Quick Start Guide

Foxit PDF SDK

For Objective-C

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 Mac Objective-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 Mac platform with Objective-C language.

Foxit PDF SDK for Mac (Objective-C API) ships with simple-to-use APIs that can help Objective-C developers seamlessly integrate powerful PDF technology into their own projects on 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, etc.

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 10-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 Objective-C program language into their own applications on Mac platform. 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. 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

Mac OS X 10.6 to 10.12

2.2 What is in this package

Download Foxit PDF SDK zip for Mac package and extract it to a new directory "foxitpdfsdk_6_4_mac_oc". The structure of the release package is shown in Figure 2-1.

***NOTE**: the highlighted rectangle in the figure is just the version of Foxit PDF SDK. Here the SDK version is 6.4, so it shows 6_4 in the package name.*

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

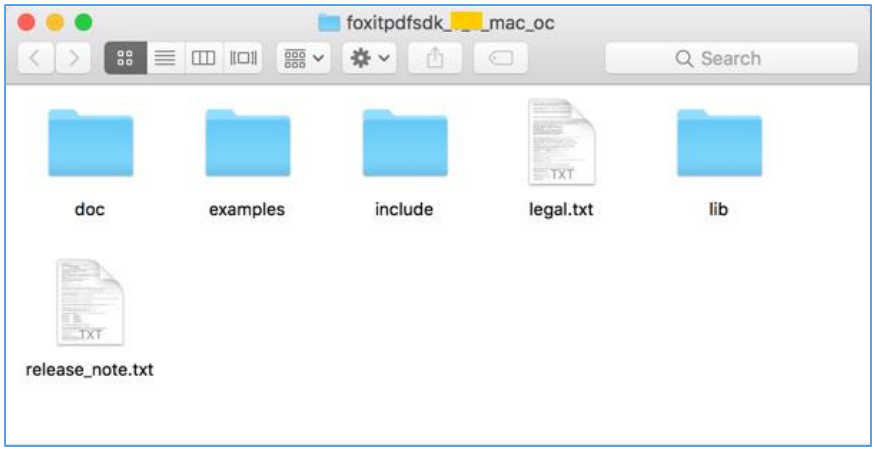


Figure 2-1

2.3 How to run a demo

Foxit PDF SDK for Mac (Objective-C) provides several simple demos in directory "\examples". All these demos can be run directly with the ".sh" files in directory "\examples\simple_demo".

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

1) Open a terminal window, navigate to "foxitpdfsdk_6_4_mac_oc\examples\simple_demo";
2) Run a demo by the ".sh" file. Choose one of the following:
   - Run "./RunAllDemo.sh" to run all of the demos one by one.
   - If you want to run a specific single demo, please locate to the directory of the demo, for example locate to "\examples\simple_demo\annotation", and run "./RunDemo.sh".

"\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\".

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.*

Compliance demos

For Compliance demos, you should build a resource directory at first, please contact Foxit support team or sales team to get the resource files package. For more detailed about how to run the demo, please refer to Section 3.31 "Compliance (PDF/A)".

2.4 How to create a simple project

In this section, we will show you how to use Foxit PDF SDK for Mac (Objective-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_oc*".

2) Copy "include" and "lib" folders from "foxitpdfsdk_6_4_mac_oc" folder to the project "*test_oc*" folder.

3) Create a class file named "test_oc.mm" under "*test_oc*" folder.

4) Open the "test_oc.mm" file, add the following "include" header statement to the beginning of test_oc.mm.

```
#include "FSPDFObjC.h"
```

5) 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.

```
int main(int argc, const 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=").
    NSString* sn = @" ";
    NSString* key = @" ";

    // Initialize library.
    FSErrorCode code = [FSLibrary initialize:sn key:key];
    if (code != FSErrSuccess)
        return -1;
}
```

6) Load a PDF document, and parse the first page of the document. Assume that you have already put a "Sample.pdf" to the "*test_oc*" folder.

```
// Load a PDF document, and parse the first page of the document.
NSString* pdfpath = [[NSBundle mainBundle] pathForResource:@"Sample"
ofType:@"pdf"];
FSPDFDoc* doc = [[FSPDFDoc alloc] initWithPath:pdfpath];
FSErrorCode errorCode = [doc load:nil];
if (errorCode != FSErrSuccess) {
    return -1;
}
```

```
FSPDFPage* page = [doc getPage:0];
[page startParse:FSPDFPageParsePageNormal pause:nil is_reparse:NO];
```

