Quick Start Guide

Foxit PDF SDK

Microsoft® Partner
Gold Independent Software Vendor (ISV)

TABLE OF CONTENTS

1 INTRODUCTION TO FOXIT PDF 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.

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

1.1 Why Foxit PDF SDK is your choice

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

- **Easy to integrate**

Developers can seamlessly integrate Foxit PDF SDK into their own applications.

- **Lightweight footprint**

Do not exhaust system resource and deploys quickly.

- **Cross-platform support**

Support current mainstream platforms, such as Windows, Mac, Linux, Web, Android, iOS, and UWP.

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

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

- **Premium World-side Support**

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

1.2 Foxit PDF SDK for C++ API

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

In this guide, we focus on the introduction of Foxit PDF SDK for C++ API on Windows, Linux and Mac platforms.

Foxit PDF SDK for C++ API ships with simple-to-use APIs that can help C++ developers seamlessly integrate powerful PDF technology into their own projects on Windows, Linux and Mac platforms. It provides rich features on PDF documents, such as PDF viewing, bookmark navigating, text selecting/copying/searching, PDF signatures, PDF forms, rights management, PDF annotations, and full text search.

1.3 Evaluation

Foxit PDF SDK allows users to download trial version to evaluate SDK. The trial version has no difference from a standard version except for the 30-day limitation trial period and the trail watermarks that will be generated on the PDF pages. After the evaluation period expires, customers should contact Foxit sales team and purchase licenses to continue using Foxit PDF SDK.

1.4 License

Developers should purchase licenses to use Foxit PDF SDK in their solutions. Licenses grant users permissions to release their applications based on PDF SDK libraries. However, users are prohibited to distribute any documents, sample codes, or source codes in the SDK released package to any third party without the permission from Foxit Software Incorporated.

1.5 About this guide

This guide is intended for the developers who need to integrate Foxit PDF SDK with C++ program language into their own applications. It aims at introducing the installation package, and the usage of SDK.

2 GETTING STARTED

It's very easy to setup Foxit PDF SDK and see it in action! This guide will provide you with a brief introduction about our SDK package. As a cross-platform product, Foxit PDF SDK supports the identical interfaces for desktop system of Windows, Linux, and Mac. The following sections introduce the contents of system requirements, the installation package as well as how to run a demo, and create your own project.

2.1 System Requirements

Platform	System Requirement	Memo
Windows	Windows XP, Vista, 7, 8 and 10 (32-bit and 64-bit) Windows Server 2003, 2008 and 2012 (32-bit and 64-bit)	It only supports for Windows 8/10 classic style, but not for Store App or Universal App.
Linux	32-bit and 64-bit OS	All Linux samples have been tested on Ubuntu14.0 32/64 bit.
Mac	Mac OS X 10.6 to 10.12	

2.2 Windows

In this guide, one thing to note is that the highlighted rectangle in the figure is the version of the SDK. Here the SDK version is 6.3, so it shows 6_3.

2.2.1 What is in the package

Download Foxit PDF SDK zip for Windows package and extract it to a new directory "foxitpdfsdk_6_3_win", which is shown in Figure 2-1. The release package contains the following folders:

docs: API references, quick start guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos

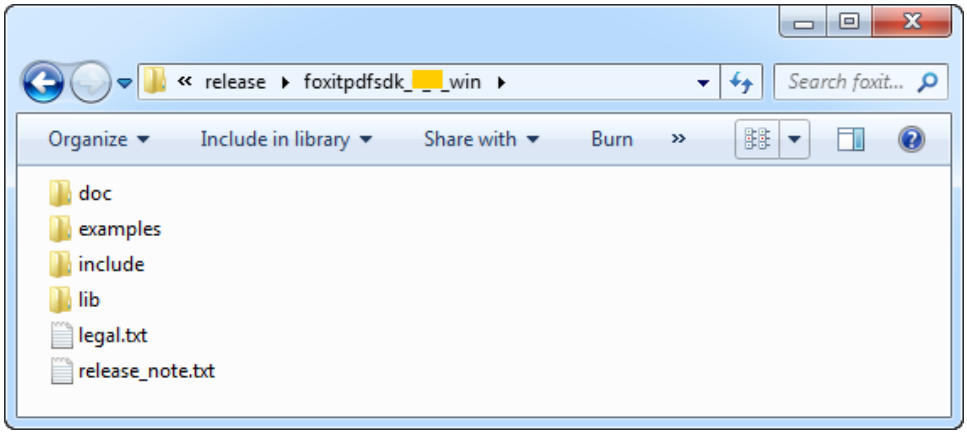

Figure 2-1

In the "examples" folder, there are two types of demos. "examples\simple_demo" contains more than 20 demos that cover a wide range of PDF applications. "examples\view_demo" contains a UI demo that realizes a lite PDF viewer.

2.2.2 How to run a demo

Simple Demo

Simple demo projects provide examples to show developers about how to effectively apply PDF SDK APIs to complete their applications.

To run a demo in Visual Studio (expect connectedpdf, security and signature demos which will be introduced later), you can follow the steps below:

1) Load the visual studio solution files "simple_demo_vs2010.sln" or "simple_demo_vs2015.sln" or "simple_demo_vs2017.sln" (depending on your Visual Studio version) in the "examples\simple_demo" folder.

2) Build all the demos by clicking "Build > Build Solution". Alternatively, if you merely want to build a specific demo, you can right-click it and then choose "Build" or load the "*.vxcproj" file in the folder of a specific demo project and then build it.

   After building, the executable file ".exe" will be generated in the "examples\simple_demo\bin" folder (See Figure 2-2). And the names of the executable files depend on the build configurations.

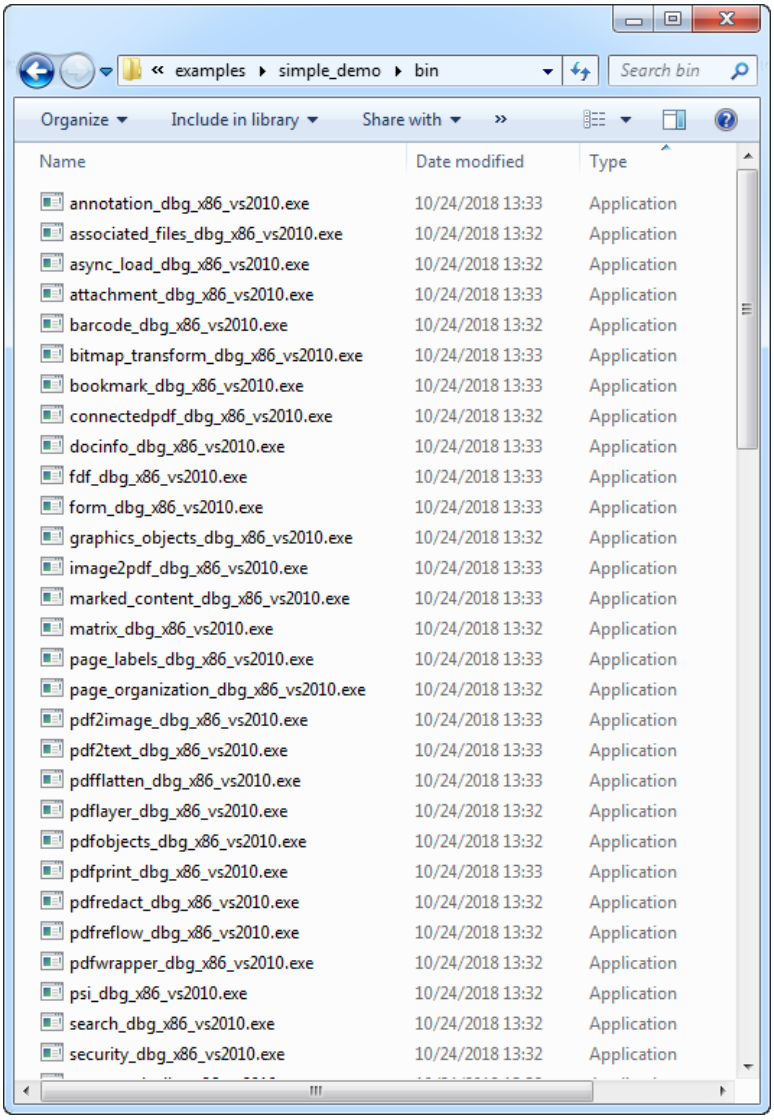

Figure 2-2

3) Run a specific executable file, just double-click it.

   Some demos will generate output files (pdf, text or image files) to a folder named by the project name under "example\simple_demo\output_files\" folder.

   ***Note**: If you want to see the detailed executing processes, you can run it in command line. Start "cmd.exe", navigate to "examples\simple_demo\bin", and run a specific executable file.*

ConnectedPDF demo

Before running **connectedpdf** demo, please make sure to configure the following settings in "connectedpdf.m" under "examples\simple_demo\connectedpdf" folder. Then run the demo according to the other demos.

```
String endpoint = "";        // Please write server address here.
String user_email = "";      // Please write user name here.
String password = "";        // Please write password here.
```

Security demo

Before running **security** demo, you should install the certificates "foxit.cer" and "foxit_all.pfx" found in "examples\simple_demo\input_files" folder.

a) To install "foxit.cer", double-click it to start the certificate import wizard. Then select "Install certificate… > Next > Next > Finish".

b) To install "foxit_all.pfx", double-click it to start the certificate import wizard. Then select "Next > Next > (Type the password for the private key in the textbox) and click Next > Next > Finish".

c) Run the demo following the steps as the other demos.

Signature demo

Before running **signature** demo, you should ensure that the OpenSSL has been already installed in your machine. Download an OpenSSL source package from the OpenSSL website, or you can contact us directly, and then extract it and do the following:

1) Put the OpenSSL folder into the "include" folder which ensures that the OpenSSL header files included in the demos can be found.

2) Put the "libeay32.lib" library into the "libs" folder.

3) Run the demo following the steps as the other demos.

***Note**: We have verified that OpenSSL 1.0.2m version is available on the signature demo, you can replace it with the desired version, and maybe need to do some changes.*

View Demo

This view demo provides an example for developers to realize a PDF reader using PDF SDK APIs.

To run the demo in Visual Studio, load "PDFReader_VS2010.sln" or "PDFReader_VS2015.sln" or "PDFReader_VS2017.sln " (depending on your Visual Studio version) in the "examples\view_demo\PDFReader\project" folder, and then click "Debug > Start Without Debugging" to run it. After the demo starts, you will see the following window:

Figure 2-3

You can drag a PDF file to the above window (Figure 2-3) to open it and browse the content by scrolling down or moving the PDF page by holding the left mouse button. Besides, you can click "Annot > HighLight" to highlight the selected text. A screenshot of the demo is shown in Figure 2-4.

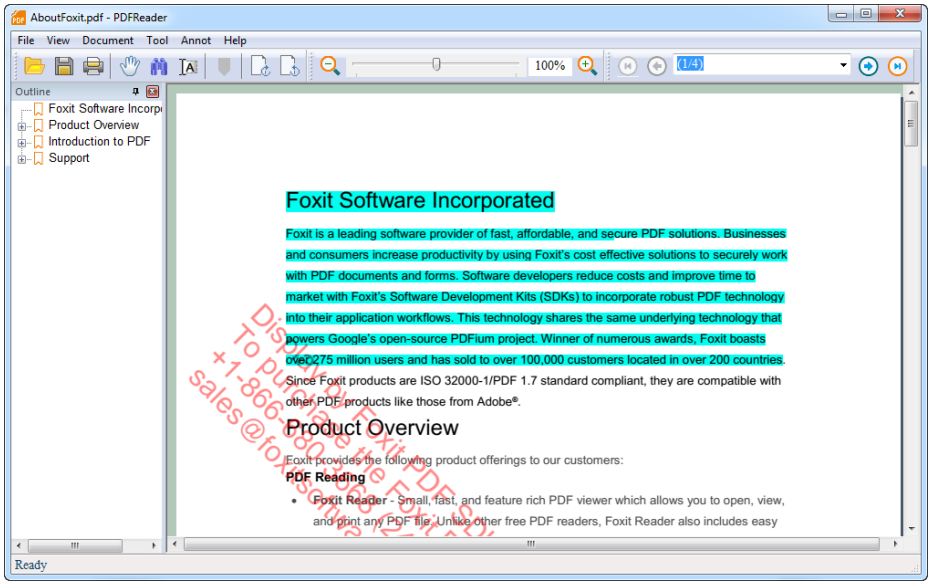

Figure 2-4

2.2.3 How to create a simple project

In this section, we will show you how to use Foxit PDF SDK for Windows to create a simple project that renders the first page of a PDF to a bitmap and saves it as a JPG image. Please follow the steps below:

1) Open Visual Studio and create a new Win32 Console Application named "test_win".

