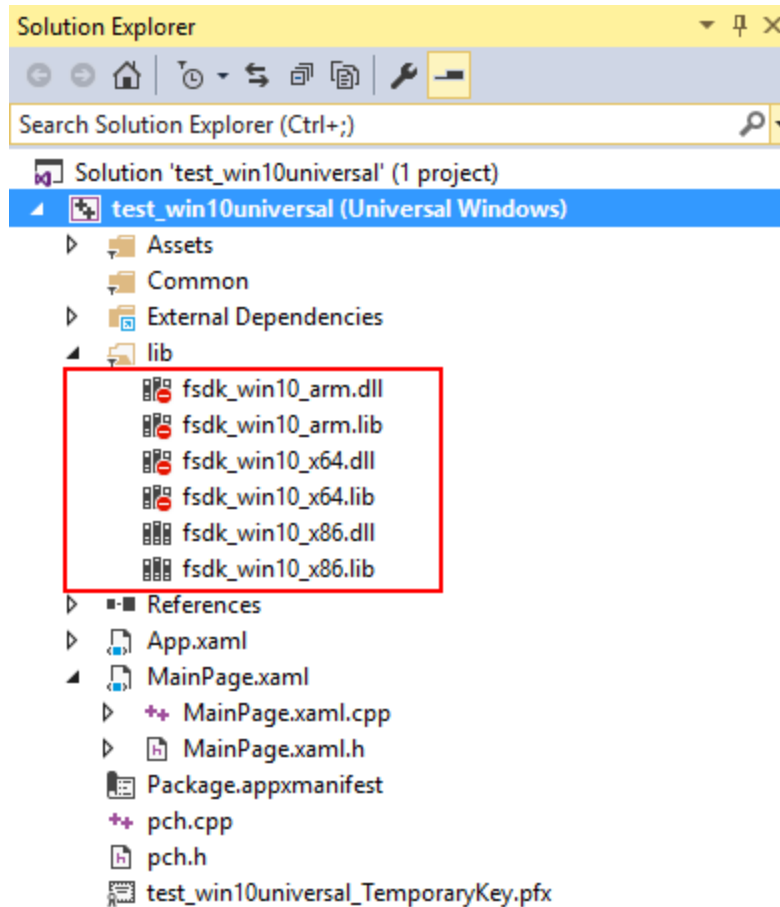
Figure 3-142

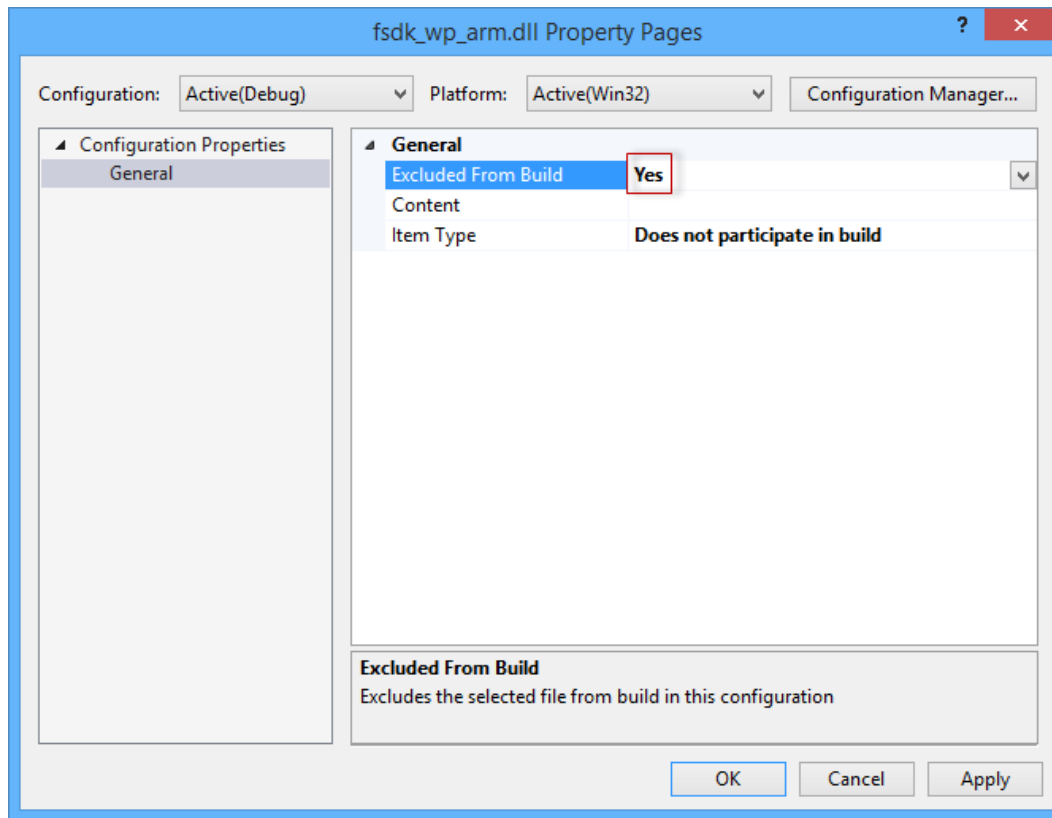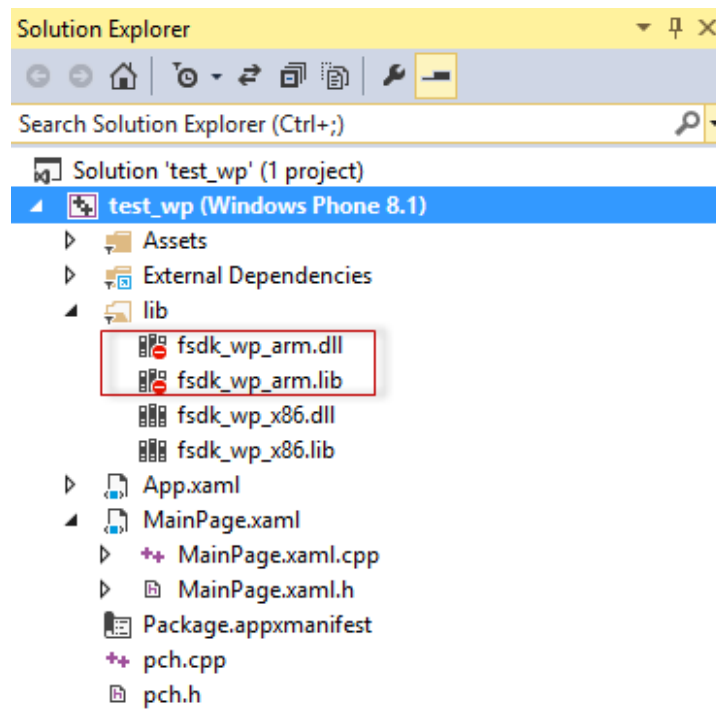e) Construct the code to build a PDF application. Open the "MainPage.xaml", add a button, and set the content to "Open File", which is shown in Figure 3-143. Here, we add a click event "Button_Click", but we did not implement it, you can do this by yourself.

Figure 3-143

f) Open the "MainPage.xaml.h", declare the methods as shown in Figure 3-144.

```
//
// MainPage.xaml.h
// Declaration of the MainPage class.
//

#pragma once

#include "MainPage.g.h"

namespace test_win10universal
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public ref class MainPage sealed
    {
    public:
        MainPage();
        int initLib();
        int applyLicense();
        int pdfOperation();
        int release();

    private:
        void Button_Click(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e);
    };
}
```

Figure 3-144

g) Open the "MainPage.xaml.cpp" and implement the methods defined in "MainPage.xaml.h". The simple implement is shown in Figure 3-145, and you can go on your application in the "pdfOperation()" method. Here we do not elaborate details on how to apply a license (applyLicense() function), which can be referred in section 3.2.4.

Note: please include the "fsdk.h" first.

119

```
#include "MainPage.xaml.h"
#include "../include/fsdk.h"
using namespace ...
// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409
MainPage::MainPage()
{
    InitializeComponent();
    initLib();
    applyLicense();
    pdfOperation();
    release();
}
int MainPage::initLib()
{
    FS_RESULT ret = FSCRT_Library_CreateDefaultMgr();
    return (ret == FSCRT_ERRCODE_SUCCESS) ? 0 : -1;
    return 0;
}
int MainPage::applyLicense()
{
    //The implementation of applying license gose here
    return 0;
}
int MainPage::pdfOperation()
{
    //The implementation of pdf operation gose here
    return 0;
}
int MainPage::release()
{
    FSCRT_Library_DestroyMgr();
    return 0;
}
```

Figure 3-145

h) Click on "Simulator" to build and run the demo. The screenshot of the demo is shown in Figure 3-146.