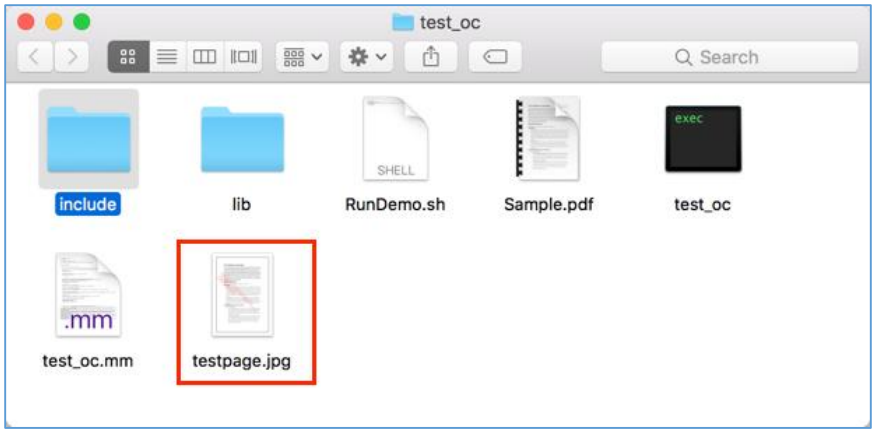
7) Render the first page to a bitmap and save it as a JPG file.

```
int width = (int)[page getWidth];
int height = (int)[page getHeight];
FSMatrix2D* matrix = [page getDisplayMatrix:0 top:0 width:width height:height
rotate:page.rotation];

// Prepare a bitmap for rendering.
FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:width height:height
format:FSBitmapDIBArgb buffer:nil pitch:0];
 [bitmap fillRect:0xFFFFFFFF rect:nil];

// Render page.
FSRenderer* render = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:NO];
[render startRender:page matrix:matrix pause:nil];

// Add the bitmap to image and save the image.
FSImage* image = [FSImage new];
[image addFrame:bitmap];
[image saveAs:@"testpage.jpg"];
```

8) Create a shell file named "RunTest.sh" to include the libfsdk_oc_mac64.dylib. A sample shell is as follows:

```
#!/bin/bash
export TEST_NAME=test_oc
clang -framework Foundation -I include -L lib -lfsdk_oc_mac64 -Xlinker -rpath -
Xlinker lib ${TEST_NAME}.mm -o ${TEST_NAME}
./${TEST_NAME}
```

9) Run the "RunTest.sh". Open a terminal window, navigate to "*test_oc*", and run "./RunTest.sh". Then the "testpage.jpg" will be generated in "*test_oc*" folder (See as below).



The final contents of "test_oc.mm" is as follow:

```
#include "FSPDFObjC.h"

int main(int argc, const 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=").
    NSString* sn = @" ";
    NSString* key = @" ";

    // Initialize library.
    FSErrorCode code = [FSLibrary initialize:sn key:key];
    if (code != FSErrSuccess) {
        return -1;
    }

    // Load a PDF document, and parse the first page of the document.
    NSString* pdfpath = [[NSBundle mainBundle] pathForResource:@"Sample"
ofType:@"pdf"];
    FSPDFDoc* doc = [[FSPDFDoc alloc] initWithPath:pdfpath];
    FSErrorCode errorCode = [doc load:nil];
    if (errorCode != FSErrSuccess) {
        return -1;
    }
    FSPDFPage* page = [doc getPage:0];
    [page startParse:FSPDFPageParsePageNormal pause:nil is_reparse:NO];

    int width = (int)[page getWidth];
    int height = (int)[page getHeight];
    FSMatrix2D* matrix = [page getDisplayMatrix:0 top:0 width:width height:height
rotate:page.rotation];

    // Prepare a bitmap for rendering.
```

```
    FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:width height:height
format:FSBitmapDIBArgb buffer:nil pitch:0];
    [bitmap fillRect:0xFFFFFFFF rect:nil];

    // Render page.
    FSRenderer* render = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:NO];
    [render startRender:page matrix:matrix pause:nil];

    // Add the bitmap to image and save the image.
    FSImage* image = [FSImage new];
    [image addFrame:bitmap];
    [image saveAs:@"testpage.jpg"];
}
```

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 Objective-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 and unlock Foxit PDF SDK license before calling any APIs. The function FSLibrary::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 FSLibrary::destroy 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 "FSPDFObjC.h"
...