2) Copy the "include" and "lib" folders from the "foxitpdfsdk_6_3_win" folder to the project "test_win" folder.

3) Add the "include" folder to your "Additional Include Directories". Right-click the *test_win* project in Solution Explorer, choose "Properties", and find "Configuration Properties > C/C++ > General > Additional Include Directories".

4) Add include header statements to the beginning of test_win.cpp.

```
#include <iostream>
#include "../include/common/fs_common.h"
#include "../include/pdf/fs_pdfdoc.h"
#include "../include/pdf/fs_pdfpage.h"
#include "../include/common/fs_render.h"
```

5) Add using namespace statements.

```
using namespace std;
using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;
```

6) Include Foxit PDF SDK library.

```
#if defined (_WIN64) // windows 64bit platforms.
	#if defined (_DEBUG)
		#pragma comment (lib, "../lib/fsdk_win64.lib")
	#else
		#pragma comment (lib, "../lib/fsdk_win64.lib")
	#endif

#elif defined (_WIN32) // windows 32bit platforms.
	#if defined (_DEBUG)
		#pragma comment (lib, "../lib/fsdk_win32.lib")
	#else
		#pragma comment (lib, "../lib/fsdk_win32.lib")
	#endif
#endif
```

7) Initialize the Foxit PDF SDK library. It is necessary for apps to initialize Foxit PDF SDK using a license before calling any APIs. The trial license files can be found in the "libs" folder.

```
const char* sn = " ";
const char* key = " ";
foxit::ErrorCode code = Library::Initialize(sn, key);
if (code != foxit::e_ErrSuccess) {
	return FALSE;
}
```

Note *The value of "sn" can be got from "**gsdk_sn.txt**" (the string after "SN=") and the value of "key" can be got from "**gsdk_key.txt**" (the string after "Sign=").*

8) Load a PDF document, and parse the first page of the document. Let us assume that you have already put a "Sample.pdf" to the "test_win\test_win" folder.

```
PDFDoc doc("Sample.pdf");
ErrorCode error_code = doc.Load();
if (error_code!= foxit::e_ErrSuccess) return 0;
PDFPage page = doc.GetPage(0);
page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);
```

9) Render a Page to a bitmap and save it as a JPG file.

```
int width = static_cast<int>(page.GetWidth());
int height = static_cast<int>(page.GetHeight());
Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

// Prepare a bitmap for rendering.
Bitmap bitmap(width, height, Bitmap::e_DIBArgb, NULL, 0);
bitmap.FillRect(0xFFFFFFFF, NULL);
// Render page.
Renderer render(bitmap, false);
render.StartRender(page, matrix, NULL);

// Add the bitmap to image and save the image.
Image img;
img.AddFrame(bitmap);
img.SaveAs("testpage.jpg");
```

10) Click "Build > Build Solution" to build the project. The executable file "test_win.exe" will be generated in "test_win\Debug" or "test_win\Release" folder depending on the build configurations.

11) Copy "fsdk_win32.dll" or "fsdk_win64.dll" in the "libs" folder to the output directory ("test_win\Debug" or "test_win\Release"). Please make sure that the "fsdk_win**.dll" architecture needs to match the platform target (Win32 or Win64) of the application.

12) Run the project. Choose one of the following:

i. Click "Debug > Start Without Debugging" in Visual Studio to run the project, and the "testpage.jpg" will be generated in the "test_win\test_win" folder (same with "test_win.cpp").

ii. Double-click the executable file "test_win.exe" to run the project. In this way, you should put the "Sample.pdf" to the same folder with the "test_win.exe", and the "testpage.jpg" will also be generated in the same folder.

The final contents of "test_win.cpp" is as follow:

```
#include "stdafx.h"

#include <iostream>
#include "../include/common/fs_common.h"
#include "../include/pdf/fs_pdfdoc.h"
#include "../include/pdf/fs_pdfpage.h"
#include "../include/common/fs_render.h"

using namespace std;
using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;

// Include Foxit PDF SDK library.
#if defined (_WIN64) // windows 64bit platforms.
	#if defined (_DEBUG)
		#pragma comment (lib, "../lib/fsdk_win64.lib")
	#else
		#pragma comment (lib, "../lib/fsdk_win64.lib")
	#endif

#elif defined (_WIN32) // windows 32bit platforms.
	#if defined (_DEBUG)
		#pragma comment (lib, "../lib/fsdk_win32.lib")
	#else
		#pragma comment (lib, "../lib/fsdk_win32.lib")
	#endif
#endif

int _tmain(int argc, _TCHAR* argv[])

{
	// The value of "sn" can be got from "gsdk_sn.txt" (the string after "SN=").
```

```
	// The value of "key" can be got from "gsdk_key.txt" (the string after "Sign=").
	const char* sn = " ";
	const char* key = " ";
	foxit::ErrorCode code = Library::Initialize(sn, key);
	if (code != foxit::e_ErrSuccess) {
		return FALSE;
	}

	// Load a PDF document, and parse the first page of the document.
	PDFDoc doc("Sample.pdf");
	ErrorCode error_code = doc.Load();
	if (error_code!= foxit::e_ErrSuccess) return 0;
	PDFPage page = doc.GetPage(0);
	page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);

	int width = static_cast<int>(page.GetWidth());
	int height = static_cast<int>(page.GetHeight());
	Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

	// Prepare a bitmap for rendering.
	Bitmap bitmap(width, height, Bitmap::e_DIBArgb, NULL, 0);
	bitmap.FillRect(0xFFFFFFFF, NULL);
	// Render page.
	Renderer render(bitmap, false);
	render.StartRender(page, matrix, NULL);

	// Add the bitmap to image and save the image.
	Image img;
	img.AddFrame(bitmap);
	img.SaveAs("testpage.jpg");
	return 0;
}
```

2.3 Linux

In this guide, one thing to note is that the highlighted rectangle in the figure is the version of the SDK. Here the SDK version is 6.3, so it shows 6_3.

2.3.1 What is in this package

Download Foxit PDF SDK zip for Linux package and extract it to a new directory "foxitpdfsdk_6_3_linux".The structure of the release package is shown in Figure 2-5. This package contains the following folders:

docs: API references, quick start guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos

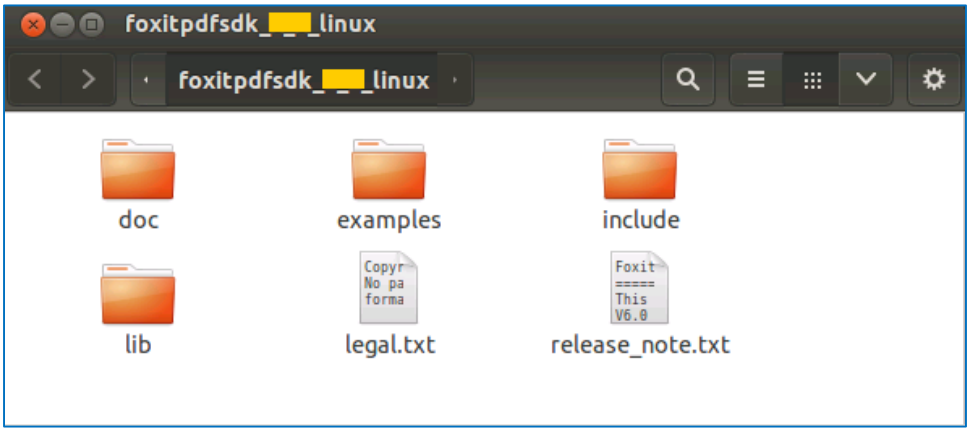

Figure 2-5

In "example\simple_demo" folder, there are a wide range of PDF document application demos.

2.3.2 How to run a demo

Before running the demos, please make sure you have configured the environment correctly and installed CMake (3.1 or later) on your machine.

OS	Compiler tools or IDE	Version
Linux	GCC	4.8 and later

To run a demo in a terminal window (expect connectedpdf, security and signature demos which will be introduced later), please follow the steps:

1) Open a terminal window, navigate to "foxitpdfsdk_6_3_linux\examples\simple_demo";
2) Run "**cmake -DPRJ_NAME=XXX**" to compile a specific demo. "XXX" is the demo name, which must be "annotation", "attachment", "pdf2text" and so on. For example, run "cmake -DPRJ_NAME=annotation".
3) Run "**make**" to build the demo specified in the above command. Then the executable file will be generated named "**XXX_xxx**". "XXX" is the demo name and "xxx" is the architecture name, such as "annotation_linux64".
4) Run "**./XXX_xxx**" to run the demo. For example, run "./annotation_linux64" to run the annotation demo.

Some demos will generate output files (pdf, text or image files) to a folder named by the project name under "example/simple_demo/output_files/".

ConnectedPDF demo

Before running **connectedpdf** demo, please make sure to configure the following settings in "connectedpdf.m" under "examples\simple_demo\connectedpdf" folder. Then run the demo according to the other demos.

```
String endpoint = "";        // Please write server address here.
String user_email = "";      // Please write user name here.
String password = "";        // Please write password here.
```

Security and Signature demos

Before running **security** and **signature** demos, please make sure that you have already installed OpenSSL. Download an OpenSSL source package from the OpenSSL website, or you can contact us directly, and then extract it and do the following:

1) Put the OpenSSL folder into the "include" folder which ensures that the OpenSSL header files included in the demos can be found.

2) Put the "libssl.a" and "libcrypto.a" libraries into the "libs" folder.

3) Run the demos following the steps as the other demos.

***Note**: We have verified that OpenSSL 1.0.2m version is available on the security and signature demos, you can replaced it with the desired version, and maybe need to do some changes.*

2.3.3 Create a simple project

In this section, we will show you how to use Foxit PDF SDK for Linux to create a simple project that renders the first page of a PDF to a bitmap and saves it as a JPG image. Please follow the steps below:

1) Create a folder called "test_linux".

2) Copy "include" and "lib" folders from "foxitpdfsdk_6_3_linux" folder to the project "test_linux" folder.

3) Create a "test_linux.cpp" file under "test_linux" folder, and add the code similar to the "test_win.cpp" described in section 2.2.3 "How to create a simple project".

   The "test_linux.cpp" will look like as follows: (For better viewing, I paste the code to Visual Studio to highlight the code)

```
#include <iostream>
#include "common/fs_common.h"
#include "pdf/fs_pdfdoc.h"
#include "pdf/fs_pdfpage.h"
```

```
#include "common/fs_render.h"

using namespace std;
using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;

int main(int argc, char* argv[])

{
	// The value of "sn" can be got from "gsdk_sn.txt" (the string after "SN=").
	// The value of "key" can be got from "gsdk_key.txt" (the string after "Sign=").
	const char* sn = " ";
	const char* key = " ";
	foxit::ErrorCode code = Library::Initialize(sn, key);
	if (code != foxit::e_ErrSuccess) {
		return FALSE;
	}

	// Load a PDF document, and parse the first page of the document.
	PDFDoc doc("../../Sample.pdf");
	ErrorCode error_code = doc.Load();
	if (error_code!= foxit::e_ErrSuccess) return 0;
	PDFPage page = doc.GetPage(0);
	page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);

	int width = static_cast<int>(page.GetWidth());
	int height = static_cast<int>(page.GetHeight());
	Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

	// Prepare a bitmap for rendering.
	Bitmap bitmap(width, height, Bitmap::e_DIBArgb, NULL, 0);
	bitmap.FillRect(0xFFFFFFFF, NULL);
	// Render page.
	Renderer render(bitmap, false);
	render.StartRender(page, matrix, NULL);

	// Add the bitmap to image and save the image.
	Image img;
	img.AddFrame(bitmap);
	img.SaveAs("testpage.jpg");
	return 0;
}
```

4) Put a "Sample.pdf" document into the project "test_linux" folder.