Figure 3-146

120
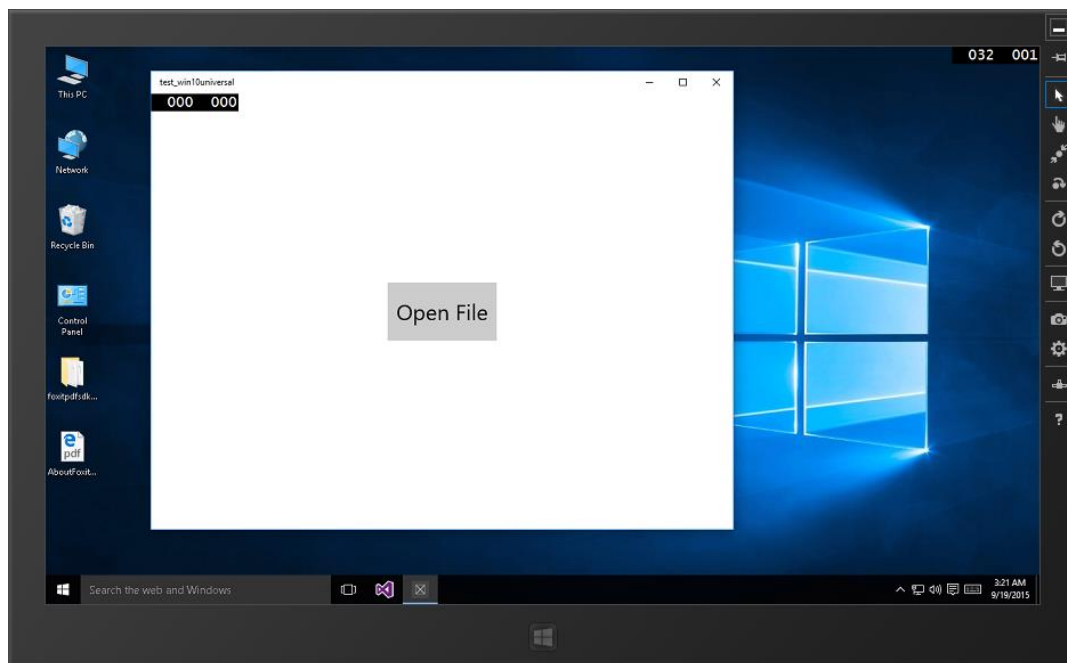
4   WORKING WITH SDK API

In this section, we will introduce a set of major features and list some examples for each feature to show you how to integrate powerful PDF capabilities with your applications using Foxit PDF SDK C API. You can refer to the API reference [2] to get more details about the APIs used in all of the examples.

Note *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**.

4.1   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.

Example:

4.1.1   How to read a file on Windows platform

```
...
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;

...
```

4.2   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.

Example:

4.2.1   How to 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;
...
```

4.3   Attachment

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

Example:

4.3.1   How to 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       attachment;

FS_RESULT ret = FSPDF_Doc_LoadAttachments(pdfDoc, &attachs);
ret = FSPDF_Attachment_Create(pdfdoc,&attachment);
ret = FSPDF_Attachments_CountAttachment(attachs, &count);
ret = FSPDF_Attachments_InsertAttachment(attachs, count, attachment);

FSCRT_BSTRC(filename, "./FoxitText.pdf");
FSCRT_FILE file = NULL;
FS_RESULT ret = FSCRT_File_CreateFromFileName(&filename, FSCRT_FILEMODE_READONLY, &file);
//associating a file to the given attachment.
ret = FSPDF_Attachment_SetFile(attachment, file);
```

```
...
```

4.3.2   How to remove a specific attachment of 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       attachment;

FS_RESULT ret = FSPDF_Doc_LoadAttachments(pdfDoc, &attachs);
ret = FSPDF_Attachments_CountAttachment(attachs, &count);
ret = FSPDF_Attachments_GetAttachment(attachs, 0, &attachment);
ret = FSPDF_Attachments_RemoveAttachment(attachs, attachment);
...
```

4.3.3   How to change the properties of an attachment

```
#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       attachment;

FS_RESULT ret = FSPDF_Doc_LoadAttachments(pdfDoc, &attachs);
ret = FSPDF_Attachments_CountAttachment(attachs, &count);
ret = FSPDF_Attachments_GetAttachment(attachs, 0, &attachment);
FSCRT_BSTR fileName;
FSCRT_BStr_Init(&fileName);
FSCRT_BStr_Set(&bstr, "Hello", 5);
FSPDF_Attachment_SetFileName(m_RecoverAttachment, &fileName, false);
...
```

4.4   Page

PDF Page is the basic and important component of PDF Document. It is represented by **FSCRT_PAGE**
handle object. **FSCRT_PAGE** object is created by **FSPDF_Doc_GetPage** and needs to be cleared by
FSPDF_Page_Clear. Page level APIs provide functions to parse, render, edit (includes creating, deleting
and flattening) a page, and read or set the properties of a page. A PDF page needs to be parsed before it
is rendered or processed for text extraction.

Example:

4.4.1   How to 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);
...
```

4.4.2   How to get page transformation matrix between PDF page coordinates and rendering device coordinates

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...
FS_FLOAT width = 0, height = 0;
FSCRT_MATRIX mt;

FS_RESULT ret = FSPDF_Page_GetSize(page, &width, &height);
// Get page transformation matrix.
ret = FSPDF_Page_GetMatrix(page, 0, 0, width, height, 0, &mt);
...
FSCRT_RECTF    m_srcRect;
FSCRT_RECT     m_destRect;
FS_INT32       m_rotation;

m_srcRect.left = 0;
m_srcRect.top = 100;
m_srcRect.right = 100;
m_srcRect.bottom = 0;

m_destRect.left = 0;
m_destRect.top = 0;
m_destRect.right = 100;
m_destRect.bottom = 100;

m_rotation = 0;
// Get a transformation matrix from PDF page coordinate to device coordinate.
ret = FSPDF_Matrix_TransformPageToDevice(&mt, &m_srcRect, &m_destRect, m_rotation);
...
// Get a transformation matrix from device coordinate to PDF page coordinate.
ret = FSPDF_Matrix_TransformDeviceToPage(&mt, &m_destRect, &m_srcRect, m_rotation);
...
```

4.4.3   How to calculate bounding box of page contents

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...
//Assuming doc has been load
//Assuming page has been load and parsed
```

```
FSCRT_RECTF bBox;
FS_RESULT ret = FSPDF_Page_CalcContentBBox(page, FSPDF_PAGEMARGIN_CONTENTSBBOX, &bBox);
...
```

4.4.4   How to create a PDF 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);
...
```

4.4.5   How to delete a PDF page

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...
// Get page object
FSCRT_PAGE removePage = NULL;
FS_RESULT result = FSPDF_Doc_GetPage(doc, 0, &removePage);
if (result != FSCRT_ERRCODE_SUCCESS) return;
// Delete page from document
result = FSPDF_Page_Delete(removePage);
...
```

4.4.6   How to flatten a PDF page

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...
// Get page object
FSCRT_PAGE flattenPage = NULL;
FS_RESULT result = FSPDF_Doc_GetPage(doc, 0, &flattenPage);
if (result != FSCRT_ERRCODE_SUCCESS) return;
// Flatten a PDF page
result = FSPDF_Page_Flatten(flattenPage, FSPDF_FLATTENFLAG_NOANNOT);
...
```

4.4.7   How to get and set page thumbnails in a PDF document

```
#include "fpdf_page_r.h"
#include "fpdf_page_w.h"
...