NSString* sn = @" ";
NSString* key = @" ";

// Initialize library.
FSErrorCode code = [FSLibrary initialize:sn key:key];
if (code != FSErrSuccess) {
      return -1;
}
```

3.2 Document

A PDF document object can be constructed with an existing PDF file from file path, memory buffer, a custom implemented FSFileReaderCallback object and an input file stream. Then call function FSPDFDoc::load or FSPDFDoc::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 "FSPDFObjC.h"
...

FSPDFDoc* doc = [[FSPDFDoc alloc] init];
```

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

3.2.2 How to load an existing PDF document from file path

```
#include "FSPDFObjC.h"
...

NSString* pdfpath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
FSPDFDoc* doc = [[FSPDFDoc alloc] initWithPath:pdfpath];
FSErrorCode errorCode = [doc load:nil];
if (errorCode != FSErrSuccess) {
    return -1;
}
```

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

```
#include "FSPDFObjC.h"
...
NSData* file_data = [NSData dataWithContentsOfFile:pdf_path];

FSPDFDoc* doc = [[FSPDFDoc alloc] initWithBuffer:file_data];
FSErrorCode errorCode = [doc load:nil];
if (errorCode != FSErrSuccess) {
    return -1;
}
```

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

```
#include "FSPDFObjC.h"
...

@interface FSFileRead : NSObject<FSFileReaderCallback>

- (id)initWithSourceFilePath:(NSString *)path offset:(long long)offset;

@end

@implementation FSFileRead {
    FileReader *imp;
}

- (id)initWithSourceFilePath:(NSString *)path offset:(long long)offset {
    if (self = [super init]) {
        imp = new FileReader(offset);
        if (!imp->LoadFile([path UTF8String])) {
            return nil;
        }
    }
    return self;
}
```

```
- (void)dealloc {
    delete imp;
}

- (unsigned long long)getSize {
    return imp->GetSize();
}

- (NSData *)readBlock:(unsigned long long)offset size:(unsigned long long)size {
    void *buffer = malloc(size);
    if (!buffer) {
        NSLog(@"failed to malloc buffer of size %llu", size);
        return nil;
    }
    if (imp->ReadBlock(buffer, offset, (size_t)size)) {
        return [NSData dataWithBytesNoCopy:buffer length:size];
    } else {
        free(buffer);
        return nil;
    }
}

@end

FSFileRead *file_reader = [[FSFileRead alloc] initWithSourceFilePath:file_name
offset:offset];

FSPDFDoc *doc_real = [[FSPDFDoc alloc] initWithFile_read:file_reader is_async:NO];
FSErrorCode code = [doc_real load:nil];
if (code != FSErrSuccess) {
    return -1;
}
```

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

```
#include "FSPDFObjC.h"
...

NSString* pdfpath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
FSPDFDoc* doc = [[FSPDFDoc alloc] initWithPath:pdfpath];
FSErrorCode errorCode = [doc load:nil];
if (errorCode != FSErrSuccess) {
    return -1;
}

// Get the first page of the document.
FSPDFPage* page = [doc getPage:0];
// Parse page.
[page startParse:FSPDFPageParsePageNormal pause:nil is_reparse:NO];
```

3.2.6 How to save a PDF to a file

```
#include "FSPDFObjC.h"
...

NSString* pdfpath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
```

```
FSPDFDoc* doc = [[FSPDFDoc alloc] initWithPath:pdfpath];
FSErrorCode errorCode = [doc load:nil];
if (errorCode != FSErrSuccess) {
    return -1;
}

[doc saveAs:output_file save_flags:FSPDFDocSaveFlagNoOriginal];
```