5) 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 as follows:

```
CXX=g++

# Foxit PDF SDK lib and head files include
INCLUDE_PATH=-Iinclude
FSDKLIB_PATH=-Llib
LIB_PATH=lib
FSDKLIB=-lfsdk_linux64
LIBNAME=libfsdk_linux64.so
LIBS=$(FSDKLIB) -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_linux

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

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

test_linux: dir test_linux.o
	$(CXX) $(OBJ_PATH)/test_linux.o $(DEST) $(FSDKLIB_PATH) $(LIBS) $(LDFLAGS)
```

6) Build the project. Open a terminal window, navigate to "test_linux", and run "**make test_linux**" to generate binary file in "test_linux\bin\rel_gcc" folder.

7) Execute the binary file. Navigate to the folder with the terminal, run "**./test_linux**", and the "testpage.jpg" will be generated in current folder.

2.4 Mac

In this guide, one thing to note is that the highlighted rectangle in the figure is the version of the SDK. Here the SDK version is 6.3, so it shows 6_3.

2.4.1 What is in this package

Download Foxit PDF SDK zip for Mac package and extract it to a new directory "foxitpdfsdk_6_3_mac". The structure of the release package is shown in Figure 2-5. This package contains the following folders:

docs: API references, quick start guide

include: header files for foxit pdf sdk API

lib: libraries and license files

samples: sample projects and demos

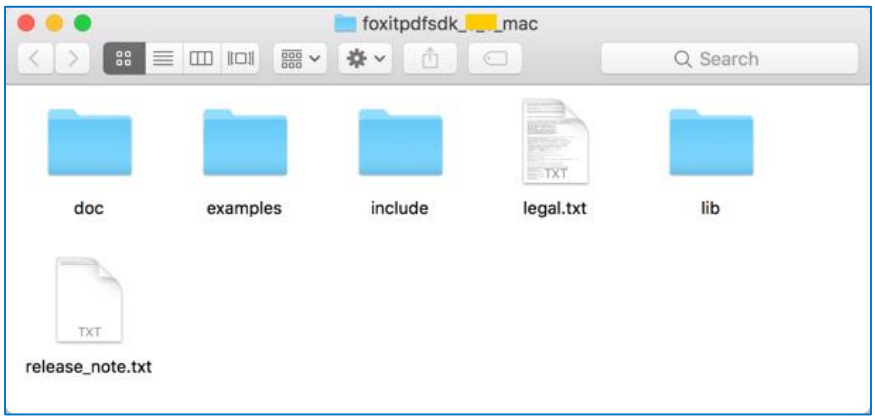

Figure 2-6

2.4.2 How to run a demo

Before running the demos, please make sure you have configured the environment correctly and installed CMake (3.1 or later) on your machine.

OS	Compiler tools or IDE	Version
Mac	Xcode	8 and later

To run a demo in a terminal window (expect connectedpdf, security and signature demos which will be introduced later), please follow the steps:

1) Open a terminal window, navigate to "foxitpdfsdk_6_3_mac\examples\simple_demo";

2) Run "**cmake -DPRJ_NAME=XXX**" to compile a specific demo. "XXX" is the demo name, which must be "annotation", "attachment", "pdf2text" and so on. For example, run "cmake -DPRJ_NAME=annotation".

3) Run "**make**" to build the demo specified in the above command. Then the executable file will be generated named "**XXX_xxx**". "XXX" is the demo name and "xxx" is the architecture name, such as "annotation_mac64".

4) Run "**./XXX_xxx**" to run the demo. For example, run "./annotation_mac64" to run the annotation demo.

"\examples\simple_demo\input_files" contains all the input files used among these demos. Some demos will generate output files (pdf, text or image files) to a folder named by the demo name under "example/simple_demo/output_files/".

ConnectedPDF demo

Before running **connectedpdf** demo, please make sure to configure the following settings in "connectedpdf.m" under "examples\simple_demo\connectedpdf" folder. Then run the demo according to the other demos.

```
String endpoint = "";        // Please write server address here.
String user_email = "";      // Please write user name here.
String password = "";        // Please write password here.
```

Security and Signature demos

Before running **security** and **signature** demos, please make sure that you have already installed OpenSSL. Download an OpenSSL source package from the OpenSSL website, or you can contact us directly, and then extract it and do the following:

1) Put the OpenSSL folder into the "include" folder which ensures that the OpenSSL header files included in the demos can be found.

2) Put the "libssl.a" and "libcrypto.a" libraries into the "libs" folder.

3) Run the demos following the steps as the other demos.

***Note**: We have verified that OpenSSL 1.0.2m version is available on the security and signature demos, you can replaced it with the desired version, and maybe need to do some changes.*

2.4.3 How to create a simple project

In this section, we will show you how to use Foxit PDF SDK for Mac (C++) to create a simple project that renders the first page of a PDF to a bitmap and saves it as a JPG image. Please follow the steps below:

1) Create a folder called "test_mac".

2) Copy "include" and "lib" folders from "foxitpdfsdk_6_3_mac" folder to the project "test_mac" folder.

3) Create a "test_mac.cpp" file under "test_mac" folder, and add the code similar to the "test_win.cpp" described in section 2.2.3 "How to create a simple project".

The "test_mac.cpp" will look like as follows:

```
#include <iostream>
#include "common/fs_common.h"
#include "pdf/fs_pdfdoc.h"
#include "pdf/fs_pdfpage.h"
#include "common/fs_render.h"

using namespace std;
using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;

int main(int argc, char* argv[])

{
    // The value of "sn" can be got from "gsdk_sn.txt" (the string after "SN=").
    // The value of "key" can be got from "gsdk_key.txt" (the string after "Sign=").
    const char* sn = " ";
    const char* key = " ";
    foxit::ErrorCode code = Library::Initialize(sn, key);
    if (code != foxit::e_ErrSuccess) {
        return FALSE;
    }

    // Load a PDF document, and parse the first page of the document.
    PDFDoc doc("../../Sample.pdf");
    ErrorCode error_code = doc.Load();
    if (error_code!= foxit::e_ErrSuccess) return 0;
    PDFPage page = doc.GetPage(0);
    page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);

    int width = static_cast<int>(page.GetWidth());
    int height = static_cast<int>(page.GetHeight());
    Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

    // Prepare a bitmap for rendering.
    Bitmap bitmap(width, height, Bitmap::e_DIBArgb, NULL, 0);
    bitmap.FillRect(0xFFFFFFFF, NULL);
    // Render page.
    Renderer render(bitmap, false);
    render.StartRender(page, matrix, NULL);

    // Add the bitmap to image and save the image.
    Image img;
    img.AddFrame(bitmap);
```

```
    img.SaveAs("testpage.jpg");
    return 0;
}
```

4) Put a "Sample.pdf" document into the project "test_mac" folder.

5) Create a Makefile. In this Makefile, the PDF SDK library shall be included in the build path. Use libfsdk_mac64.dylib. A sample Makefile is as follows:

```
CXX=g++

# Foxit PDF SDK lib and head files include
INCLUDE_PATH=-Iinclude
LIBNAME=./lib/libfsdk_mac64.dylib
LDFLAGS=-Wl,-rpath,../../lib
DEST_PATH=./bin/rel_gcc
OBJ_PATH=./obj/rel
CCFLAGS=-c

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

all: test_mac

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

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

test_mac: dir test_mac.o
	$(CXX) $(OBJ_PATH)/test_mac.o $(DEST) $(LDFLAGS) $(LIBNAME)
```

6) Build the project. Open a terminal window, navigate to "test_mac", and run "**make test_mac**" to generate binary file in "test_mac\bin\rel_gcc" folder.

7) Execute the binary file. Navigate to the folder with the terminal, run "**./test_mac**", and the "testpage.jpg" will be generated in current folder.

3 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.

3.1 Initialize Library

It is necessary for applications to initialize Foxit PDF SDK before calling any APIs. The function foxit::common::Library::Initialize is provided to initialize Foxit PDF SDK. A license should be purchased for the application and pass unlock key and code to get proper supports. When there is no need to use Foxit PDF SDK any more, please call function foxit::common::Library::Release to release it.

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

Example:

3.1.1 How to initialize Foxit PDF SDK

```
#include "include/common/fs_common.h"

using namespace foxit;
using namespace common;
...

consta char* sn = " ";
const char* key = " ";
foxit::ErrorCode code = Library::Initialize(sn, key);
if (code != foxit::e_ErrSuccess) {
       return FALSE;
}
```

3.2 Document

A PDF document object can be constructed with an existing PDF file from file path, memory buffer, a custom implemented ReaderCallback object and an input file stream. Then call function PDFDoc::Load or PDFDoc::StartLoad to load document content. A PDF document object is used for document level operation, such as opening and closing files, getting page, metadata and etc.

Example:

3.2.1 How to create a PDF document from scratch

```
#include "include/pdf/fs_pdfdoc.h"
```

```
using namespace foxit;
using namespace common;
using namespace pdf;
...

PDFDoc doc();
```

***Note**: It creates a new PDF document without any pages.*

3.2.2 How to load an existing PDF document from file path

```
#include "include/pdf/fs_pdfdoc.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...

PDFDoc doc("Sample.pdf");
ErrorCode error_code = doc.Load();
if (error_code!= foxit::e_ErrSuccess) return 0;
```

3.2.3 How to load an existing PDF document from a memory buffer

```
#include "include/pdf/fs_pdfdoc.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...

FILE* pFile = fopen(TEST_DOC_PATH"blank.pdf", "rb");
ASSERT_EQ(TRUE, NULL != pFile);
fseek(pFile, 0, SEEK_END);
long lFileSize = ftell(pFile);
char* buffer = new char[lFileSize];
memset(buffer, 0, sizeof(char)*lFileSize);
fseek(pFile, 0, SEEK_SET);
fread(buffer, sizeof(char), lFileSize, pFile);
fclose(pFile);
PDFDoc doc = PDFDoc(buffer, lFileSize);
ErrorCode error_code = doc.Load();
if (error_code!= foxit::e_ErrSuccess) return 0;
```