FSCRT_BITMAP getThumbnail = NULL;
```

```
FSPDF_Page_GetThumbnail(page, &getThumbnail);

FSCRT_BITMAP newThumbnail = NULL;
FSCRT_Bitmap_Create(100, 100, FSCRT_BITMAPFORMAT_24BPP_BGR, NULL, 0, &newThumbnail);
FSCRT_Bitmap_FillRect(newThumbnail, 0xFF00FF00, NULL);
FSPDF_Page_GetThumbnail(page, newThumbnail);
...
```

4.5   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**.

Example:

4.5.1   How to 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);
...
```

4.5.2　How to 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);
...
```

4.5.3　How to print a page in iOS system

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

//Create a temporary device context based on the device bitmap.
CGContextRef cgContext = CGBitmapContextCreate((void*)pBMPBuffer, width, height, 8,
width*4,colorspace, bmpInfo);

//Create a pdf render based on the device context.
FSCRT_RENDERER render;
FS_RESULT ret = FSCRT_Renderer_CreateOnCGContext(cgContext, FSCRT_APPLEDEVICE_DISPLAY,
&render);
```

```
if (ret != FSCRT_ERRCODE_SUCCESS) return;

//Calculate the page matrix. The matrix will decide the display position and display
size/rotation of the page.
FSPDF_Page_GetMatrix(m_curPage, 0, 0, width, height, 0, &m_matrix);

//Create the render context.
FSPDF_RENDERCONTEXT context;
ret = FSPDF_RenderContext_Create(&context);
if (ret != FSCRT_ERRCODE_SUCCESS) {
        FSCRT_Renderer_Release(render);
        return;
}
//Set the page matrix to render context.
FSPDF_RenderContext_SetMatrix(context, &m_matrix);

//Set the render flag 'FSPDF_RENDERCONTEXTFLAG_ANNOT', so that the annotations on the page
would be also rendered.
//Set the render flag 'FSPDF_RENDERCONTEXTFLAG_LIMITEDIMAGECACHE', so that it would take less
memory when the page contains too many images.
FSPDF_RenderContext_SetFlags(context,
FSPDF_RENDERCONTEXTFLAG_ANNOT|FSPDF_RENDERCONTEXTFLAG_LIMITEDIMAGECACHE);

//Start to render the page content.
FSCRT_PROGRESS progress;
ret = FSPDF_RenderContext_StartPage(context, render, m_curPage, FSPDF_PAGERENDERFLAG_NORMAL,
&progress);
if (ret != FSCRT_ERRCODE_SUCCESS) {
        FSPDF_RenderContext_Release(context);
        FSCRT_Renderer_Release(render);
        return;
}

//Continue rendering the page content, until the rendering progress is finished. It could be
interrupted if it takes too much time.
ret = FSCRT_ERRCODE_TOBECONTINUED;
while (ret == FSCRT_ERRCODE_TOBECONTINUED || ret == FSCRT_ERRCODE_ROLLBACK) {
  ret = FSCRT_Progress_Continue(progress, NULL);
}
FSCRT_Progress_Release(progress);
...
```

4.6   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).

Example:

4.6.1   How to extract text from a PDF

```c
#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 (ret != FSCRT_ERRCODE_SUCCESS) return;

FS_INT32 count = -1;
ret = FSPDF_TextPage_CountChars(textPage, count);
if (ret != FSCRT_ERRCODE_SUCCESS) return;

ret = FSPDF_TextPage_GetChars(textPage, 0, count-1, &bstrChars);
if (ret != FSCRT_ERRCODE_SUCCESS) return;
...
```

4.6.2   How to select text of a rectangle area in a PDF

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

FSCRT_RECTF rcRectF;
rcRectF.left = 90;
rcRectF.right = 450;
rcRectF.top = 595;
rcRectF.bottom = 580;
FS_RESULT ret = FSCRT_ERRCODE_SUCCESS;
FSPDF_TEXTPAGE textPage = NULL;
FSPDF_TEXTSELECTION textSelection = NULL;

ret = FSPDF_TextPage_Load(pdfPage, &textPage);
if (ret != FSCRT_ERRCODE_SUCCESS) return;

ret = FSPDF_TextPage_SelectByRectangle(textPage, &rcRectF, &textSelection);
if (ret != FSCRT_ERRCODE_SUCCESS) return;
...
```

4.6.3   How to search a text pattern in a page

```c
#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);
                ...
        }
}
```

4.6.4   How to retrieve hyperlinks in a PDF page

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

FSPDF_TEXTLINK textLink;
FSPDF_TEXTPAGE textPage;
FS_RESULT ret = FSPDF_TextPage_Load(page, &textPage);
if (ret != FSCRT_ERRCODE_SUCCESS) return;

Ret = FSPDF_TextPage_ExtractLinks(textPage, &textLink);
if (ret != FSCRT_ERRCODE_SUCCESS) return;
...
```

4.7   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 to display the form data.

Example:

4.7.1   How to load the forms in a PDF

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

FSPDF_FORM form = NULL;
FS_BOOL hasForm = 0;
FS_RESULT ret = FSPDF_Doc_HasForm(pdfDoc, &hasForm);
if(hasForm)
        ret = FSPDF_Form_Load(pdfoc, &form);
```

4.7.2 How to count form fields and get the properties

```c
#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);
                ...
        }
}
```

4.7.3 How to export the form data in a PDF to a FDF file

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

FSPDF_FORM form = NULL;
FS_RESULT ret = FSPDF_Form_Load(pdfDoc, &form);
ret = FSFDF_Doc_Create(FSCRT_DOCUMENTTYPE_FDF, &fdfDoc);
ret = FSPDF_Form_ExportToFDFDoc(form, fdfDoc)
if (ret != FSCRT_ERRCODE_SUCCESS) return;
...
```

4.7.4 How to import form data from a FDF file

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

FSPDF_FORM form = NULL;
FSCRT_DOCUMENT fdfDoc = NULL;
```

```
FSCRT_FILE fdfFile = NULL;

FSCRT_BSTRC(filename, "./FoxitText.fdf");
FS_RESULT ret = FSCRT_File_CreateFromFileName(&filename, FSCRT_FILEMODE_READONLY, &fdffile);
if (ret != FSCRT_ERRCODE_SUCCESS) return;

ret = FSPDF_Form_Load(pdfDoc, &form);
ret = FSFDF_Doc_Load(fdfFile, &fdfDoc);
ret = FSPDF_Form_ImportFromFDFDoc(form, fdfDoc)
if (ret != FSCRT_ERRCODE_SUCCESS) return;
...
```

4.7.5   How to get and set the properties of form fileds

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

FSCRT_BSTR bsValue;
FSCRT_BStr_Init(&bsValue);
bsValue.len = strlen("Default value");
bsValue.str = "Default value";

ret = FSPDF_Form_Load(pdfDoc, &form);

FSCRT_BSTR bsFieldName;
FSCRT_BStr_Init(&bsFieldName);
FSCRT_BStr_Set(&bsFieldName, "123", 3);
FSPDF_FormField_SetValue(form, &bsFieldName, &bsValue);

FSCRT_BSTR value;
FSCRT_BStr_Init(&value);
FSPDF_FormField_GetValue(form, &bsFieldName, &value);
...
```

4.8   Form Filler

Form filler is the most commonly used feature for users. Form filler allows 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.

Example:

4.8.1   How to 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);
...
```

4.9   Form Design

Fillable PDF forms (AcroForm) are especially convenient for preparation of various applications, such as taxes and other government forms. Form design provides APIs to add or remove form fields (Acroform) to or from a PDF file. Designing a form from scratch allows developers to create the exact content and layout of the form they want.

Example:

4.9.1   How to add a text form field to a PDF

```
#include "fpdf_form_r.h"
#include "fpdf_form_w.h"

//Assuming FSCRT_DOCUMENT document has been loaded.
//Assuming FSCRT_PAGE page has been got.
//Assuming the annnots in the page have been loaded.

FSPDF_FORM form = NULL;
ret = FSPDF_Doc_CreateForm(document, form);
FSPDF_FORMCONTROL formControl = NULL;
FSCRT_RECTF rect;
rect = setRect(191.5, 711.5, 281.5, 693.5)
FSCRT_BSTR bstrFieldName;
FSCRT_BStr_Init(&bstrFieldName);
ret = FSCRT_BStr_Set(&bstrFieldName,"Text Field", strlen("Text Field"));

//add a text form field
ret =
FSPDF_Form_AddField(form,page,&bstrFieldName,FSPDF_FORMFIELDTYPE_TEXTFIELD,&rect,formControl);
FSCRT_BStr_Clear(&bstrFieldName);
FSCRT_BSTR bstrValue;
```

```
FSCRT_BStr_Init(&bstrValue);
ret = FSCRT_BStr_Set(&bstrValue, "Foxit", strlen("Foxit"));
ret = FSPDF_FormField_SetValue(form, &bstrFieldName, &bstrValue);
FSCRT_BStr_Clear(&bstrValue);
```

4.9.2   How to remove a text form field from a PDF

```
#include "fpdf_form_r.h"
#include "fpdf_form_w.h"
...
//Assuming FSCRT_DOCUMENT document has been loaded.
//Assuming FSCRT_PAGE page has been got.
//Assuming the annnots in the page have been loaded.

FS_RESULT ret = FSPDF_Form_Load(document, &form);
FSCRT_BSTR fieldName;
FSCRT_BStr_Init(&fieldName);
FSCRT_BStr_Set(&fieldName,"textField",strlen("textField"));
FSPDF_Form_RemoveField(form, &fieldName);
...
```

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 4-1. 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 4-1 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.

Table 4-1

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 reference [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.

Example:

4.10.1.1 How to add a link annotation to another page in the same PDF

```
#include "fpdf_annot_r.h"
#include "fpdf_annot_w.h"
...
//Assuming FSCRT_DOCUMENT document has been loaded.
//Assuming FSCRT_PAGE page has been got.
//Assuming the annnots in the page have been loaded.