3.2.7 How to save a document into memory buffer by FileWriterCallback?

```
#include "FSPDFObjC.h"
...

@interface FSFileWriterCallbackImpl : NSObject<FSFileWriterCallback>
@property (nonatomic) NSMutableData* mutableData;
@end

@implementation FSFileWriterCallbackImpl
- (instancetype)init
    {
            self = [super init];
            if (self) {
                    _mutableData = [[NSMutableData alloc] init];
                }
            return self;
        }
-(unsigned long long)getSize {
        if (!self.mutableData) return NO;

        return self.mutableData.length;
}

-(BOOL)writeBlock:(NSData*)data offset:(unsigned long long)offset {
        if (!self.mutableData) return NO;

        [self.mutableData appendData:data];
        return YES;
}

-(BOOL)flush {
        return YES;
}
@end

...
FSFileWriterCallbackImpl* filewriter = [[FSFileWriterCallbackImpl alloc] init];

// Assuming FSPDFDoc doc has been loaded.
...

[doc startSaveAsWithWriterCallback:filewriter save_flags:FSPDFDocSaveFlagNoOriginal
pause:nil];
...
```

3.3 Page

PDF Page is the basic and important component of PDF Document. A FSPDFPage object is retrieved from a PDF document by function FSPDFDoc::getPage. Page level APIs provide functions to parse, render, edit (includes creating, deleting and flattening) 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 "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.
...
float width = [page getWidth];
float height = [page getHeight];
...
```

3.3.2 How to calculate bounding box of page contents

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.
...
FSRectF* content_box = [page calcContentBBox:FSPDFPageCalcContentsBox];
```

3.3.3 How to create a PDF page and set the size

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

float w = 612.0;
float h = 792.0;
FSPDFPage* page = [doc insertPage:index width:w height:h];
```

3.3.4 How to delete a PDF page

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.

// Remove a PDF page by page index.
[doc removePage:index];

// Remove a specified PDF page.
```

```
[doc removePageWithPDFPage:page];
...
```

3.3.5 How to flatten a PDF page

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.

// Flatten all contents of a PDF page.
[page flatten:YES options:FSPDFPageFlattenAll];

// Flatten a PDF page without annotations.
[page flatten:YES options:FSPDFPageFlattenNoAnnot];

// Flatten a PDF page without form controls.
[page flatten:YES options:FSPDFPageFlattenNoFormControl];

// Flatten a PDF page without annotations and form controls (Equals to nothing to be
flattened).
[page flatten:YES options:FSPDFPageFlattenNoAnnot|FSPDFPageFlattenNoFormControl];
...
```

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.
FSBitmap* thumbnail_bmp = [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 FSRenderer::setRenderContentFlags to decide whether to render page and annotation both or not, and then use function FSRenderer::startRender to do the rendering. Function FSRenderer::startQuickRender can also be used to render page but only for thumbnail purpose.
- To render a single annotation, use function FSRenderer::renderAnnot.
- To render on a bitmap, use function FSRenderer::startRenderBitmap.
- To render a reflowed page, use function FSRenderer::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 FSFiller object to fill the form, the function FSFiller::render should be used to render the focused form control instead of the function FSRenderer::renderAnnot.

Example:

3.4.1 How to render a page to a bitmap

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.

int width = (int)[page getWidth];
int height = (int)[page getHeight];
FSMatrix2D* matrix = [page getDisplayMatrix:0 top:0 width:width height:height
rotate:page.rotation];

// Prepare a bitmap for rendering.
FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:width height:height
format:FSBitmapDIBArgb buffer:nil pitch:0];
[bitmap fillRect:0xFFFFFFFF rect:nil];

// Render page.
FSRenderer* render = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:NO];
[render startRender:page matrix:matrix pause:nil];
...
```

3.4.2 How to render page and annotation

```
#include "FSPDFObjC.h"
...

// Assuming PDFPage page has been loaded and parsed.

int width = (int)[page getWidth];
int height = (int)[page getHeight];
FSMatrix2D* matrix = [page getDisplayMatrix:0 top:0 width:width height:height
rotate:page.rotation];

// Prepare a bitmap for rendering.
FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:width height:height
format:FSBitmapDIBArgb buffer:nil pitch:0];
[bitmap fillRect:0xFFFFFFFF rect:nil];
```

```
// Render page.
FSRenderer* render = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:NO];
unsigned int render_flag = FSRendererRenderPage | FSRendererRenderAnnot;
[render setRenderContentFlags:render_flag];
[render startRender:page matrix:matrix pause:nil];
...
```

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 "FSPDFObjC.h"
...