3.2.4 How to load an existing PDF document from a file read callback object

```
#include "include/pdf/fs_pdfdoc.h"

using namespace foxit;
using namespace common;
using namespace pdf;
```

```
...

class CFSFile_Read : public ReaderCallback
{
public:
	CFSFile_Read():m_fileFP(NULL)
		,m_bLargeFile(FALSE)
	{}

	~CFSFile_Read() {}

	bool LoadFile(const wchar_t* wFilePath, bool bLargeFile = FALSE)
	{
		std::wstring strTemp(wFilePath);
		string bstrFilepath = wchar2utf8(strTemp.c_str(), strTemp.size());

		m_fileFP = fopen(bstrFilepath.c_str(), "rb");
		if (!m_fileFP) return FALSE;

		m_bLargeFile = bLargeFile;
		return TRUE;
	}

	bool LoadFile(const char* filePath, bool bLargeFile = FALSE)
	{
		m_fileFP = fopen(filePath, "rb");
		if (!m_fileFP) return FALSE;

		m_bLargeFile = bLargeFile;
		return TRUE;
	}

	FILESIZE	GetSize()
	{
		if (m_bLargeFile)
		{
#if defined(_WIN32) || defined(_WIN64)
			_fseeki64(m_fileFP, 0, SEEK_END);
			long long sizeL = _ftelli64(m_fileFP);
#elif defined(__linux__) || defined(__APPLE__)
			fseeko(m_fileFP, 0, SEEK_END);
			long long sizeL = ftello(m_fileFP);
#endif
			return sizeL;
		}
		else
		{
			fseek(m_fileFP, 0, SEEK_END);
			return (uint32)ftell(m_fileFP);
		}
	}

	int ReadBlock(void* buffer, FILESIZE offset, size_t size)
	{
		if (m_bLargeFile)
		{
#if defined(_WIN32) || defined(_WIN64)
			_fseeki64(m_fileFP, offset, SEEK_SET);
#elif defined(__linux__) || defined(__APPLE__)
			fseeko(m_fileFP, offset, SEEK_SET);
#endif
```

```
                        long long readSize = fread(buffer, 1, size, m_fileFP);
                        return (readSize == size);
                }
                else
                {
                        if (!m_fileFP) return false;
                        if(0 != fseek(m_fileFP, offset, 0))
                                return false;
                        if(0 == fread(buffer, size, 1, m_fileFP))
                                return false;
                        return true;
                }
        }

        size_t ReadBlock(void* buffer, size_t size) {
                if (m_bLargeFile)
                {
#if defined(_WIN32) || defined(_WIN64)
                        _fseeki64(m_fileFP, 0, SEEK_SET);
#elif defined(__linux__) || defined(__APPLE__)
                        fseeko(m_fileFP, 0, SEEK_SET);
#endif
                        return fread(buffer, 1, size, m_fileFP);
                }
                else
                {
                        if (!m_fileFP) return false;
                        if(0 != fseek(m_fileFP, 0, 0))
                                return 0;
                        return fread(buffer, size, 1, m_fileFP);
                }
        }

        void Release()
        {
                if(m_fileFP)
                        fclose(m_fileFP);
                m_fileFP = NULL;
                delete this;
        }

private:
        FILE* m_fileFP;
        bool m_bLargeFile;
}
...
string inputPDFPath = "Sample.pdf";
CFSFile_Read* pFileRead = new CFSFile_Read();
If(!pFileRead->LoadFile(inputPDFPath.c_str()))
        Return;
PDFDoc doc = PDFDoc(pFileRead);
ErrorCode error_code = doc.Load();
if (error_code!= foxit::e_ErrSuccess) return 0;
```

3.2.5 How to load PDF document and get the first page of the PDF document

```
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"


using namespace foxit;
```

```
using namespace common;
using namespace pdf;
...

PDFDoc doc("Sample.pdf");
ErrorCode error_code = doc.Load();
if (error_code!= foxit::e_ErrSuccess) return 0;
PDFPage page = doc.GetPage(0);
page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);
```

3.2.6 How to save a PDF to a file

```
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...

PDFDoc doc("Sample.pdf");
ErrorCode error_code = doc.Load();
if (error_code!= foxit::e_ErrSuccess) return 0;
doc.SaveAs("new_Sample.pdf", PDFDoc::e_SaveFlagNoOriginal);
```

3.3 Page

PDF Page is the basic and important component of PDF Document. A foxit::pdf::PDFPage object is retrieved from a PDF document by function foxit::pdf::PDFDoc::GetPage. Page level APIs provide functions to parse, render, edit (includes creating, deleting, flattening and etc.) a page, retrieve PDF annotations, read and set the properties of a page, and etc. For most cases, A PDF page needs to be parsed before it is rendered or processed.

Example:

3.3.1 How to get page size

```
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...
// Assuming PDFPage page has been loaded and parsed.

int width = static_cast<int>(page.GetWidth());
int height = static_cast<int>(page.GetHeight());
```

3.3.2 How to calculate bounding box of page contents

```
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...
//Assuming PDFDoc doc has been loaded.
//Assuming PDFPage page has been loaded and parsed.

RectF ret = page.CalcContentBBox(PDFPage::e_CalcContentsBox);
...
```

3.3.3 How to create a PDF page and set the size

```
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...
// Assuming PDFDoc doc has been loaded.

PDFPage page = doc.InsertPage(index, PageWidth, PageHeight);
```

3.3.4 How to delete a PDF page

```
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...
// Assuming PDFDoc doc has been loaded.

// Remove a PDF page by page index.
doc.RemovePage(index);

// Remove a specified PDF page.
doc.RemovePage(&page);
...
```

3.3.5 How to flatten a PDF page

```
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
```

```
using namespace pdf;
...
// Assuming PDFPage page has been loaded and parsed.

// Flatten all contents of a PDF page.
page.Flatten(true, PDFPage::e_FlattenAll);
```

```
// Flatten a PDF page without annotations.
page.Flatten(true, PDFPage::e_FlattenNoAnnot);

// Flatten a PDF page without form controls.
page.Flatten(true, PDFPage::e_FlattenNoFormControl);

// Flatten a PDF page without annotations and form controls (Equals to nothing to be
flattened).
page.Flatten(true, PDFPage::e_FlattenNoAnnot | PDFPage::e_FlattenNoFormControl);
...
```

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

```
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace common;
using namespace pdf;
...
// Assuming PDFPage page has been loaded and parsed.

Bitmap bmp();
// Write bitmap data to the bmp object.
...
// Set thumbnails to the page.
page.SetThumbnail(bmp);
// Load thumbnails in the page.
Bitmap bitmap = page.LoadThumbnail();
...
```

3.4 Render

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

- To render page and annotations, first use function Renderer::SetRenderContentFlags to decide whether to render page and annotation both or not, and then use function Renderer::StartRender to do the rendering. Function Renderer::StartQuickRender can also be used to render page but only for thumbnail purpose.
- To render a single annotation, use function Renderer::RenderAnnot.
- To render on a bitmap, use function Renderer::StartRenderBitmap.

- To render a reflowed page, use function Renderer::StartRenderReflowPage.

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

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

Example:

3.4.1 How to render a page to a bitmap

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_render.h"

using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;

// Assuming PDFPage page has been loaded and parsed.

int width = static_cast<int>(page.GetWidth());
int height = static_cast<int>(page.GetHeight());
Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

// Prepare a bitmap for rendering.
Bitmap bitmap(width, height, Bitmap::e_DIBArgb, NULL, 0);
bitmap.FillRect(0xFFFFFFFF, NULL);
// Render page.
Renderer render(bitmap, false);
render.StartRender(page, matrix, NULL);
...
```

3.4.2 How to render page and annotation

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_render.h"

using namespace foxit;
```

```
using namespace common;
using namespace pdf;
using namespace foxit::common;

// Assuming PDFPage page has been loaded and parsed.
int width = static_cast<int>(page.GetWidth());
int height = static_cast<int>(page.GetHeight());
Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

// Prepare a bitmap for rendering.
Bitmap bitmap(width, height, Bitmap::e_DIBArgb, NULL, 0);
bitmap.FillRect(0xFFFFFFFF, NULL);

Renderer render(bitmap, false);
uint32 dwRenderFlag = Renderer::e_RenderAnnot | Renderer::e_RenderPage;
render.SetRenderContentFlags(dwRenderFlag);
render.StartRender(page, matrix, NULL);
...
```

3.5 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:

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

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfattachments.h"

using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;

// Assuming PDFDoc doc has been loaded.

// Get information of attachments.
Attachments attachments(doc);
int count = attachments.GetCount();
for (int i = 0; i < count; i++) {
    WString key = attachments.GetKey(i);
    FileSpec file_spec = attachments.GetEmbeddedFile(key);
    if (!file_spec.IsEmpty()) {
       WString name = file_spec.GetFileName();
       if (file_spec.IsEmbedded()) {
           WString exFilePath = "output_directory";
           file_spec.ExportToFile(exFilePath);
       }
```

```
    }
}
...
```

3.5.2 How to remove all the attachments of a PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfattachments.h"

...
using namespace foxit;
using namespace common;
using namespace pdf;
using namespace foxit::common;

// Assuming PDFDoc doc has been loaded.

// Get information of attachments.
Attachments attachments(doc);
int count = attachments.GetCount();
for (int i = 0; i < count; i++) {
     WString key = attachments.GetKey(i);
     attachment.RemoveEmbeddedFile(&key);
}
...
```

3.6 Text Page

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

- To search text in text contents of a PDF page, construct a TextSearch object with TextPage object.
- To access text such like hypertext link, construct a PageTextLinks object with TextPage object.

Example:

3.6.1 How to extract text from a PDF page

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_search.h"

using namespace std;
using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
```

```
using namespace pdf;
...

// Assuming PDFPage page has been loaded and parsed.

// Get the text page object.
TextPage text_page(page);
int count = text_page.GetCharCount();
if (count > 0) {
    WString text = text_page.GetChars();
    String s_text = text.UTF8Encode();
    fwrite((const char*)s_text, sizeof(char), s_text.GetLength(), file);
}
...
```

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

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_search.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

RectF rect;
rect.left = 90;
rect.right = 450;
rect.top = 595;
rect.bottom = 580;
TextPage textPage = new TextPage (&page, TextPage::e_ParseTextNormal);
textPage.GetTextInRect(&rect);
...
```

3.7 Text Search

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

- To specify the searching pattern and options, use functions TextSearch::SetPattern, TextSearch::SetStartPage (only useful for a text search in PDF document), TextSearch::SetEndPage (only useful for a text search in PDF document) and TextSearch::SetSearchFlags.
- To do the searching, use function TextSearch::FindNext or TextSearch::FindPrev.
- To get the searching result, use function TextSearch::GetMatchXXX().

Example:

3.7.1 How to search a text pattern in a PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_search.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

// Assuming PDFDoc doc has been loaded.

// Search for all pages of doc.
TextSearch search(doc, NULL);

int start_index = 0, end_index = doc.GetPageCount() - 1;
search.SetStartPage(start_index);
search.SetEndPage(end_index);

WString pattern = L"Foxit";
search.SetPattern(pattern);

foxit::uint32 flags = TextSearch::e_SearchNormal;
search.SetSearchFlags(flags);
...
int match_count = 0;
while (search.FindNext()) {
      RectFArray rect_array = search.GetMatchRects();
      match_count ++;
    }
...
```

3.8 Text Link

In a PDF page, some text contents that represent a hypertext link to a website or a resource on the intent, or an email address are the same with common texts. Prior to text link processing, user should first call PageTextLinks::GetTextLink to get a textlink object.

Example:

3.8.1 How to retrieve hyperlinks in a PDF page

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
```

```
#include "include/pdf/fs_search.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

// Assuming PDFPage page has been loaded and parsed.

// Get the text page object.
TextPage text_page(page);
PageTextLinks pageTextLink(text_page);
TextLink textLink = pageTextLink.GetTextLink(index);
String strURL = textLink.GetURI();
...
```

3.9 Bookmark

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

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

- To access the parent bookmark, use function Bookmark::GetParent.
- To access the first child bookmark, use function Bookmark::GetFirstChild.
- To access the next sibling bookmark, use function Bookmark::GetNextSibling.
- To insert a new bookmark, use function Bookmark::Insert.
- To move a bookmark, use function Bookmark::MoveTo.

Example:

3.9.1 How to find and list all bookmarks of a PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_filespec.h"
#include "include/pdf/fs_bookmark.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
```

```
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

// Assuming PDFDoc doc has been loaded.