FSCRT_BSTR annotType;
FSCRT_BStr_Init(&annotType);
FSCRT_BStr_Set(&annotType, FSPDF_ANNOTTYPE_LINK, strlen(FSPDF_ANNOTTYPE_LINK));
FSCRT_RECTF rect = {0, 100, 100, 0};
```

```
FSCRT_ANNOT annot = NULL;
FS_RESULT ret = FSPDF_Annot_Add(page, &rect, &annotType, NULL, 0, &annot);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;
...
```

4.10.1.2   How to 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);
...
```

4.10.1.3   How to set the popup information when creating markup annotations

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

//Create a new square annot
FSCRT_RECTF rect = {100, 200, 140, 150};
FSCRT_BSTRC(bsAnnotType, FSPDF_ANNOTTYPE_SQUARE);
FSCRT_ANNOT annot = NULL;
FSPDF_Annot_Add(pdfPage, &rect, &bsAnnotType, NULL, 0, &annot);
//Set some properties for the new square annot and reset its ap.
...
//Create a new popup annot and set to the new square annot.
FSCRT_BSTRC(bsPopupType, FSPDF_ANNOTTYPE_POPUP);
FSCRT_RECTF newPopup_rect = {0, 100, 50, 30}
FSCRT_ANNOT newPopupAnnot = NULL;
FSPDF_Annot_Add(pdfPage, &newPopup_rect, &bsPopupType, NULL, 0, &newPopupAnnot);
FSPDF_Annot_SetOpenStatus(newPopupAnnot, true);
```

```
FSPDF_Annot_SetPopup(annot, newPopupAnnot);
```

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.

Example:

4.10.2.1   How to 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;

//Assuming 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);

//Assuming 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);
        ...
}

//Assuming 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 functionalities like image creation and image conversion which supports the following image formats:

BMP, TIFF, PNG, JPX, JPEG, and GIF. Foxit PDF SDK can make the conversion between PDF files and the supported image formats except for GIF. It only supports converting GIF images to PDF files.

Example:

4.11.1   How to 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);
        //assuming pages are parsed before they are handled further.
        ...
        //assuming 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);
        ...
}
...
```

4.11.2   How to convert png file to PDF file

```
#include "fpdf_image_r.h"
#include "fpdf_image_w.h"

FSCRT_DOCUMENT pdfDoc;
FSCRT_PAGE page;
FSPDF_Doc_Create(&pdfDoc);
FSPDF_Page_Create(pdfDoc, 0, &page);
FSPDF_Page_SetSize(page, width, height);
FS_RESULT ret = FSCRT_ERRCODE_ERROR;

FSPDF_PAGEOBJECT imageObj;
ret = FSPDF_ImageObject_Create(page, &imageObj);

FSCRT_FILE file = NULL;
FSCRT_BSTR imageFileName;
FSCRT_BStr_Init(&imageFileName);
FSCRT_BStr_Set(&imageFileName, "image.png", strlen("image.png"));
ret = FSCRT_File_CreateFromFileName(&imageFileName, FSCRT_FILEMODE_READONLY, &file);

FSCRT_IMAGE image = NULL;
FSCRT_Image_LoadFromFile(file, &image);
ret = FSCRT_Image_LoadFrame(image, 0);
ret = FSPDF_ImageObject_SetImage(page, imageObj, image);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;

FSCRT_MATRIX matrix = {114, 0, 0, 105, 289, 278};
FSPDF_ImageObject_SetMatrix(page, imageObj, &matrix);
FSPDF_PAGEOBJECTS pageObjs;
FSPDF_Page_GetPageObjects(page, &pageObjs);
ret = FSPDF_PageObjects_InsertObject(page, pageObjs, FSPDF_PAGEOBJECT_IMAGE, 0, imageObj);
ret = FSPDF_PageObjects_GenerateContents(page, pageObjs);

FSCRT_PROGRESS progress;
FSCRT_FILE saveFile = NULL;
FSCRT_BSTR bstrSavePath;
FSCRT_BStr_Init(&bstrSavePath);
FSCRT_BStr_Set(&bstrSavePath, "image.pdf", strlen("image.pdf"));
ret = FSCRT_File_CreateFromFileName(&bstrSavePath, FSCRT_FILEMODE_TRUNCATE, &saveFile);
ret = FSPDF_Doc_StartSaveToFile(pdfDoc, saveFile, FSPDF_SAVEFLAG_NOORIGINAL, &progress);
ret = FSCRT_Progress_Continue(progress, 0);
ret = FSCRT_Progress_Release(progress);
ret = FSCRT_File_Release(saveFile);
...
```

4.12 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.

Example:

4.12.1 How to 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);

...
```

4.12.2 How to create an image watermark and insert it into the first page

```
#include "fpdf_watermark_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded
//Assuming and the first page has been loaded and parsed
...
FSCRT_FILE bmpImageFile;
...
FSCRT_IMAGE image = NULL;
FSCRT_Image_LoadFromFile(bmpImageFile, &image);

FSPDF_WATERMARK imagewatermark;
```

```
FSPDF_WATERMARK_SETTINGS settings = {FSPDF_WATERMARKPOS_CENTER, 0, 0,
FSPDF_WATERMARKFLAG_ONTOP | FSPDF_WATERMARKFLAG_ASANNOT, 0.25f, 0.25f, 0, 100};
FSPDF_Watermark_CreateFromImage(doc, image, &settings, &imagewatermark);
FSPDF_Watermark_InsertToPage(imagewatermark, page);
//Save document to file
...
FSPDF_Watermark_Release(imagewatermark);
FSCRT_Image_Release(image);
...
```

4.12.3 How to remove a specified watermark from a page

```
#include "fpdf_watermark_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded
//Assuming and the first page has been loaded and parsed
...
FSPDF_Watermark_GetCount(page, &count);
FSPDF_Watermark_Remove(page, count - 1);
...
//Save document to file
...
```

4.12.4 How to remove all watermarks from a page

```
#include "fpdf_watermark_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded
//Assuming and the first page has been loaded and parsed
...
FSPDF_Page_RemoveWatermarks(page);
...
//Save document to file
...
```

4.13 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 4-2.

Table 4-2

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

Example:

4.13.1 How to 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();
...
```

4.14 Security

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

Example:

4.14.1 How to 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);
}
...
```

4.14.2 How to encrypt a PDF file with Certificate

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

FSCRT_BSTR envolops;
FSCRT_BSTR initialKey;
FS_BOOL bEncryptMetadata = FALSE;
FSCRT_FILE file;
FSCRT_PROGRESS progress;
FSPDF_SECURITYHANDLER securityhandler;
// Used CPSL_CryptLib to get related info of certFile
GetCertInfo(certFile, &envolops, &initialKey, bEncryptMetadata, 16);
// load encrypt file
FSCRT_FILE file;
...
// start encrypt file
ret = FSPDF_Security_StartCertificateEncryption(pdfDoc, &envolops, 1, FSCRT_CIPHER_AES,
&initialKey, bEncryptMetadata, file, FSPDF_SAVEFLAG_NOORIGINAL, &progress);
FSCRT_Progress_Continue(encryptProgress, NULL);
FSCRT_Progress_Release(encryptProgress);
...
// release
FSCRT_File_Release(file);
...
```

4.14.3 How to encrypt a PDF file with Foxit DRM

```
#include "fpdf_security_r.h"
#include "fpdf_security_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded

FSCRT_BSTRC(filter, "FoxitDRM");
FSCRT_BSTRC(fileID, "FileID");
FSCRT_BSTRC(initialKey, "123");
FSCRT_FILE encryptFile = NULL;
FSCRT_PROGRESS progress = NULL;

// load encrypt file
FSCRT_FILE encryptFile;
...
// start encrypt file
ret = FSPDF_Security_StartFoxitDRMEncryption(pdfDoc, &filter, TRUE, 0xFFFFFFFC,
FSCRT_CIPHER_AES, 16, &fileID, &initialKey, TRUE, encryptFile,
```

```
                        FSPDF_SAVEFLAG_NOORIGINAL, &progress);
FSCRT_Progress_Continue(encryptProgress, NULL);
FSCRT_Progress_Release(encryptProgress);

// release
FSCRT_File_Release(encryptFile);
...
```

4.15 RMS

RMS (Right Management Service) module can be used to encrypt and decrypt PDF documents with Microsoft RMS Encryption/Decryption. Foxit PDF SDK provides APIs to integrate with the third-party security mechanism (Microsoft RMS). These APIs allow developers to work with the Microsoft RMS SDK to both encrypt (protect) and decrypt (unprotect) PDF documents.

Example:

4.15.1   How to encrypt a PDF document with Microsoft RMS

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

FSCRT_BSTR bstrfilter;
FSCRT_BStr_Init(&bstrfilter);
FSCRT_BStr_Set(&bstrfilter, "MicrosoftIRMServices", strlen("MicrosoftIRMServices"));
FSPDF_SECURITYHANDLER securityHandler;
//set the callback function.
...