FSAttachments *attachments = [[FSAttachments alloc] initWithDoc:doc
nametree:[[FSPDFNameTree alloc] init]];
int count = [attachments getCount];
for (int i = 0; i < count; i++) {
    NSString *key = [attachments getKey:i];

    FSFileSpec *file_spec = [attachments getEmbeddedFile:key];
    if (![file_spec isEmpty]) {
        NSString *name = [file_spec getFileName];

        if ([file_spec isEmbedded]) {
            NSString *exFilePath = [[NSString alloc] initWithFormat:@"%@%@",
output_directory, name];
            bool bExportStatus = [file_spec exportToFile:exFilePath];
        }
    }
}
```

3.5.2 How to remove all the attachments of a PDF

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.

FSAttachments *attachments = [[FSAttachments alloc] initWithDoc:doc
nametree:[[FSPDFNameTree alloc] init]];
[attachments removeAllEmbeddedFiles];
...
```

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 FSTextPage objects which are related to a specific page. FSTextPage 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 FSTextSearch object with FSTextPage object.
- To access text such like hypertext link, construct a FSPageTextLinks object with FSTextPage object.

Example:

3.6.1 How to extract text from a PDF page

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.


// Get the text page object.
FSTextPage *textPage = [[FSTextPage alloc] initWithPage:page
flags:FSTextPageParseTextNormal];
int charCount = [textPage getCharCount];
if (charCount > 0) {
    NSString *currentText = [textPage getChars:0 count:-1];
}
...
```

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

```
#include "FSPDFObjC.h"
...

FSTextPage *textPage = [[FSTextPage alloc] initWithPage:page
flags:FSTextPageParseTextNormal];
FSRectF* rect = [[FSRectF alloc] initWithLeft1:90 bottom1:580 right1:450 top1:595];
NSString* text = [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 FSTextSearch::setPattern, FSTextSearch::setStartPage (only useful for a text search in PDF document), FSTextSearch::setEndPage (only useful for a text search in PDF document) and FSTextSearch::setSearchFlags.
- To do the searching, use function FSTextSearch::findNext or FSTextSearch::findPrev.
- To get the searching result, use function FSTextSearch::getMatchXXX().

Example:

3.7.1 How to search a text pattern in a PDF

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSTextSearch *search = [[FSTextSearch alloc] initWithDocument:doc cancel:nil];
int startIndex = 0;
int endIndex = [doc getPageCount] - 1;
[search setStartPage:startIndex];
[search setEndPage:endIndex];
NSString *pattern = @"Foxit";
[search setPattern:pattern];
NSInteger flags = FSTextSearchSearchNormal;
[search setSearchFlags:(unsigned int)flags];

int match_count = 0;
while ([search findNext]) {
      FSRectFArray *rects = [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 FSPageTextLinks::getTextLink to get a textlink object.

Example:

3.8.1 How to retrieve hyperlinks in a PDF page

```
#include "FSPDFObjC.h"
...
FSTextPage *textPage = [[FSTextPage alloc] initWithPage:page
flags:FSTextPageParseTextNormal];
FSPageTextLinks* page_text_links = [[FSPageTextLinks alloc] initWithPage:textPage];
if (NO == [page_text_links isEmpty]) {
    int index = 0;
    FSTextLink* text_link = [page_text_links getTextLink:index];
```

```
    if (NO == [text_link isEmpty])
    NSString* url = [text_link 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 FSPDFDoc::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 FSBookmark::getFirstChild.

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

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

Example:

3.9.1 How to find and list all bookmarks of a PDF

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSBookmark *root = [doc getRootBookmark];
if (![root isEmpty]) {
    ShowBookmarkInfo(root, info, 0);
}

void ShowBookmarkInfo(FSBookmark *bookmark, int depth) {
    if (depth > 32)
        return;
    if ([bookmark isEmpty]) {
        return;
    }
    ShowBookmarkInfo([bookmark getFirstChild], depth + 1);
    ShowBookmarkInfo([bookmark getNextSibling], depth);
}
```

```
...
```

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

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

Example:

3.10.1 How to load the forms in a PDF

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.