Bookmark root = doc.GetRootBookmark();
Bookmark first_bookmark = root.GetFirstChild();

if (first_bookmark != null)
{
       TraverseBookmark(first_bookmark, 0);
}

Private void TraverseBookmark(Bookmark root, int iLevel)
{
       if (root != null)
       {
              Bookmark child = root.GetFirstChild();
              while (child != null)
              {
                 TraverseBookmark(child, iLevel + 1);
                 child = child.GetNextSibling();
              }
        }
}
...
```

3.10 Form (AcroForm)

PDF currently supports two different forms for gathering information interactively from the user - AcroForms and XFA forms. Acroforms are the original PDF-based fillable forms, based on the PDF architecture. Foxit PDF SDK provides APIs to view and edit form field programmatically. Form fields are commonly used in PDF documents to gather data. The Form class offers functions to retrieve form fields or form controls, import/export form data and other features, for example:

- To retrieve form fields, please use functions Form::GetFieldCount and Form::GetField.
- To retrieve form controls from a PDF page, please use functions Form::GetControlCount and Form::GetControl.
- To import form data from an XML file, please use function Form::ImportFromXML; to export form data to an XML file, please use function Form::ExportToXML.
- To retrieve form filler object, please use function Form::GetFormFiller.

To import form data from a FDF/XFDF file or export such data to a FDF/XFDF file, please refer to functions pdf::PDFDoc::ImportFromFDF and pdf::PDFDoc::ExportToFDF.

Example:

3.10.1 How to load the forms in a PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfform.h"

using namespace foxit;
using namespace pdf;
using namespace interform;
...

// Assuming PDFDoc doc has been loaded.

bool hasForm = doc.HasForm();
if(hasForm)
   Form form(doc);
...
```

3.10.2 How to count form fields and get the properties

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/interform/fs_pdfform.h"

using namespace foxit;
using namespace pdf;
using namespace interform;

...
// Assuming PDFDoc doc has been loaded.

Form form(doc);
int countFields = form. GetFieldCount(NULL);
for (int i = 0; i < nFieldCount; i++)
{
     Field field = form.GetField(i, filter);
     Field::Type type = field.GetType();
     WString org_alternateName = field.GetAlternateName();
     field.SetAlternateName(L"signature");
}
```

3.10.3 How to export the form data in a PDF to a XML file

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/interform/fs_pdfform.h"

using namespace foxit;
using namespace pdf;
using namespace interform;
...
// Assuming PDFDoc doc has been loaded.
```

```
Form form(doc);
...
form.ExportToXML(XMLFilePath);
...
```

3.10.4 How to import form data from a XML file

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/interform/fs_pdfform.h"

using namespace foxit;
using namespace pdf;
using namespace interform;
...
// Assuming PDFDoc doc has been loaded.

Form form(doc);
...
form.ImportFromXML(XMLFilePath);
...
```

3.11 XFA Form

XFA (XML Forms Architecture) forms are XML-based forms, wrapped inside a PDF. The XML Forms Architecture provides a template-based grammar and a set of processing rules that allow uses to build interactive forms. At its simplest, a template-based grammar defines fields in which a user provides data.

Foxit PDF SDK provides APIs to render the XFA form, fill the form, export or import form's data.

***Note**: Foxit PDF SDK provides two callback classes foxit::addon::xfa::AppProviderCallback and foxit::addon::xfa::DocProviderCallback to represent the callback objects as an XFA document provider and an XFA application provider respectively. All the functions in those classes are used as callback functions. Pure virtual functions should be implemented by users.*

Example:

3.11.1 How to load XFA Doc and represent an Interactive XFA form

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"

#include "include/pdf/interform/fs_pdfform.h"
#include "include/addon/xfa/fs_xfa.h"

using namespace foxit;
using namespace foxit::common;
```

```
using namespace pdf;
using namespace foxit::addon::xfa;

CFS_XFAAppHandler* pXFAAppHandler = new CFS_XFAAppHandler(); // implement from
foxit::addon::xfa::AppProviderCallback
Library::RegisterXFAAppProviderCallback(pXFAAppHandler);
WString input_file = input_path + L"xfa_dynamic.pdf";
PDFDoc doc(input_file);
ErrorCode error_code = doc.Load();
if (error_code != foxit::e_ErrSuccess) {
	return 1;
}

CFS_XFADocHandler* pXFADocHandler = new CFS_XFADocHandler(); // implement from
foxit::addon::xfa::DocProviderCallback
XFADoc xfa_doc(doc, pXFADocHandler);
xfa_doc.StartLoad(NULL);
...
```

3.11.2 How to export and import XFA form data

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/interform/fs_pdfform.h"
#include "include/addon/xfa/fs_xfa.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace foxit::addon::xfa;

// Assuming FSXFADoc xfa_doc has been loaded.

xfa_doc.ExportData(L"xfa_form.xml", XFADoc::e_ExportDataTypeXML);

xfa_doc.ResetForm();
doc.SaveAs( L"xfa_dynamic_resetform.pdf");

xfa_doc.ImportData(L"xfa_form.xml");
doc.SaveAs(L"xfa_dynamic_importdata.pdf");
...
```

3.12 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. To fill the form, please construct a Filler object by current Form object or retrieve the Filler object by function Form::GetFormFiller if such object has been constructed. (There should be only one form filler object for an interactive form). For how to fill a form with form filler, you can refer to the view demo "**view_demo**" in the "examples\simple_demo" folder of the download package for Windows.

3.13 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:

3.13.1 How to add a text form field to a PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/interform/fs_pdfform.h"

using namespace foxit;
using namespace pdf;
using namespace interform;
...
// Assuming PDFDoc doc has been loaded.
// Assuming PDFPage page has been loaded and parsed.

// Add text field
Control control = form.AddControl(page, L"Text Field0", Field::e_TypeTextField, RectF(50, 600,
90, 640));
control.GetField().SetValue(L"3");
// Update text field's appearance.
control.GetWidget().ResetAppearanceStream();

Control control1 = form.AddControl(page, L"Text Field1", Field::e_TypeTextField, RectF(100,
600, 140, 640));
control1.GetField().SetValue(L"123");
// Update text field's appearance.
control1.GetWidget().ResetAppearanceStream();
...
```

3.13.2 How to remove a text form field from a PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/interform/fs_pdfform.h"

using namespace foxit;
using namespace pdf;
using namespace interform;
...
// Assuming PDFDoc doc has been loaded.

Form form(doc);
```

```
const wchar_t* filter = L"text1";
int countFields = form.GetFieldCount(NULL);
for (int i = 0; i < countFields; i++)
{
     Field field = form.GetField(i, filter);
     if (field.GetType() == Field::e_TypeTextField) {
       form.RemoveField(field);
     }
}
...
```

3.14 Annotations

3.14.1 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 3-1. 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 3-1 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 3-1

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

Ink(pencil)	Ink annotation	Yes	Yes
Popup	Popup annotation	No	Yes
File Attachment	FileAttachment annotation	Yes	Yes
Sound	Sound annotation	Yes	No
Movie	Movie annotation	No	No
Widget*	Widget annotation	No	Yes
Screen	Screen annotation	No	Yes
PrinterMark	PrinterMark annotation	No	No
TrapNet	Trap network annotation	No	No
Watermark*	Watermark annotation	No	Yes
3D	3D annotation	No	No
Redact	Redact annotation	Yes	Yes

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:

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

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace annots;

// Assuming PDFPage page has been loaded and parsed.
// Assuming the annnots in the page have been loaded.

// Add link annotation.
annots::Link link(page.AddAnnot( Annot::e_Link, RectF(350,350,380,400)));
link.SetHighlightingMode(Annot::e_HighlightingToggle);
```

```
...
```

3.14.1.2 How to add a highlight annotation to a page and set the related annotation properties

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace annots;

// Assuming PDFPage page has been loaded and parsed.
// Assuming the annnots in the page have been loaded.

// Add highlight annotation.
annots::Highlight highlight(page.AddAnnot(Annot::e_Highlight,RectF(10,450,100,550)));
highlight.SetContent(L"Highlight");
annots::QuadPoints quad_points;
quad_points.first = PointF(10, 500);
quad_points.second = PointF(90, 500);
quad_points.third = PointF(10, 480);
quad_points.fourth = PointF(90, 480);
annots::QuadPointsArray quad_points_array;
quad_points_array.Add(quad_points);
highlight.SetQuadPoints(quad_points_array);
highlight.SetSubject(L"Highlight");
highlight.SetTitle(L"Foxit SDK");
highlight.SetCreationDateTime(GetLocalDateTime());
highlight.SetModifiedDateTime(GetLocalDateTime());
highlight.SetUniqueID(WString::FromLocal(RandomUID()));

// Appearance should be reset.
highlight.ResetAppearanceStream();
...
```

3.14.1.3 How to set the popup information when creating markup annotations

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace annots;
```

```
// Assuming PDFPage page has been loaded and parsed.
// Assuming the annnots in the page have been loaded.

// Create a new note annot and set the properties for it.
annots::Note note(page.AddAnnot(Annot::e_Note, RectF(10,350,50,400)));
note.SetIconName("Comment");
note.SetSubject(L"Note");
note.SetTitle(L"Foxit SDK");
note.SetContent(L"Note annotation.");
note.SetCreationDateTime(GetLocalDateTime());
note.SetModifiedDateTime(GetLocalDateTime());
note.SetUniqueID(WString::FromLocal(RandomUID()));

// Create a new popup annot and set it to the new note annot.
Popup popup(page.AddAnnot(Annot::e_Popup, RectF(300,450,500,550)));
popup.SetBorderColor(0x00FF00);
popup.SetOpenStatus(false);
popup.SetModifiedDateTime(GetLocalDateTime());
note.SetPopup(popup);
...
```

3.14.1.4 How to get a specific annotation in a PDF using device coordinates

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace annots;

// Assuming PDFDoc doc has been loaded.
// Assuming PDFPage page has been loaded and parsed.
...

int width = static_cast<int>(page.GetWidth());
int height = static_cast<int>(page.GetHeight());

// Get page transformation matrix.
Matrix displayMatrix= page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());
int iAnnotCount = page.GetAnnotCount();

for(int i=0; i<iAnnotCount; i++)
{
	Annot pAnnot = page.GetAnnot(i);
	ASSERT_FALSE(pAnnot.IsEmpty());
	if (Annot::e_Popup == pAnnot.GetType()) continue;
```

```
        RectI annotRect = pAnnot.GetDeviceRect(false, displayMatrix);
        PointF pt;
        float tolerance = 1.0;

        // Get the same annot (pAnnot) using annotRect.
        pt.x = annotRect.left + tolerance;
        pt.y = (annotRect.top - annotRect.bottom)/2 + annotRect.bottom;
        Annot gAnnot = page.GetAnnotAtDevicePoint(pt, tolerance, &displayMatrix);
        ...
}
```

3.14.2 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:

3.14.2.1 How to load annotations from a FDF file and add them into the first page of a given PDF

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"
#include "include/pdf/fs_pdfpage.h"

using namespace std;
using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace annots;

// Assuming PDFDoc doc has been loaded.
...

FILE* file = NULL;
#if defined(_WIN32) || defined(_WIN64)
        fopen_s(&file, (const char*)(const char*)String::FromUnicode(fdf_file), "rb+");
#else
        file = fopen((const char*)(const char*)String::FromUnicode(fdf_file), "rb+");
#endif
fseek(file, 0, SEEK_END);
size_t file_size = (size_t)ftell(file);
char* buffer = (char*)malloc(file_size * sizeof(char));
memset(buffer, 0 , file_size);

fseek(file, 0, SEEK_SET);
fread(buffer, sizeof(char), file_size, file);
fclose(file);

fdf::FDFDoc fdf_doc(buffer, file_size);
pdf_doc.ImportFromFDF(fdf_doc, PDFDoc::e_Annots);
```

3.15 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:

3.15.1 How to convert PDF pages to bitmap files.

```
#include "include/common/fs_common.h"
#include "include/common/fs_image.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_render.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;

// Assuming PDFDoc doc has been loaded.
...

// Get page count
int nPageCount = doc.GetPageCount();
for(int i=0;i<nPageCount;i++) {
    PDFPage page = doc.GetPage(i);

    // Parse page.
    page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);

    int width = static_cast<int>(page.GetWidth());
    int height = static_cast<int>(page.GetHeight());
    Matrix matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

    // Prepare a bitmap for rendering.
    Bitmap bitmap(width, height, foxit::common::Bitmap::e_DIBArgb, NULL, 0);
    bitmap.FillRect(0xFFFFFFFF, NULL);

    // Render page.
    Renderer render(bitmap, false);
    render.StartRender(page, matrix, NULL);
    image.AddFrame(bitmap);
}
...
```

3.15.2 How to convert an image file to PDF file

```
#include "include/common/fs_common.h"
#include "include/common/fs_image.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
```

```
using namespace foxit;
using namespace foxit::common;
using namespace pdf;

Image image(input_file);
int count = image.GetFrameCount();

PDFDoc doc;
for (int i = 0; i < count; i++) {
    PDFPage page = doc.InsertPage(i);
    page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);
    // Add image to page.
    page.AddImage(image, i, PointF(0, 0), page.GetWidth(), page.GetHeight(), true);
}

doc.SaveAs(output_file, PDFDoc::e_SaveFlagNoOriginal);
...
```

3.16 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:

3.16.1 How to create a text watermark and insert it into the first page

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_watermark.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

// Assuming PDFDoc doc has been loaded.