FSPDF_Security_RegisterHandler(&bstrfilter, &securityHandler);
FS_LPCSTR g_DynamicWMArray[4] = { "WM-1", "This document has been encrypted by RMS
encryption.", "WM-2", "Just for testing dynamic watermark." };

//array
FSCRT_BSTR strServerEULList[4];

//use function ::FSCRT_Flate_Compress to compress <b>g_DynamicWMArray</b>.
//Then use function ::FSCRT_Base64_EncodeFromBuffer encode the result to
<b>strServerEULList</b>.
...

FS_LPCSTR g_LicenseData = "0123456789";
FSCRT_BSTR publicLicense;
FSCRT_BStr_Init(&publicLicense);

//The way to set the content of publicLicense is same with <b>strServerEULList</b>.
...

FSCRT_FILE encryptFile;
```

```
//load output file.
...

FSPDF_Security_StartRMSEncryption(doc, &publicLicense, strServerEULList, 4, 1, false,
encryptFile, FSPDF_SAVEFLAG_INCREMENTAL, &encryptProgress);

ret = FSCRT_ERRCODE_TOBECONTINUED;
while (ret == FSCRT_ERRCODE_TOBECONTINUED)
{
        ret = FSCRT_Progress_Continue(encryptProgress,NULL);
}
FSCRT_Progress_Release(encryptProgress);
```

4.15.2   How to decrypt a PDF document with Microsoft RMS

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

FSCRT_BSTR bstrfilter;
FSCRT_BStr_Init(&bstrfilter);
FSCRT_BStr_Set(&bstrfilter, "MicrosoftIRMServices", strlen("MicrosoftIRMServices"));
FSPDF_SECURITYHANDLER securityHandler;
//set the callback function.
...

FSPDF_Security_RegisterHandler(&bstrfilter, &securityHandler);
FS_LPCSTR g_DynamicWMArray[4] = { "WM-1", "This document has been encrypted by RMS
encryption.", "WM-2", "Just for testing dynamic watermark." };

//array
FSCRT_BSTR strServerEULList[4];

//use function ::FSCRT_Flate_Compress to compress <b>g_DynamicWMArray</b>.
//Then use function ::FSCRT_Base64_EncodeFromBuffer encode the result to
<b>strServerEULList</b>.
...

FS_LPCSTR g_LicenseData = "0123456789";
FSCRT_BSTR publicLicense;
FSCRT_BStr_Init(&publicLicense);

//The way to set the content of publicLicense is same with <b>strServerEULList</b>.
...

FSCRT_FILE encryptFile;

//load output file.
...

FSPDF_Security_StartRMSEncryption(doc, &publicLicense, strServerEULList, 4, 1, false,
encryptFile, FSPDF_SAVEFLAG_INCREMENTAL, &encryptProgress);
```

```
ret = FSCRT_ERRCODE_TOBECONTINUED;
while (ret == FSCRT_ERRCODE_TOBECONTINUED)
{
        ret = FSCRT_Progress_Continue(encryptProgress,NULL);
}
FSCRT_Progress_Release(encryptProgress);
// release encryptFile and doc
...

//load rms encrypt file and open.
...
//load rms decrypt file.
FSCRT_FILE decryptFile;
...
FSPDF_Doc_StartSaveToFile(doc, decryptFile, FSPDF_SAVEFLAG_NOORIGINAL , decryptProgress)
ret = FSCRT_ERRCODE_TOBECONTINUED;
while (ret == FSCRT_ERRCODE_TOBECONTINUED)
{
        ret = FSCRT_Progress_Continue(decryptProgress,NULL);
}
FSCRT_Progress_Release(decryptProgress);
// release decryptFile and doc
...
```

4.16 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.

Example:

4.16.1   How to create a reflow page and render it to a bmp file.

```
#include "fpdf_reflow_r.h"
//Assuming a pdfPage has been opened here.
//Assuming 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;
}
...
```

4.17 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.

Example:

4.17.1   How to open and parse pages with asynchronous mode.

```
#include "fpdf_async_r.h"
... //Assuming 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
        }
}
...
```

4.18 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.

Example:

4.18.1   How to 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);
...
```

4.19 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.

Example:

4.19.1   How to 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)
        {
                //...
        }
}
...
```

4.20 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 4.21) which are associated with a specific page each. Foxit PDF SDK provides APIs to create, modify, retrieve and delete these objects in a document.

Example:

4.20.1 How to 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);
...
```

4.21 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 4.20 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 footers to PDF documents, adding an image logo to each page, or generating a template PDF on demand.

Example:

4.21.1 How to create a text object in a PDF 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);


        ...
}
...
```

4.21.2   How to add an image logo to each page of a PDF

```
#include "fpdf_pageobjects_r.h"
#include "fpdf_pageobjects_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded

FSPDF_PAGEOBJECT imageObj;
ret = FSPDF_ImageObject_Create(page, &imageObj);

FSCRT_FILE file = NULL;
FSCRT_BSTR imageFileName;
FSCRT_BStr_Init(&imageFileName);
FSCRT_BStr_Set(&imageFileName, "image.png", strlen("image.png"));
ret = FSCRT_File_CreateFromFileName(&imageFileName, FSCRT_FILEMODE_READONLY, &file);

FSCRT_IMAGE image = NULL;
FSCRT_Image_LoadFromFile(file, &image);
ret = FSCRT_Image_LoadFrame(image, 0);
ret = FSPDF_ImageObject_SetImage(page, imageObj, image);
if (FSCRT_ERRCODE_SUCCESS != ret) return ret;

FSCRT_MATRIX matrix = {114, 0, 0, 105, 289, 278};
FSPDF_ImageObject_SetMatrix(page, imageObj, &matrix);

for (int i = 0; i < count; i++)
{
        FSPDF_Doc_GetPage(pdfDoc, i, &page);
        FSPDF_PAGEOBJECTS pageObjs;
        FSPDF_Page_GetPageObjects(page, &pageObjs);
        ret = FSPDF_PageObjects_InsertObject(page, pageObjs, FSPDF_PAGEOBJECT_IMAGE, 0,
imageObj);
        ret = FSPDF_PageObjects_GenerateContents(page, pageObjs);
}
//Save document to file
...
```

4.22 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 could be found in chapter 10.5 of PDF reference 1.7 [1].

Example:

4.22.1 How to 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);
...
```

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

As described in "What is 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.

4.23.1 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.

4.23.2   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.

4.23.2.1   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.

4.23.2.2   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.

4.23.3 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 4.23.2.2 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:

4.23.3.1 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);
        ...
}
```

4.23.3.2 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);
//...
```

4.23.3.3   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);
//....
```

4.24 Layer

PDF Layers, in other words, Optional Content Groups (OCG), are supported in Foxit PDF SDK. Users can selectively view or hide the contents in different layers of a multi-layer PDF document. Multi-layers are 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. Foxit PDF SDK provides APIs to view or hide the contents in different layers, set layers' name, add or remove layers, and edit layers.

Example:

4.24.1   How to create a PDF layer

```
#include "fpdf_layer_w.h"
#include "fpdf_layer_r.h"
...
//Assuming PDFDocument pdfDoc has been loaded
//Assuming page has been loaded and parsed

FSPDF_LAYER layer;
FSCRT_BSTR layerName;
FSCRT_BStr_Init(&layerName);
FSCRT_BStr_Set(&layerName, "layer", strlen("layer");
FSPDF_Doc_AddLayer(doc,&LayerName,&layer);
...
//add pageobject to layer
FSPDF_PAGEOBJECTS pageobjects;
FSPDF_PAGEOBJECT obj;
FSPDF_Page_GetPageObjects(page, &pageobjects);
FSPDF_PageObjects_GetObject(page, pageobjects, FSPDF_PAGEOBJECT_ALL, 0, &obj);
FSPDF_Layer_AddPageObject(layer, page, obj);
FSPDF_PageObjects_GenerateContents(page, pageobjects);
...
```

4.24.2   How to set the name of an existing layer

```
#include "fpdf_layer_r.h"
#include "fpdf_layer_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded

FSPDF_LAYER layer;
FSCRT_BSTR layerName;
FSCRT_BStr_Init(&layerName);
FSCRT_BStr_Set(&layerName, "layername", strlen("layername");
FSPDF_LAYERNODE layers;
```

```
FSPDF_LayerNode_Init(&layers);
FSPDF_Doc_EnumLayers(pdfDoc, &layers);
FSPDF_Layer_SetName(layers.layer, &layerName);
...
```