BOOL hasform = [doc hasForm];
if(hasform)
  FSForm *form = [[FSForm alloc] initWithDocument:doc];
...
```

3.10.2 How to count form fields and get the properties

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
BOOL hasform = [doc hasForm];
if(hasform) {
    FSForm *form = [[FSForm alloc] initWithDocument:doc];
    int count = [form getFieldCount:@""];
    for (int i = 0; i < count; i++) {
        FSField* field = [form getField:i filter:@""];
```

```
        FSFieldType field_type = [field getType];
        NSString* name = [field getName];
    }
}
```

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSForm *form = [[FSForm alloc] initWithDocument:doc];
BOOL is_success = [field exportToXML:@"test.xml"];
...
```

3.10.4 How to import form data from a XML file

```
#include "FSPDFObjC.h"
...
FSForm *form = [[FSForm alloc] initWithDocument:doc];
BOOL is_success = [field importFromXML:@"test.xml"];
...
```

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 FSAppProviderCallback and FSDocProviderCallback 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 "FSPDFObjC.h"
...

// implement from FSAppProviderCallback.
CFS_XFAAppHandler* pXFAAppHandler = [CFS_XFAAppHandler alloc];
[FSLibrary registerXFAAppProviderCallback:pXFAAppHandler];
FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:input_file];
FSErrorCode errorCode = [doc load:@""];
```

```
if (errorCode != FSErrSuccess) {
    return -1;
}

// implement from FSDocProviderCallback.
CFS_XFADocHandler* pXFADocHandler = [CFS_XFADocHandler alloc];
FSXFADoc* xfa_doc = [[FSXFADoc alloc] initWithDocument:doc
xfa_doc_provider_handler:pXFADocHandler];
[xfa_doc startLoad:nil];
...
```

3.11.2 How to export and import XFA form data

```
#include "FSPDFObjC.h"
...

// Assuming FSXFADoc xfa_doc has been loaded.

[xfa_doc exportData:@"xfa_form.xml" export_type:FSXFADocExportDataTypeXML];
[xfa_doc resetForm];
[doc saveAs:@"xfa_dynamic_resetform.pdf" save_flags:FSPDFDocSaveFlagNormal];

[xfa_doc importData:@"xfa_form.xml"];
[doc saveAs:@"xfa_dynamic_importdata.pdf" save_flags:FSPDFDocSaveFlagNormal];
...
```

3.12 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:

3.12.1 How to add a text form field to a PDF

```
#include "FSPDFObjC.h"
...
// Add test field
FSControl *control = [form addControl:page field_name:@"Text Field0"
field_type:FSFieldTypeTextField rect:[[FSRectF alloc] initWithLeft1:50.0 bottom1:600
right1:90 top1:640]];
[control getField].value = @"3";

// Update text field's appearance.
[[control getWidget] resetAppearanceStream];
...
```

3.12.2 How to remove a text form field from a PDF

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
```

```
...

FSForm *form = [[FSForm alloc] initWithDocument:doc];
int count = [form getFieldCount:@""];
for (int i = 0; i < count; i++) {
    FSField* field = [form getField:i filter:@""];
    FSFieldType field_type = [field getType];
    if (FSFieldTypeTextField == field_type)
        [field removeField:field];
}
...
```

3.13 Annotations

3.13.1 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 (TypeWriter/TextBox/Callout)	Free text annotation	Yes	Yes
Line	Line annotation	Yes	Yes
Square	Square annotation	Yes	Yes
Circle	Circle annotation	Yes	Yes
Polygon	Polygon annotation	Yes	Yes
PolyLine	PolyLine annotation	Yes	Yes
Highlight	Highlight annotation	Yes	Yes
Underline	Underline annotation	Yes	Yes
Squiggly	Squiggly annotation	Yes	Yes
StrikeOut	StrikeOut annotation	Yes	Yes
Stamp	Stamp annotation	Yes	Yes
Caret	Caret annotation	Yes	Yes

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:

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.

// Add link annotation.
FSRectF* annot_rect = [[FSRectF alloc] initWithLeft1:350 bottom1:350 right1:380
top1:400];
FSLink* link = [[FSLink alloc] initWithAnnot:[page addAnnot:FSAnnotLink
rect:annot_rect]];
[link setHighlightingMode:FSAnnotHighlightingToggle];
[link resetAppearanceStream];
...
```