WatermarkSettings settings;
settings.flags = WatermarkSettings::e_FlagASPageContents | WatermarkSettings::e_FlagOnTop;
settings.offset_x = 0;
settings.offset_y = 0;
settings.opacity = 90;
settings.position = common::e_PosTopRight;
settings.rotation = -45.f;
settings.scale_x = 1.f;
settings.scale_y = 1.f;
```

```
WatermarkTextProperties text_properties;
text_properties.alignment = CommonDefines::e_AlignmentCenter;
text_properties.color = 0xF68C21;
text_properties.font_style = WatermarkTextProperties::e_FontStyleNormal;
text_properties.line_space = 1;
text_properties.font_size = 12.f;
text_properties.font = Font(Font::e_StdIDTimesB);

Watermark watermark(doc, L"Foxit PDF SDK\nwww.foxitsoftware.com", text_properties, settings);
watermark.InsertToPage(doc.GetPage(0));

// Save document to file
...
```

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

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_watermark.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

// Assuming PDFDoc doc has been loaded.

WatermarkSettings settings;
settings.flags = WatermarkSettings::e_FlagASPageContents | WatermarkSettings::e_FlagOnTop;
settings.offset_x = 0.f;
settings.offset_y = 0.f;
settings.opacity = 20;
settings.position = common::e_PosCenter;
settings.rotation = 0.0f;

Image image(image_file);
Bitmap bitmap = image.GetFrameBitmap(0);
settings.scale_x = page.GetWidth() * 0.618f / bitmap.GetWidth();
settings.scale_y = settings.scale_x;

Watermark watermark(doc, image, 0, settings);
watermark.InsertToPage(doc.GetPage(0));

// Save document to file.
...
```

3.16.3 How to remove all watermarks from a page

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
```

```
using namespace pdf;
...
// Assuming PDFPage page has been loaded and parsed.
...
page.RemoveAllWatermarks();
...
// Save document to file
...
```

3.17 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 3-2.

Table 3-2

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

Example:

3.17.1 How to generate a barcode bitmap from a string

```
#include "include/common/fs_common.h"
#include "include/common/fs_barcode.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
...

// Strings used as barcode content.
WString sz_code_string = L"TEST-SHEET";

// Barcode format types.
Barcode::Format code_format = Barcode::e_FormatCode39;

//Format error correction level of QR code.
Barcode::QRErrorCorrectionLevel sz_qr_level = Barcode::e_QRCorrectionLevelLow;

//Image names for the saved image files for QR code.
WString bmp_qr_name = L"/QR_CODE_TestForBarcodeQrCode_L.bmp";
```

```
// Unit width for barcode in pixels, preferred value is 1-5 pixels.
int unit_width = 2;

// Unit height for barcode in pixels, preferred value is >= 20 pixels.
int unit_height = 120;

Barcode barcode;
Bitmap bitmap = barcode.GenerateBitmap(sz_code_string, code_format, unit_width, unit_height,
sz_qr_level);
...
```

3.18 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. It also 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.

***Note**: For more detailed information about the RMS encryption and decryption, please refer to the simple demo "**security**" in the* "examples\simple_demo" folder of the download package.

Example:

3.18.1 How to encrypt a PDF file with Certificate

```
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_security.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...
PDFDoc doc(input_file);
ErrorCode error_code = doc.Load();
    if (error_code != foxit::e_ErrSuccess) {
      return false;
    }

// Do encryption.
StringArray envelopes;
String initial_key;
WString cert_file_path = input_path + L"foxit.cer";
if (!GetCertificateInfo((const char*)String::FromUnicode(cert_file_path), envelopes,
initial_key, true, 16)) {
     return false;
}
CertificateSecurityHandler handler;
CertificateEncryptData encrypt_data(true, SecurityHandler::e_CipherAES, envelopes);
handler.Initialize(encrypt_data, initial_key);
```

```
doc.SetSecurityHandler(handler);
WString output_file = output_directory + L"certificate_encrypt.pdf";
doc.SaveAs(output_file, PDFDoc::e_SaveFlagNoOriginal);
...
```

3.18.2 How to encrypt a PDF file with Foxit DRM

```
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_security.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
...

PDFDoc doc(input_file);
ErrorCode error_code = doc.Load();
if (error_code != foxit::e_ErrSuccess) {
   return false;
}

// Do encryption.
DRMSecurityHandler handler = DRMSecurityHandler();
const char* file_id = "Simple-DRM-file-ID";
String initialize_key = "Simple-DRM-initialize-key";
DRMEncryptData encrypt_data(true, "Simple-DRM-filter", SecurityHandler::e_CipherAES, 16, true,
0xfffffffc);
handler.Initialize(encrypt_data, file_id, initialize_key);
doc.SetSecurityHandler(handler);

WString output_file = output_directory + L"foxit_drm_encrypt.pdf";
doc.SaveAs(output_file, PDFDoc::e_SaveFlagNoOriginal);
...
```

3.19 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:

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

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_reflowpage.h"
#include "include/common/fs_render.h"

using namespace foxit;
using namespace foxit::common;
```

```
using foxit::common::Library;
using namespace pdf;
...

// Assuming PDFDoc doc has been loaded.

PDFPage page = doc.GetPage(0);
// Parse PDF page.
page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);

ReflowPage reflow_page(page);

// Set some arguments used for parsing the reflow page.
reflow_page.SetLineSpace(0);
reflow_page.SetZoom(100);
reflow_page.SetParseFlags(ReflowPage::e_Normal);

// Parse reflow page.
reflow_page.StartParse(NULL);

// Get actual size of content of reflow page. The content size does not contain the margin.
float content_width = reflow_page.GetContentWidth();
float content_height = reflow_page.GetContentHeight();

// Assuming Bitmap bitmap has been created.

// Render reflow page.
Renderer renderer(bitmap, false);
foxit::Matrix matrix = reflow_page.GetDisplayMatrix(0, 0);
renderer.StartRenderReflowPage(reflow_page, matrix, NULL);
...
```

3.20 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. For how to open and parse pages with asynchronous mode, you can refer to the simple demo "**async_load**" in the "examples\simple_demo" folder of the download package.

3.21 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:

3.21.1 How to create a PSI and set the related properties for it

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_psi.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace annots;

PSI psi(480, 180, true);

// Set ink diameter.
psi.SetDiameter(9);

// Set ink color.
psi.SetColor(0x434236);

// Set ink opacity.
psi.SetOpacity(0.8f);

// Add points to pressure sensitive ink.
float x = 121.3043f;
float y = 326.6846f;
float pressure = 0.0966f;
Path::PointType type = Path::e_TypeMoveTo;
psi.AddPoint(PointF(x, y), type, pressure);
...
```

3.22 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:

3.22.1 How to open a document including wrapper data.

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_render.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace common::file;
```

```
// file_name is PDF document which includes wrapper data.
PDFDoc doc(file_name);
ErrorCode code = doc.Load();
if (code != foxit::e_ErrSuccess) {
    return false;
}
if(!doc.IsWrapper()){
    return false;
}
int64 offset = doc.GetWrapperOffset();

FileReader file_reader(offset);
file_reader.LoadFile(String::FromUnicode(file_name));
...
```

3.23 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 3.25) 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:

3.23.1 How to remove some properties from catalog dictionary

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace objects;
...

// Assuming PDFDoc doc has been loaded.

PDFDictionary* catalog = doc.GetCatalog();
if (NULL == catalog) return;

const char* key_strings[] = { "Type", "Boolean", "Name", "String", "Array", "Dict"};
int count = sizeof(key_strings)/sizeof(key_strings[0]);
for (int i = 0; i < count; i ++) {
  if (catalog->HasKey(key_strings[i]))
      catalog->RemoveAt(key_strings[i]);
}
...
```

3.24 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 3.24 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:

3.24.1 How to create a text object in a PDF page

```
#include "include/common/fs_common.h"
#include "include/pdf/graphics/fs_pdfgraphicsobject.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"


using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace graphics;
...

// Assuming PDFPage page has been loaded and parsed.

POSITION position = page.GetLastGraphicsObjectPosition(GraphicsObject::e_TypeText);
TextObject* text_object = TextObject::Create();

text_object->SetFillColor(0xFFFF7F00);

// Prepare text state.
TextState state;
state.font_size = 80.0f;
state.font =  Font(L"Simsun", Font::e_StylesSmallCap, Font::e_CharsetGB2312, 0);
state.textmode = TextState::e_ModeFill;
text_object->SetTextState(page, state, false, 750);

// Set text.
text_object->SetText(L"Foxit Software");
POSITION last_position = page.InsertGraphicsObject(position, text_object);
...
```

3.24.2 How to add an image logo to a PDF page

```
#include "include/common/fs_common.h"
#include "include/pdf/graphics/fs_pdfgraphicsobject.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
```

```
using foxit::common::Library;
using namespace pdf;
using namespace graphics;
...
//Assuming PDFPage page has been loaded and parsed.

POSITION position = page.GetLastGraphicsObjectPosition(GraphicsObject::e_TypeImage);
Image image(image_file);
ImageObject* image_object = ImageObject::Create(page.GetDocument());
image_object->SetImage(image, 0);

float width = static_cast<float>(image.GetWidth());
float height = static_cast<float>(image.GetHeight());

float page_width = page.GetWidth();
float page_height = page.GetHeight();

// Please notice the matrix value.
image_object->SetMatrix(Matrix(width, 0, 0, height, (page_width - width) / 2.0f, (page_height
- height) / 2.0f));

page.InsertGraphicsObject(position, image_object);
page.GenerateContent();
...
```

3.25 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:

3.25.1 How to get marked content in a page and get the tag name

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_image.h"
#include "include/pdf/graphics/fs_pdfgraphicsobject.h"
#include "include/pdf/objects/fs_pdfobject.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace graphics;
using namespace objects;

...
// Assuming PDFPage page has been loaded and parsed.

POSITION position = page.GetFirstGraphicsObjectPosition(GraphicsObject::e_TypeText);
```

```
TextObject* text_obj = reinterpret_cast<TextObject*>(page.GetGraphicsObject(position));
MarkedContent* content = text_obj->GetMarkedContent();
int item_count = content->GetItemCount();

// Get marked content property
for (int i = 0; i < item_count; i++) {
    String tag_name = content->GetItemTagName(i);
    int mcid = content->GetItemMCID(i);
}
...
```

3.26 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.

In Foxit PDF SDK, a PDF layer is associated with a layer node. To retrieve a layer node, user should construct a PDF LayerTree object first and then call function LayerTree::GetRootNode to get the root layer node of the whole layer tree. Furthermore, you can enumerate all the nodes in the layer tree from the root layer node. Foxit PDF SDK provides APIs to get/set layer data, view or hide the contents in different layers, set layers' name, add or remove layers, and edit layers.

Example:

3.26.1 How to create a PDF layer

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/common/fs_render.h"
#include "include/pdf/fs_pdflayer.h"

using namespace foxit;
using namespace pdf;
 ...
// Assuming PDFDoc doc has been loaded.