4.24.3 How to traverse layer tree and set the opposite visible state of every PDF layer

```
void traverseLayer(FSPDF_LAYERCONTEXT context, FSPDF_LAYERNODE* pLayers)
{
        FSCRT_BSTR layerName;
        FS_RESULT nRet = FSCRT_BStr_Init(&layerName);
        nRet = FSPDF_Layer_GetName(pLayers->layer, &layerName);
        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);
        }
        for (int i = 0; i < pLayers->count; i ++)
        {
                traverseLayer(context, &(pLayers->children[i]));
        }
}

traverseLayer(context, pLayers);
```

4.24.4 How to remove a specific layer

```
#include "fpdf_layer_r.h"
#include "fpdf_layer_w.h"
...
//Assuming PDFDocument pdfDoc has been loaded

// Retrieve a specific layer
FSPDF_LAYERNODE retrieve(FSPDF_LAYERNODE p, char *name)
{
        if (strncmp(p->name.str, name, p->name.len) == 0)
        return p;
        for (int i = 0; i < p->count; i ++)
        {
                FSPDF_LAYERNODE* layer = retrieve(&(p->children[i]), name);
                if (layer)
                return layer;
```

```
        }
        return NULL;
}


FSPDF_LAYERNODE layers;
FSPDF_LAYERNODE* layerNode;
FSPDF_LayerNode_Init(&layers);
FSPDF_Doc_EnumLayers(doc, &layers);
//Get a layer of specific name
layerNode = retrieve(&layers, "View:Hello");

FSPDF_Layer_Remove(layerNode->layer, 0);
...
```

4.25 Signature

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

Note: *The Signature module only provides the third-party signature interface and requires the customers have their own signature implementation. If you want to purchase Foxit PDF SDK license and use any functions of this module, please contact Foxit to enable this module explicitly.*

Example:

4.25.1   How to sigh the PDF document with a signature

```
//Third-party handler register their handler Filter, SubFilter and handler to SDK.
//Register the global signature handler, in order to implement signature functions.
FSPDF_SIGNATUREHANDLER sigHandler;
sigHandler.clientData = savefile;
sigHandler.StartCalcDigest = _GlobalStartCalcDigest;
sigHandler.ContinueCalcDigest = _GlobalContinueCalcDigest;
sigHandler.FinishCalcDigest = _GlobalFinishCalcDigest;
sigHandler.Sign = _GlobalSign;
sigHandler.Verify = _GlobalVerify;
FSCRT_BSTRC(filter, "Adobe.PPKLite");
FSCRT_BSTRC(subFilter, "adbe.pkcs7.detached");
ret = FSPDF_Signature_RegisterHandler(&filter, &subFilter, &sigHandler);
if (ret != FSCRT_ERRCODE_SUCCESS)
{
        fclose(savefile);
        return ret;
}
```

```
//Sign the document by the custom way in progressive manner.
FSCRT_PROGRESS signProgress = NULL;
FSCRT_FILE savedFile = FSDK_OpenFile(strSavefile.c_str(), "wb+");
ret = FSPDF_Signature_StartSign(sign, savedFile, &signProgress);
```

4.25.2   How to implement signature callback function of signing on Windows

```
FS_RESULT _GlobalSign(FS_LPVOID clientData, FS_LPVOID context, FSPDF_SIGNATURE sig, const
FSCRT_BSTR* digest, FSCRT_BSTR* signedData)
{
        //Get the digest context.
        CFSCRT_DigestContext* pContext = CFSCRT_DigestContext::GetDigestContext();
        //Check validation of signed file.
        FILE* pFile = (FILE*)clientData;
        if (!pFile)
        {
                pContext->Release();
                return FSCRT_ERRCODE_ERROR;
        }

        //Open certificate file, read data to memory.
        string strCertFile = FSDK_GetInputFilesFolder("pdfdigitalsignature");
        void* dir = FSDK_OpenFolder(strCertFile.c_str(),FALSE);
        if(!dir)
        {
                strCertFile = FSDK_GetFixFolder();
        }
        else
                FSDK_CloseFolder(dir);
        strCertFile += "/";
        strCertFile += "foxit_all.pfx";
        FILE* fpCert = fopen(strCertFile.c_str(), "rb");
        if(!fpCert)
        {
                pContext->Release();
                return FSCRT_ERRCODE_ERROR;
        }

        fseek(fpCert, 0, SEEK_END);
        long cbData =  ftell(fpCert);
        FS_BYTE* pbData = new FS_BYTE[cbData];
        fseek(fpCert, 0, SEEK_SET);
        fread(pbData, sizeof(char), cbData, fpCert);
        fclose(fpCert);

        //Declare and initialize a PFX BLOB containing a PFX packet with the exported and
encrypted certificates and keys.
        CRYPT_DATA_BLOB blob;
        memset(&blob, 0, sizeof(blob));
        blob.pbData = pbData;
        blob.cbData = cbData;
```

```
        //Import a PFX BLOB and return the handle of a store containing certificates and any
associated private keys.
        HCERTSTORE hCertStore = NULL;
        hCertStore = PFXImportCertStore(&blob, L"123", CRYPT_EXPORTABLE);
        //Find certificates and link to certificates in stores that meets some criteria
        PCCERT_CONTEXT pSignedCertContext = CertFindCertificateInStore(hCertStore,
PSL_ENCODING_TYPE, 0, CERT_FIND_ANY, NULL, NULL);
        if (!pSignedCertContext) {
                CertFreeCertificateContext(pSignedCertContext);
                CertCloseStore(hCertStore, CERT_CLOSE_STORE_FORCE_FLAG);
                pContext->Release();
                return FSCRT_ERRCODE_ERROR;
        }
        //Declare and initialize variables for function: SignedData.
        FSCRT_DigestData* pData = NULL;
        pContext->GetData(sig, pData);
        DWORD cbTxts[] = {pData->m_pByteRangeArray[1], pData->m_pByteRangeArray[3]};
        FS_BYTE* pbFileBuffer1 = (FS_BYTE*)malloc(pData->m_pByteRangeArray[1]);
        FS_BYTE* pbFileBuffer2 = (FS_BYTE*)malloc(pData->m_pByteRangeArray[3]);
        fseek(pFile, pData->m_pByteRangeArray[0], SEEK_SET);
        fread(pbFileBuffer1, sizeof(char), pData->m_pByteRangeArray[1],pFile);
        fseek(pFile, pData->m_pByteRangeArray[2], SEEK_SET);
        fread(pbFileBuffer2, sizeof(char), pData->m_pByteRangeArray[3],pFile);

        //Sign the document.
        const unsigned char* pbTxts[] = {pbFileBuffer1, pbFileBuffer2};
        FS_BYTE* pbCrypted = NULL;
        DWORD cbCrypted = 0;
        FS_BOOL bSigned = SignedData(pbTxts, cbTxts, sizeof(cbTxts)/sizeof(FS_DWORD),
pSignedCertContext, pbCrypted, cbCrypted);

        //Release all resources allocated.
        CertFreeCertificateContext(pSignedCertContext);
        CertCloseStore(hCertStore, CERT_CLOSE_STORE_FORCE_FLAG);
        free(pbFileBuffer1);
        free(pbFileBuffer2);
        delete[] pbData;
        pContext->Release();

        //Return when sign fails.
        if (!bSigned)
                return FSCRT_ERRCODE_ERROR;

        //Copy signed data to parameter signedData.
        FSCRT_BStr_SetLength(signedData, cbCrypted);
        memcpy(signedData->str, pbCrypted, cbCrypted);
        signedData->len = cbCrypted;
        free(pbCrypted);

        return FSCRT_ERRCODE_SUCCESS;
}
```

4.25.3 How to implement signature callback function of signing on Linux

```cpp
//Need openssl support.
FS_RESULT _GlobalSign(FS_LPVOID clientData, FS_LPVOID context, FSPDF_SIGNATURE sig, const
FSCRT_BSTR* digest, FSCRT_BSTR* signedData)
{
        FILE* pFile = (FILE*)clientData;
        if (!pFile) return FSCRT_ERRCODE_ERROR;

        CFSCRT_DigestContext* pContext = CFSCRT_DigestContext::GetDigestContext();
        FSCRT_DigestData* pData = NULL;
        pContext->GetData(sig, pData);

        FS_DWORD cbTxts[] = { pData->m_pByteRangeArray[1], pData->m_pByteRangeArray[3]};
        int nFileLength = pData->m_pByteRangeArray[1]+pData->m_pByteRangeArray[3];
        char* pbFileBuffer = (char*)malloc(nFileLength);

        int ret = fseek(pFile, pData->m_pByteRangeArray[0], SEEK_SET);
        fread(pbFileBuffer, sizeof(char), pData->m_pByteRangeArray[1],pFile);
        ret = fseek(pFile, pData->m_pByteRangeArray[2], SEEK_SET);
        fread(pbFileBuffer+pData->m_pByteRangeArray[1], sizeof(char), pData-
>m_pByteRangeArray[3],pFile);

        unsigned char* pbCrypted = NULL;
        int cbCrypted = 0;
        string sFoxitCertFile = FSDK_GetInputFilesFolder("pdfdigitalsignature");
        void* dir = FSDK_OpenFolder(sFoxitCertFile.c_str());
        if(!dir)
        {
                sFoxitCertFile = FSDK_GetFixFolder();
        }
        else
                FSDK_CloseFolder(dir);

        sFoxitCertFile += "/";
        sFoxitCertFile += "foxit_all.pfx";
        pbCrypted = PKCS7_GetEncode(sFoxitCertFile.c_str(), "123", pbFileBuffer, cbCrypted,
nFileLength);
        free(pbFileBuffer);
        if (!pbCrypted)
        {
                pContext->Release();
                return FSCRT_ERRCODE_ERROR;
        }

        FSCRT_BStr_SetLength(signedData, cbCrypted);
        memcpy(signedData->str, pbCrypted, cbCrypted);
        signedData->len = cbCrypted;
        free(pbCrypted);

        pContext->Release();
        return FSCRT_ERRCODE_SUCCESS;
}
```