LayerTree layertree(doc);
LayerNode root = layertree.GetRootNode();
if (root.IsEmpty()) {
       printf("No layer information!\r\n");
       return ;
}
...
```

3.26.2 How to set all the layer nodes information

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_render.h"
```

```
#include "include/pdf/fs_pdflayer.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
 ...
// Assuming PDFDoc doc has been loaded.
// Assuming PDFpage page has been loaded and parsed.

void SetAllLayerNodesInformation(LayerNode layer_node) {
  if (layer_node.HasLayer()) {
    layer_node.SetDefaultVisible(true);
    layer_node.SetExportUsage(LayerTree::e_StateUndefined);
    layer_node.SetViewUsage(LayerTree::e_StateOFF);
    LayerPrintData print_data("subtype_print", LayerTree::e_StateON);
    layer_node.SetPrintUsage(print_data);
    LayerZoomData zoom_data(1, 10);
    layer_node.SetZoomUsage(zoom_data);
    WString new_name = WString(L"[View_OFF_Print_ON_Export_Undefined]") +
layer_node.GetName();
    layer_node.SetName(new_name);
  }
  int count = layer_node.GetChildrenCount();
  for (int i = 0; i < count; i++) {
    LayerNode child = layer_node.GetChild(i);
    SetAllLayerNodesInformation(child);
  }
}

LayerTree layertree(doc);
LayerNode root = layertree.GetRootNode();
SetAllLayerNodesInformation(root);
...
```

3.26.3 How to edit layer tree

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/common/fs_render.h"
#include "include/pdf/fs_pdflayer.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
 ...
// Assuming PDFDoc doc has been loaded.

// edit layer tree
PDFDoc doc(input_file);
error_code = doc.Load();
layertree = LayerTree(doc);
root = layertree.GetRootNode();
int children_count = root.GetChildrenCount();
root.RemoveChild(children_count -1);
LayerNode child = root.GetChild(children_count - 2);
LayerNode child0 = root.GetChild(0);
child.MoveTo(child0, 0);
```

```
child.AddChild(0, L"AddedLayerNode", true);
child.AddChild(0, L"AddedNode", false);
```

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

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

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

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

Example:

3.27.1 How to sigh the PDF document with a signature

```
#include "include/pdf/annots/fs_annot.h"
#include "include/common/fs_image.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_signature.h"

using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace objects;
using namespace file;

// AdobePPKLiteSignature
const char* filter = "Adobe.PPKLite";
const char* sub_filter = "adbe.pkcs7.detached";

if (!use_default) {
	InitializeOpenssl();
	sub_filter = "adbe.pkcs7.sha1";
	SignatureCallbackImpl* sig_callback = new SignatureCallbackImpl(sub_filter);
	Library::RegisterSignatureCallback(filter, sub_filter, sig_callback);
}

printf("Use signature callback object for filter \"%s\" and sub-filter \"%s\"\r\n",
	filter, sub_filter);
```

```
PDFPage pdf_page = pdf_doc.GetPage(0);
// Add a new signature to the first page.
Signature new_signature = AddSiganture(pdf_page, sub_filter);
// Set filter and subfilter for the new signature.
new_signature.SetFilter(filter);
new_signature.SetSubFilter(sub_filter);
bool is_signed = new_signature.IsSigned();
uint32 sig_state = new_signature.GetState();
printf("[Before signing] Signed?:%s\t State:%s\r\n",
         is_signed? "true" : "false",
         TransformSignatureStateToString(sig_state).c_str());

// Sign the new signature.
WString signed_pdf_path = output_directory + L"signed_newsignature.pdf";
if (use_default)
       signed_pdf_path = output_directory + L"signed_newsignature_default_handler.pdf";

WString cert_file_path = input_path + L"foxit_all.pfx";
WString cert_file_password = L"123456";
// Cert file path will be passed back to application through callback function
FSSignatureCallback::Sign().
// In this demo, the cert file path will be used for signing in callback function
FSSignatureCallback::Sign().
new_signature.StartSign((const wchar_t*)cert_file_path, cert_file_password,
         Signature::e_DigestSHA1, (const wchar_t*)signed_pdf_path, NULL, NULL);
printf("[Sign] Finished!\r\n");
is_signed = new_signature.IsSigned();
sig_state = new_signature.GetState();
printf("[After signing] Signed?:%s\tState:%s\r\n",
         is_signed? "true" : "false",
         TransformSignatureStateToString(sig_state).c_str());

// Open the signed document and verify the newly added signature (which is the last one).
printf("Signed PDF file: %s\r\n", (const char*)String::FromUnicode(signed_pdf_path));
PDFDoc signed_pdf_doc((const wchar_t*)signed_pdf_path);
ErrorCode error_code = signed_pdf_doc.Load(NULL);
if (foxit::e_ErrSuccess !=error_code ) {
       printf("Fail to open the signed PDF file.\r\n");
       return;
}
// Get the last signature which is just added and signed.
int sig_count = signed_pdf_doc.GetSignatureCount();
Signature signed_signature = signed_pdf_doc.GetSignature(sig_count-1);
// Verify the signature.
signed_signature.StartVerify(NULL, NULL);
printf("[Verify] Finished!\r\n");
is_signed = signed_signature.IsSigned();
sig_state = signed_signature.GetState();
printf("[After verifying] Signed?:%s\tState:%s\r\n",
         is_signed? "true" : "false",
         TransformSignatureStateToString(sig_state).c_str());
```

3.27.2 How to implement signature callback function of signing

```
#include "include/pdf/annots/fs_annot.h"
#include "include/common/fs_image.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
#include "include/pdf/fs_signature.h"
```

```
using namespace foxit;
using namespace foxit::common;
using foxit::common::Library;
using namespace pdf;
using namespace objects;
using namespace file;

// Implementation of pdf::SignatureCallback
class SignatureCallbackImpl : public pdf::SignatureCallback {
public:
	SignatureCallbackImpl(string subfilter)
		: sub_filter_(subfilter)
		, digest_context_(NULL) {}
	~SignatureCallbackImpl();

	virtual void Release() {
		delete this;
	}
	virtual bool StartCalcDigest(const ReaderCallback* file, const uint32*
byte_range_array,
		uint32 size_of_array, const Signature& signature, const void* client_data);
	virtual Progressive::State ContinueCalcDigest(const void* client_data, const
PauseCallback* pause);
	virtual String GetDigest(const void* client_data);
	virtual String Sign(const void* digest, uint32 digest_length, const wchar_t* cert_path,
		const WString& password, Signature::DigestAlgorithm digest_algorithm,
		void* client_data);
	virtual uint32 VerifySigState(const void* digest, uint32 digest_length,
		const void* signed_data, uint32 signed_data_len,
		void* client_data);
	virtual bool IsNeedPadData() {return false;}
protected:
	bool GetTextFromFile(unsigned char *plainString);

	unsigned char* PKCS7Sign(const wchar_t* cert_file_path, String cert_file_password,
		String plain_text, int& signed_data_size);
	bool PKCS7VerifySignature(String signed_data, String plain_text);
	bool ParseP12File(const wchar_t* cert_file_path, String cert_file_password,
		EVP_PKEY** pkey, X509** x509, STACK_OF(X509)** ca);
	ASN1_INTEGER* CreateNonce(int bits);

private:
	string sub_filter_;
	DigestContext* digest_context_;

	string cert_file_path_;
	string cert_file_password_;
};

SignatureCallbackImpl::~SignatureCallbackImpl() {
	if (digest_context_) {
		delete digest_context_;
		digest_context_ = NULL;
	}
}

bool SignatureCallbackImpl::GetTextFromFile(unsigned char* file_buffer) {
	if (!digest_context_ || !digest_context_->GetFileReadCallback()) return false;
	ReaderCallback* file_read = digest_context_->GetFileReadCallback();
	file_read->ReadBlock(file_buffer, digest_context_->GetByteRangeElement(0),
digest_context_->GetByteRangeElement(1));
```

```
	file_read->ReadBlock(file_buffer + (digest_context_->GetByteRangeElement(1)-
digest_context_->GetByteRangeElement(0)),
		digest_context_->GetByteRangeElement(2),
digest_context_->GetByteRangeElement(3));
	return true;
}

bool SignatureCallbackImpl::StartCalcDigest(const ReaderCallback* file, const uint32*
byte_range_array,
	uint32 size_of_array, const Signature& signature, const void* client_data) {
		if (digest_context_) {
			delete digest_context_;
			digest_context_ = NULL;
		}
		digest_context_ = new DigestContext(const_cast<ReaderCallback*>(file),
byte_range_array, size_of_array);
		if(!SHA1_Init(&digest_context_->sha_ctx_)) {
			delete digest_context_;
			digest_context_ = NULL;
			return false;
		}
		return true;
}

Progressive::State SignatureCallbackImpl::ContinueCalcDigest(const void* client_data, const
PauseCallback* pause) {
	if (!digest_context_) return Progressive::e_Error;

	uint32 file_length = digest_context_->GetByteRangeElement(1) +
digest_context_->GetByteRangeElement(3);
	unsigned char* file_buffer = (unsigned char*)malloc(file_length);
	if (!file_buffer || !GetTextFromFile(file_buffer)) return Progressive::e_Error;

	SHA1_Update(&digest_context_->sha_ctx_, file_buffer, file_length);
	free(file_buffer);
	return Progressive::e_Finished;
}

String SignatureCallbackImpl::GetDigest(const void* client_data) {
	if (!digest_context_) return "";
	unsigned char* md = reinterpret_cast<unsigned
char*>(OPENSSL_malloc((SHA_DIGEST_LENGTH)*sizeof(unsigned char)));
	if (1 != SHA1_Final(md, &digest_context_->sha_ctx_))
		return "";
	String digest = String(reinterpret_cast<const char*>(md), SHA_DIGEST_LENGTH);
	OPENSSL_free(md);
	return digest;
}

String SignatureCallbackImpl::Sign(const void* digest, uint32 digest_length, const wchar_t*
cert_path,
	const WString& password, Signature::DigestAlgorithm digest_algorithm,
	void* client_data) {
		if (!digest_context_) return "";
		String plain_text;
		if ("adbe.pkcs7.sha1" == sub_filter_) {
			plain_text = String((const char*)digest, digest_length);
		}
		int signed_data_length = 0;
		unsigned char* signed_data_buffer = PKCS7Sign(cert_path,
String::FromUnicode(password),
```

```
                        plain_text, signed_data_length);
                if (!signed_data_buffer) return "";

                String signed_data = String((const char*)signed_data_buffer,
signed_data_length);
                free(signed_data_buffer);
                return signed_data;
}

uint32 SignatureCallbackImpl::VerifySigState(const void* digest, uint32 digest_length,
        const void* signed_data, uint32 signed_data_len, void* client_data) {
                // Usually, the content of a signature field is contain the certification of
signer.
                // But we can't judge this certification is trusted.
                // For this example, the signer is ourself. So when using api PKCS7_verify to
verify,
                // we pass NULL to it's parameter <i>certs</i>.
                // Meanwhile, if application should specify the certificates, we suggest pass
flag PKCS7_NOINTERN to
                // api PKCS7_verify.
                if (!digest_context_) return Signature::e_StateVerifyErrorData;
                String plain_text;
                unsigned char* file_buffer = NULL;
                if ("adbe.pkcs7.sha1" == sub_filter_) {
                        plain_text = String(reinterpret_cast<const char*>(digest),
digest_length);
                } else {
                        return Signature::e_StateUnknown;
                }

                String signed_data_str = String(reinterpret_cast<const char*>(signed_data),
signed_data_len);
                bool ret = PKCS7VerifySignature(signed_data_str, plain_text);
                if (file_buffer) free(file_buffer);
                return ret ? Signature::e_StateVerifyValid : Signature::e_StateVerifyInvalid;

}

ASN1_INTEGER* SignatureCallbackImpl::CreateNonce(int bits) {
        unsigned char buf[20];
        int len = (bits - 1) / 8 + 1;
        // Generating random byte sequence.
        if (len > (int)sizeof(buf)) {
                return NULL;
        }
        if (RAND_bytes(buf, len) <= 0) {
                return NULL;
        }
        // Find the first non-zero byte and creating ASN1_INTEGER object.
        int i = 0;
        for (i = 0; i < len && !buf[i]; ++i) ;
        ASN1_INTEGER* nonce = NULL;
        if (!(nonce = ASN1_INTEGER_new())) {
                ASN1_INTEGER_free(nonce);
                return NULL;
        }
        OPENSSL_free(nonce->data);
        // Allocate at least one byte.
        nonce->length = len - i;
        if (!(nonce->data = reinterpret_cast<unsigned char*>(OPENSSL_malloc(nonce->length +
1)))) {
```