5 FAQ

5.1 Technical FAQ

1. How can I make the renderer faster?

There are three ways to help you make the renderer faster:

First, render only part of the PDF page;

Second, render in a lower quality. Reduce the width and height and scale it to fit the size needed;

Last, recreate the PDF. Some PDF page consumes a lot of time because of its complexity.

2. How to render only part of a PDF page?

To render only part of a PDF page, the key point is to ensure that the rendering page size is larger than the device's display area. Contents that are not on the display area will not be rendered.

Foxit PDF SDK provides the interface **FSPDF_Page_GetMatrix** to get page's transformation matrix which can be used for rendering page to device's context. Users can call **FSPDF_RenderContext_SetMatrix** to set the matrix to a given rendering context**.**

Assuming that you have downloaded the trial package of Foxit PDF SDK, we can use the simple samples as an example. To render only top-left 1/4 part of a PDF page, you can do as follows:

1) Go to "samples\simple_sample\comm_src\fgsdk_common.cpp" and locate the call of **FSPDF_Page_GetMatrix**;

2) Change the parameters of this interface to

FSPDF_Page_GetMatrix (page, 0, 0, (FS_INT32) width*2, (FS_INT32) height*2, 0, &mt)";

3) Build and run demo "pdf2img".

Then check the generated files under "samples\simple_sample\output_files\pdf2img", you will find that only top-left 1/4 part of the PDF page has been rendered.

Following is the diagrams to further explain the processes above. Only the orange area in PDF page will be rendered to device as shown in Figure 1(a). Figure 1(b) shows the display area in device which has the same size with the page. Figure 2 shows that only the orange area has been rendered in device by quadrupling the rendering page size.

(a) The page in page coordinate (the page size is 270×300, the orange area is only to be rendered.)

(b) The display area in device coordinate (the device size is also 270×300)

Figure 1

Figure 2 Render only top-left 1/4 part of the PDF page to the display area of the device.
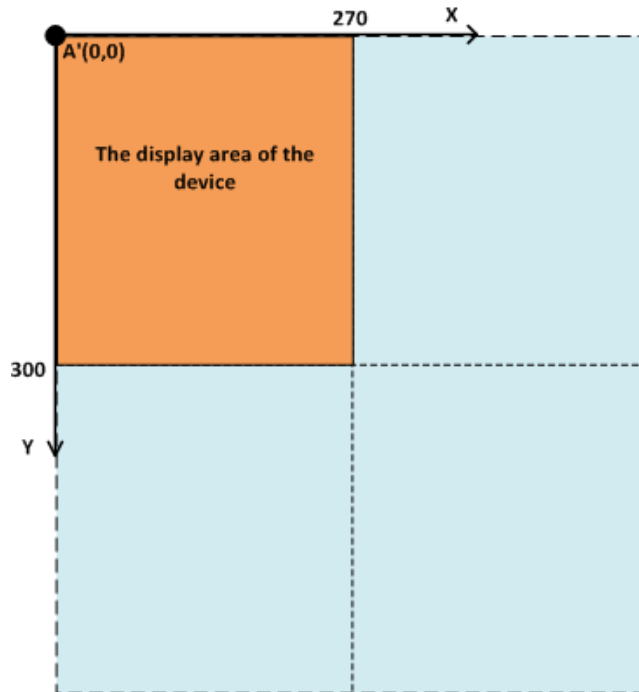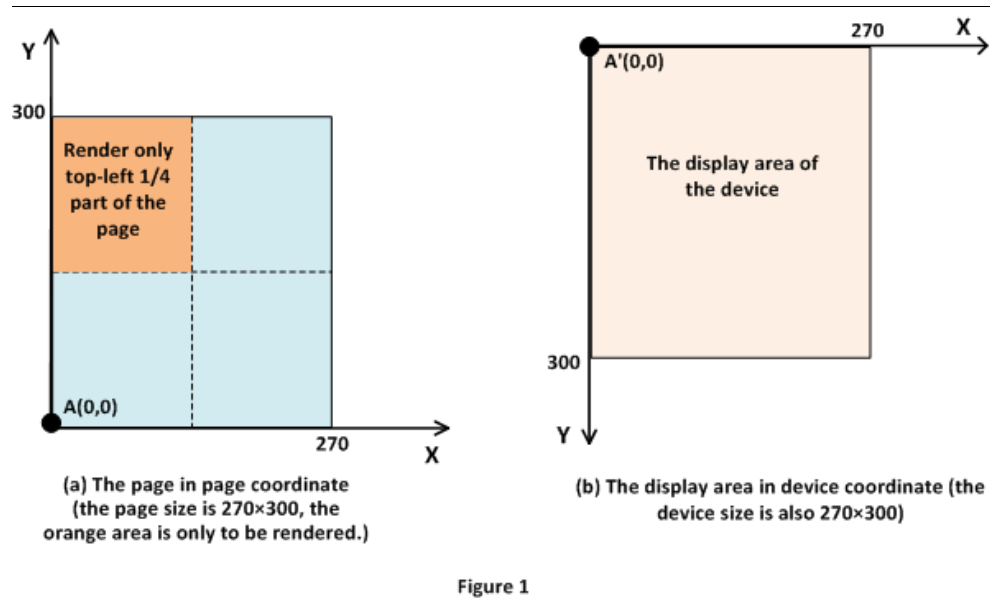
Note:

FSPDF_Page_GetMatrix (FSCRT_PAGE page, FS_INT32 x, FS_INT32 y, FS_INT32 width, FS_INT32 height, FS_INT32 rotation, FSCRT_MATRIX *matrix)

Parameters **x** and **y** specify the location of the top-left point of a PDF page in device coordinate. For example, setting (x, y) to (0, 0) means that the top-left point of the page just matches the origin in device coordinate system; setting (x, y) to (-200, -100) means that the top-left point of the page is in negative direction of x-axis by 200 and in negative direction of y-axis by 100 points of the origin in device coordinate system.

Parameters **width** and **height** specify the rendering page size which may larger or smaller than the actual size of a PDF page. For example, setting them to **width*2** and **height*2** means that the size of the rendering page is four times as large as the actual one, but the actual page size has not been changed.

3. **How can I determine the width and height of a text object before inserting it to a page, which is in order to calculate the starting position more conveniently?**

    Foxit PDF SDK provides the interface **FSCRT_Font_GetCharBBox (FSCRT_Font font, FS_DWORD Unicode, FSCRT_RECT * bbox)** to get the width and height of a text object**.**

4. **After creating a new FreeText annotation, I called FSPDF_Annot_SetContents to set its contents, and then called FSPDF_Annot_ResetAppearance. But why are the contents not displayed?**

    For a FreeText annotation, it's necessary to set its default appearance (DA). Calling the interface **FSPDF_Annot_SetDefaultAppearance** is needed for this task. Usually, we suggest setting up at least **FSPDF_DEFAULTAPPEARANCE::font** and **FSPDF_DEFAULTAPPEARANCE::fontSize**, while setting the value of **FSPDF_DEFAULTAPPEARANCE::flags** to **FSPDF_DEFAULTAPPEARANCE_FONT**.

5. **How can I add an object to FormXObject?**

    FormXObject is one kind of **PageObject**. So we can call the interface **FSPDF_PageObjects_InsertObject** to add an object to **FormXObject.**

6. **How can I print layers which are supposed only to be printed but not displayed?**

    Foxit PDF SDK provides parameter setting to control printing. We can set the parameter "**flag**" in the interface **FSPDF_RenderContext_SetFlags(FSPDF_RENDERCONTEXT pdfRenderContext, FS_DWORD flags)** to "**FSPDF_RENDERCONTEXTFLAG_OCGPRINT**" to print the layers as intended.