```
                ASN1_INTEGER_free(nonce);
                return NULL;
        }
        memcpy(nonce->data, buf + i, nonce->length);
        return nonce;
}

bool SignatureCallbackImpl::ParseP12File(const wchar_t* cert_file_path, String
cert_file_password,
        EVP_PKEY** pkey, X509** x509, STACK_OF(X509)** ca) {
                FILE* file = NULL;
#if defined(_WIN32) || defined(_WIN64)
                _wfopen_s(&file, cert_file_path, L"rb");
#else
                file = fopen(String::FromUnicode(cert_file_path), "rb");
#endif  // defined(_WIN32) || defined(_WIN64)
                if (!file) {
                        return false;
                }

                PKCS12* pkcs12 = d2i_PKCS12_fp(file, NULL);
                fclose (file);
                if (!pkcs12) {
                        return false;
                }

                if (!PKCS12_parse(pkcs12, (const char*)cert_file_password, pkey, x509, ca)) {
                        return false;
                }

                PKCS12_free(pkcs12);
                if (!pkey)
                        return false;
                return true;
}

unsigned char* SignatureCallbackImpl::PKCS7Sign(const wchar_t* cert_file_path, String
cert_file_password,
        String plain_text, int& signed_data_size) {
                PKCS7* p7 = NULL;
                EVP_PKEY* pkey = NULL;
                X509* x509 = NULL;
                STACK_OF(X509)* ca = NULL;
                if(!ParseP12File(cert_file_path, cert_file_password, &pkey, &x509, &ca))
                        return NULL;

                p7 = PKCS7_new();
                PKCS7_set_type(p7, NID_pkcs7_signed);
                PKCS7_content_new(p7, NID_pkcs7_data);

                // Application should not judge the sign algorithm with the content's length.
                // Here, just for convenient;
                if (plain_text.GetLength() > 32)
                        PKCS7_ctrl(p7, PKCS7_OP_SET_DETACHED_SIGNATURE, 1, NULL);

                PKCS7_SIGNER_INFO* signer_info = PKCS7_add_signature(p7, x509, pkey,
EVP_sha1());
                PKCS7_add_certificate(p7, x509);

                for (int i = 0; i< sk_num(CHECKED_STACK_OF(X509,ca)); i++)
                        PKCS7_add_certificate(p7, (X509*)sk_value(CHECKED_STACK_OF(X509,ca), i));
```

```
                // Set source data to BIO.
                BIO* p7bio = PKCS7_dataInit(p7, NULL);
                BIO_write(p7bio, plain_text.GetBuffer(1), plain_text.GetLength());
                PKCS7_dataFinal(p7, p7bio);

                FREE_CERT_KEY;
                BIO_free_all(p7bio);
                // Get signed data.
                unsigned long der_length = i2d_PKCS7(p7, NULL);
                unsigned char* der = reinterpret_cast<unsigned char*>(malloc(der_length));
                memset(der, 0, der_length);
                unsigned char* der_temp = der;
                i2d_PKCS7(p7, &der_temp);
                PKCS7_free(p7);
                signed_data_size = der_length;
                return (unsigned char*)der;
}

bool SignatureCallbackImpl::PKCS7VerifySignature(String signed_data, String plain_text) {
        // Retain PKCS7 object from signed data.
        BIO* vin = BIO_new_mem_buf((void*)signed_data.GetBuffer(1), signed_data.GetLength());
        PKCS7* p7 = d2i_PKCS7_bio(vin, NULL);
        STACK_OF(PKCS7_SIGNER_INFO) *sk = PKCS7_get_signer_info(p7);
        int sign_count = sk_PKCS7_SIGNER_INFO_num(sk);

        int length = 0;
        bool bSigAppr = false;
        unsigned char *p = NULL;
        for(int i=0;i<sign_count; i++) {
                PKCS7_SIGNER_INFO* sign_info = sk_PKCS7_SIGNER_INFO_value(sk,i);

                BIO *p7bio = BIO_new_mem_buf((void*)plain_text.GetBuffer(1),
plain_text.GetLength());
                X509 *x509= PKCS7_cert_from_signer_info(p7,sign_info);
                if(1 == PKCS7_verify(p7, NULL, NULL,p7bio, NULL, PKCS7_NOVERIFY))
                        bSigAppr = true;
                BIO_free(p7bio);
        }
        PKCS7_free(p7);
        BIO_free(vin);
        return bSigAppr;
}
```

3.28 PDF Action

PDF Action is represented as the base PDF action class. Foxit PDF SDK provides APIs to create a series of actions and get the action handlers, such as embedded goto action, JavaScript action, named action and launch action, etc.

Example:

3.28.1 How to create a URI action and insert to a link annot

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
```

```
#include "include/pdf/objects/fs_pdfobject.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace annots;

// Assuming PDFPage page has been loaded and parsed.
// Assuming the annnots in the page have been loaded.
...

// Add link annotation
annots::Link link(page.AddAnnot(Annot::e_Link, RectF(350,350,380,400)));
link.SetHighlightingMode(Annot::e_HighlightingToggle);
// Add action for link annotation
using foxit::pdf::actions::Action;
using foxit::pdf::actions::URIAction;
URIAction action = (URIAction)Action::Create(page.GetDocument(), Action::e_TypeURI);
action.SetTrackPositionFlag(true);
action.SetURI("www.foxitsoftware.com");
link.SetAction(action);
// Appearance should be reset.
link.ResetAppearanceStream();
```

3.28.2 How to create a GoTo action and insert to a link annot

```
#include "include/common/fs_common.h"
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/objects/fs_pdfobject.h"
#include "include/pdf/fs_pdfpage.h"

using namespace foxit;
using namespace foxit::common;
using namespace pdf;
using namespace annots;

// Assuming the PDFDoc doc has been loaded.
// Assuming PDFPage page has been loaded and parsed.

// Add link annotation
annots::Link link(page.AddAnnot(Annot::e_Link, RectF(350,350,380,400)));
link.SetHighlightingMode(Annot::e_HighlightingToggle);

GotoAction action = Action::Create(page.GetDocument(), Action::e_TypeGoto);
Destination newDest = Destination::CreateXYZ(page.GetDocument(), 0,0,0,0);
action.SetDestination(newDest);
```

3.29 JavaScript

JavaScript was created to offload Web page processing from a server onto a client in Web-based applications. Foxit PDF SDK JavaScript implements extensions, in the form of new objects and their accompanying methods and properties, to the JavaScript language. It enables a developer to manage document security, communicate with a database, handle file attachments, and manipulate a PDF file so that it behaves as an interactive, web-enabled form, and so on.

JavaScript action is an action that causes a script to be compiled and executed by the JavaScript interpreter. Class com.foxit.sdk.pdf.actions.JavaScriptAction is derived from Action and offers functions to get/set JavaScript action data.

Example:

3.29.1 How to add JavaScript Action to Document

```
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"

using namespace foxit;
using namespace pdf;
using namespace annots;

// Load Document doc.
...

actions::JavaScriptAction javascript_action =
(actions::JavaScriptAction)Action::Create(form.GetDocument(), Action::e_TypeJavaScript);
javascipt_action.SetScript(L"app.alert(\"Hello Foxit \");");
AdditionalAction additional_act(doc);
additional_act.SetAction(AdditionalAction::e_TriggerDocWillClose,javascipt_action);
...
```

3.29.2 How to add JavaScript Action to Annotation

```
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"

using namespace foxit;
using namespace pdf;
using namespace annots;

// Load Document and get a widget annotation.
...

actions::JavaScriptAction javascript_action =
(actions::JavaScriptAction)Action::Create(form.GetDocument(), Action::e_TypeJavaScript);
javascipt_action.SetScript(L"app.alert(\"Hello Foxit \");");
AdditionalAction additional_act(annot);
additional_act.SetAction(AdditionalAction::e_TriggerAnnotMouseButtonPressed,javascipt_action);
...
```

3.29.3 How to add JavaScript Action to FormField

```
#include "include/pdf/actions/fs_action.h"
#include "include/pdf/annots/fs_annot.h"

using namespace foxit;
using namespace pdf;
using namespace annots;

// Load Document and get a form field.
...

actions::JavaScriptAction javascript_action =
(actions::JavaScriptAction)Action::Create(form.GetDocument(), Action::e_TypeJavaScript);
javascipt_action.SetScript(L"AFSimple_Calculate(\"SUM\", new Array (\"Text Field0\", \"Text
Field1\"));" );
AdditionalAction additional_act(field);
additional_act.SetAction(AdditionalAction::e_TriggerFieldRecalculateValue,javascipt_action);
...
```

3.30 Redaction

Redaction is the process of removing sensitive information while keeping the document's layout. It allows users to permanently remove (redact) visible text and images from PDF documents to protect confidential information, such as social security numbers, credit card information, product release dates, and so on.

Redaction is a type of markup annotation, which is used to mark some contents of a PDF file and then the contents will be removed once the redact annotations are applied.

To do Redaction, you can use the following APIs:

- Call function foxit::addon::Redaction::Redaction to create a redaction module. If module "Redaction" is not defined in the license information which is used in function common::Library::Initialize , that means user has no right in using redaction related functions and this constructor will throw exception foxit::e_ErrInvalidLicense.
- Then call function foxit::addon::Redaction::MarkRedactAnnot to create a redaction object and mark page contents (text object, image object, and path object) which are to be redacted.
- Finally call function foxit::addon::Redaction::Apply to apply redaction in marked areas: remove the text or graphics under marked areas permanently.

Example:

3.30.1 How to redact the text "PDF" on the first page of a PDF?

```
#include "include/common/fs_common.h"
#include "include/pdf/fs_pdfdoc.h"
#include "include/pdf/fs_pdfpage.h"
```

```
#include "include/pdf/fs_search.h"
#include "include/addon/fs_redaction.h"
#include "include/common/fs_render.h"

using namespace foxit;
using namespace common;
using namespace addon;
using namespace pdf;
using namespace foxit::pdf::annots;
...

Redaction redaction(doc);
// Parse PDF page.
PDFPage page = doc.GetPage(0);
page.StartParse(foxit::pdf::PDFPage::e_ParsePageNormal, NULL, false);
//Find Text Object to redact
TextPage text_page(page);
TextSearch text_search(text_page);
text_search.SetPattern(L"PDF");
RectFArray rect_array;
while(text_search.FindNext()) {
	rect_array.Append(text_search.GetMatchRects());
  }
if(rect_array.GetSize() > 0) {
	Redact redact = redaction.MarkRedactAnnot(page, rect_array);
	redact.ResetAppearanceStream();
	doc.SaveAs(output_directory + L"AboutFoxit_redected_default.pdf");

	// set border color to Green
	redact.SetBorderColor((long)0x00FF00);
	// set fill color to Blue
	redact.SetFillColor((long)0x0000FF);
	// set rollover fill color to Red
	redact.SetApplyFillColor((long)0xFF0000);
	redact.ResetAppearanceStream();
	doc.SaveAs(output_directory + L"AboutFoxit_redected_setColor.pdf");

	redact.SetOpacity((float)0.5);
	redact.ResetAppearanceStream();
	doc.SaveAs(output_directory + L"AboutFoxit_redected_setOpacity.pdf");

	if(redaction.Apply())
		cout << "Redact page(0) succeed." << endl;
	else
		cout << "Redact page(0) failed." << endl;
  }
doc.SaveAs(output_directory + L"AboutFoxit_redected_apply.pdf");
```

REFERENCES

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

[2] PDF reference 2.0
https://www.iso.org/standard/63534.html

[3] Foxit PDF SDK API reference
sdk_folder/doc/Foxit PDF SDK API Reference.html

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