7. **What is the relationship between FSPDF_FORM and FSPDF_FORMCONTROL? What about FSPDF_FORMCONTROL and FSCRT_ANNOT?**

    One PDF Form (**FSPDF_FORM**) can contain more than one Form Field, and one Form Field can contain more than one Form Control (**FSPDF_ FORMCONTROL**).

    Form Control (**FSPDF_ FORMCONTROL**) and Annotation (**FSCRT_ANNOT**) belong to the type of "annotation". The difference is that **FSPDF_ FORMCONTROL** denotes the "widget" annotations, and **FSCRT_ANNOT** denotes other annotations which are not "widget".

8. **What is the role of the interface FSPDF_Doc_InitiateJavaScript? And why would it destroy the form actions when calling the interface FSPDF_Doc_InitiateJavaScript before FSPDF_Doc_SetActionHandler?**

    FSPDF_Doc_InitiateJavaScript is mainly used for initiating javascript, such as a series of javascript functions and global variables that are stated by users. When opening a document, we should call this interface to initiate javascript, which may generate other actions that need support from external application. So the interface **FSPDF_Doc_InitiateJavaScript** should be called after **FSPDF_Doc_SetActionHandler.** Otherwise, the form actions will be destroyed.

9. **How can I display fonts correctly?**

    Foxit PDF SDK provides the interface of **FSCRT_Library_AddFontFile** or **FSCRT_Library_SetFontMapperHandler** to display fonts. Following is a sample to show how to use **FSCRT_Library_SetFontMapperHandler** in C++.

```cpp
static FSCRT_FONTMAPPERHANDLER ExternalFontMapper;
static string FallbackFontPath;

// MapExternalFont function will be triggered when an external font is needed.
static FS_RESULT MapExternalFont(FS_LPVOID clientData, FS_LPCSTR fontName, FS_DWORD
fontStyles, FS_INT32 weight, FS_INT32 charset, FSCRT_FILE *fontFile, FS_INT32 *faceIndex)
{
/* The values of fontName, fontStyles, weight, charset will be provided. The setting of
fontFile is depending on programmer. An simple example is to set it to randomFontName*/

    if(fontName=="randomFontName") {//set font file randomFontName.ttf}.

/*Set the font file. Call FSCRT_File_Create with FSCRT_File_Write,
FSCRT_File_CreateCacheFile, FSCRT_File_CreateFromMemory, or FSCRT_File_CreateFromFileName
to create the handle for FSCRT_FILE. Please see fs_base_r.h or fs_codec_r.h for details.
//faceIndex can be set to 0 for default*/

Return 0;
}
//Apply the font mapper after unlocking the library.
 ExternalFontMapper.MapFont = MapExternalFont;

 ExternalFontMapper.clientData = 0; //clientData is your own data that you want to pass to
MapExternalFont when it is triggered

 FSCRT_Library_SetFontMapperHandler (&ExternalFontMapper);
```

10. **How can I rotate a text object around its center instead of the origin of coordinates?**

    Figure 1(a) shows the rotation of a text object around the origin of coordinates in Foxit PDF SDK. If we want to rotate a text object around its center as shown in Figure 1(b), a set of complex transformations are needed. The complete transformation process is shown in Figure 2.

First, translate the center of the text object to the origin of the coordinates. The translation vector is (-Cx,-Cy), as shown in Figure 2(b). Second, rotate the text object around the origin of coordinates with angle θ, as shown in Figure 2(c). Finally, translate the text object by using vector (Cx,Cy). The red text is the result as shown in Figure 2(d).
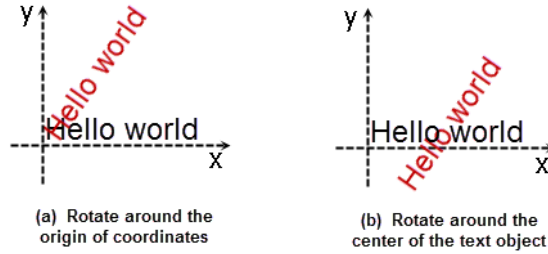
Figure 1: Rotation of a text object
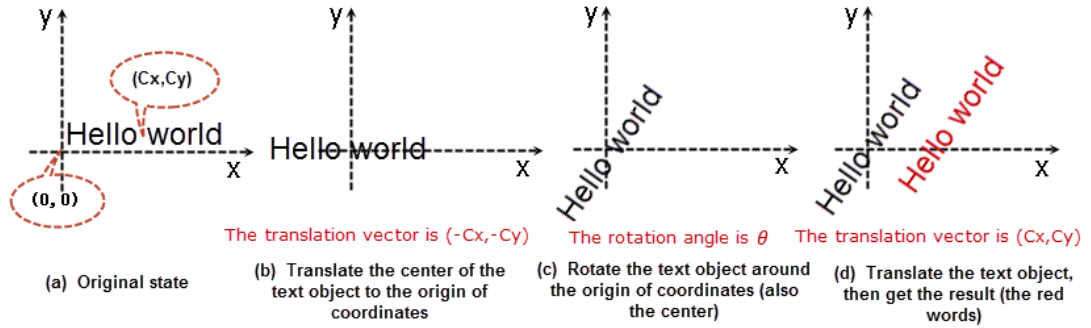
Figure 2: Complete transformation process of rotating a text object around its center

The complex transformation matrix is made up of two translation matrices and a rotation matrix. The homogeneous coordinates in PDF Reference are used for coordinate representation. As a result, the complex transformation matrix is calculated by the following formula.

$M = T_1 \cdot R \cdot T_2$, where:

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -C_x & -C_y & 1 \end{bmatrix} \quad R = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ C_x & C_y & 1 \end{bmatrix}$$

$$M = T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -C_x & -C_y & 1 \end{bmatrix}\begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ C_x & C_y & 1 \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ S_x & S_y & 1 \end{bmatrix}$$

$$S_x = C_x - (C_x \cdot cos\theta - C_y \cdot sin\theta)$$

$$S_y = C_y - (C_x \cdot sin\theta + C_y \cdot cos\theta)$$

Once the complex transformation matrix is obtained, we can call the following functions to transform the text object.

FSCRT_MATRIX textmatrix = { $cos\theta, sin\theta, -sin\theta, cos\theta,\ S_x\ ,S_y$ **};**
ret = FSPDF_PageObject_SetMatrix(page, atuoTextObj.GetObject(), &textmatrix);

11. **Why can't I get the desired filling color if the bitmap format is not ARGB or RGB or if the bitmap contains alpha channel when using the interface FSCRT_Bitmap_FillRect?**

Before SDK 4.4 version, there exist two bugs when using the interface FSCRT_Bitmap_FillRect to fill a bitmap object with a specified color.

1) **FSCRT_Bitmap_FillRect** just only receives the color format of **ARGB**, but it does not judge the format of the bitmap and treats all bitmap formats as ARGB. So, users cannot get the desired filling color if the bitmap format is not ARGB or RGB.
   For example, assume that the bitmap format is BGR. Interface **FSCRT_Bitmap_FillRect** will also treat it as RGB, and then the component value of R and B will be exchanged, which leads to unexpected filling color.

2) If you use 0xffxxxxxx to fill a bitmap that contains alpha channel, some rendering contents will be lost.

These two bugs have been fixed in PDF SDK 4.4 version.

12. **Why the length of the string retrieved from interface FSPDF_TextPage_GetChars is different from the count of characters getting from interface FSPDF_TextPage_CountChars?**

The string retrieved from interface **FSPDF_TextPage_GetChars** is a UTF-8 string. The interface **FSPDF_TextPage_CountChars** gets the count of characters in a page.

"The length of string" and "the count of characters" are two totally different concepts. The length of a UTF-8 string represents how many bytes the UTF-8 string consumes, rather than how many characters the UTF-8 string contains. When a character just takes up one byte, the length of a UTF-8 string will be same with the count of characters. But for some characters, they take up more than one byte, such as Chinese characters and some special characters. In this case, the length of a UTF-8 string will usually be larger than the count of characters. So, the length of the string retrieved from interface **FSPDF_TextPage_GetChars** will not always be just same with the count of characters getting from interface **FSPDF_TextPage_CountChars.**

On the other hand, a PDF document may have some invisible characters which can be counted by interface **FSPDF_TextPage_CountChars,** but cannot be retrieved by interface **FSPDF_TextPage_GetChars.** In this case, the length of the string retrieved from interface

FSPDF_TextPage_GetChars will also be different from the count of characters getting from interface **FSPDF_TextPage_CountChars.**

5.2 Sales & Marketing FAQ

1. **What is 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 it 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 question or comment 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 API Reference.chm

[3] Foxit PDF SDK Demo guide
     sdk_folder/docs/FoxitPDFSDK_DemoTutorial.doc

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