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.
```

```
// Add highlight annotation.
FSRectF* annot_rect = [[FSRectF alloc] initWithLeft1:10 bottom1:450 right1:100
top1:550];
FSHighlight* highlight = [[FSHighlight alloc] initWithAnnot:[page
addAnnot:FSAnnotHighlight rect:annot_rect]];
[highlight setContent:@"Highlight"];
FSQuadPoints* quad_points = [FSQuadPoints new];
FSPointF* point = [FSPointF new];
[point set:10 y:500];
quad_points.first = point;
[point set:90 y:500];
quad_points.second = point;
[point set:10 y:480];
quad_points.third = point;
[point set:90 y:480];
quad_points.fourth = point;
FSQuadPointsArray* quad_points_array = [FSQuadPointsArray new];
[quad_points_array add:quad_points];
[highlight setQuadPoints:quad_points_array];
[highlight setSubject:@"Highlight"];
[highlight setTitle:@"Foxit SDK"];
FSDateTime* local_time = getLocalDateTime();
[highlight setCreationDateTime:local_time];
[highlight setModifiedDateTime:local_time];
[highlight setUniqueID:randomUID()];

// Appearance should be reset.
[highlight resetAppearanceStream];
...
```

3.13.1.3 How to set the popup information when creating markup annotations

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.

// Add note annotation
FSRectF* annot_rect = [[FSRectF alloc] initWithLeft1:10 bottom1:350 right1:50
top1:400];
FSNote* note = [[FSNote alloc] initWithAnnot:[page addAnnot:FSAnnotNote
rect:annot_rect]];
[note setIconName:@"Comment"];
[note setSubject:@"Note"];
[note setTitle:@"Foxit SDK"];
[note setContent:@"Note annotation."];
FSDateTime* local_time = getLocalDateTime();
[note setCreationDateTime:local_time];
[note setModifiedDateTime:local_time];
[note setUniqueID:randomUID()];

// Add popup to note annotation
annot_rect = [[FSRectF alloc] initWithLeft1:300 bottom1:450 right1:500 top1:550];
FSPopup* popup = [[FSPopup alloc] initWithAnnot:[page addAnnot:FSAnnotPopup
rect:annot_rect]];
[popup setBorderColor:0x00FF00];
[popup setOpenStatus:NO];
local_time = getLocalDateTime();
[popup setModifiedDateTime:local_time];
```

```
[note setPopup:popup];

[note resetAppearanceStream];
...
```

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

```
#include "FSPDFObjC.h"
...

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

int width = (int)[page getWidth];
int height = (int)[page getHeight];
FSMatrix2D* display_matrix = [page getDisplayMatrix:0 top:0 width:width height:height
rotate:page.rotation];
int annot_count = [page getAnnotCount];
for (int i=0; i<annot_count; i++) {
    FSAnnot* annot = [page getAnnot:i];
    FSAnnotType annot_type = [annot getType];
    if (FSAnnotPopup == annot_type) continue;
    FSRectI* device_rect = [annot getDeviceRect:NO matrix:display_matrix];

    // Get the same annot by using device point.
    float tolerance = 1.0;
    FSPointF* point = [FSPointF alloc];
    [point set:device_rect.left+tolerance y:(device_rect.top -
device_rect.bottom)/2+device_rect.bottom];
    FSAnnot* get_annot = [page getAnnotAtDevicePoint:point tolerance:tolerance
matrix:display_matrix];
}
```

3.13.2 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:

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSFDFDoc* fdf_doc = [[FSFDFDoc alloc] initWithPath: @"AnnotationData.fdf"];
[doc importFromFDF:fdf_doc types:FSPDFDocAnnots page_range:[FSRange new]];
...
```

3.14 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:

3.14.1 How to convert PDF pages to bitmap files.

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSImage* image = [FSImage new];

// Get page count
int page_count = [doc getPageCount];
for(int i=0;i<page_count;i++) {
    FSPDFPage* page = [doc getPage:i];
    // Parse page.
    [page startParse:FSPDFPageParsePageNormal pause:nil is_reparse:NO];

    int width = (int)[page getWidth];
    int height = (int)[page getHeight];
    FSMatrix2D* matrix = [page getDisplayMatrix:0 top:0 width:width height:height
rotate:page.rotation];

    // Prepare a bitmap for rendering.
    FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:width height:height
format:FSBitmapDIBArgb buffer:nil pitch:0];
    [bitmap fillRect:0xFFFFFFFF rect:nil];

    // Render page
    FSRenderer* render = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:NO];
    [render startRender:page matrix:matrix pause:nil];
    [image addFrame:bitmap];
}
```

3.14.2 How to convert an image file to PDF file

```
#include "FSPDFObjC.h"
...

FSImage *image = [[FSImage alloc] initWithPath:input_file];
int count = [image getFrameCount];

FSPDFDoc *doc = [[FSPDFDoc alloc] init];
for (int i = 0; i < count; i ++) {
    FSBitmap *bitmap = [image getFrameBitmap:i];
    float w = 612.0;
    float h = 792.0;
    FSPDFPage *page = [doc insertPage:i width:w height:h];
    FSPointF *ptZero = [[FSPointF alloc] init];
    ptZero.x = 0;
    ptZero.y = 0;
    [page addImage:image frame_index:i position:ptZero width:w height:h
```

```
auto_generate_content:YES];
}

[doc saveAs:output_file save_flags:FSPDFDocSaveFlagNoOriginal];
...
```

3.15 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:

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.

FSWatermarkSettings* settings = [FSWatermarkSettings new];
settings.flags = FSWatermarkSettingsFlagASPageContents | FSWatermarkSettingsFlagOnTop;
settings.offset_x = 0;
settings.offset_y = 0;
settings.opacity = 90;
settings.position = FSPosTopRight;
settings.rotation = -45.f;
settings.scale_x = 1.f;
settings.scale_y = 1.f;

FSWatermarkTextProperties* text_properties = [FSWatermarkTextProperties new];
text_properties.alignment = FSAlignmentCenter;
text_properties.color = 0xF68C21;
text_properties.font_style = FSWatermarkTextPropertiesFontStyleNormal;
text_properties.line_space = 1;
text_properties.font_size = 12.f;
FSFont* new_font = [[FSFont alloc] initWithFont_id:FSFontStdIDTimesB];
text_properties.font = new_font;

FSWatermark* watermark = [[FSWatermark alloc] initWithDocument:doc text:@"Foxit PDF
SDK\nwww.foxitsoftware.com" properties:text_properties settings:settings];
[watermark insertToPage:page];

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

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

```
#include "FSPDFObjC.h"
...
```

```
// Assuming FSPDFDoc doc has been loaded.

FSWatermarkSettings* settings = [FSWatermarkSettings new];
settings.flags = FSWatermarkSettingsFlagASPageContents | FSWatermarkSettingsFlagOnTop;
settings.offset_x = 0.f;
settings.offset_y = 0.f;
settings.opacity = 20;
settings.position = FSPosCenter;
settings.rotation = 0.0f;

FSImage* image = [[FSImage alloc] initWithPath:image_file_path];
FSBitmap* bitmap = [image getFrameBitmap:0];
settings.scale_x = [page getWidth] * 0.618f / [bitmap getWidth];
settings.scale_y = settings.scale_x;

FSWatermark* watermark = [[FSWatermark alloc] initWithDocument:doc image:image
frame_index:0 settings:settings];
[watermark insertToPage:page];

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

3.15.3 How to remove all watermarks from a page

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded and parsed.
...
[page removeAllWatermarks];

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

3.16 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:

3.16.1 How to generate a barcode bitmap from a string

```
#include "FSPDFObjC.h"
...

// Strings used as barcode content.
NSString *code_string = @"TEST-SHEET";
// Barcode format types.
FSBarcodeFormat code_format = FSBarcodeFormatCode39;
// Format error correction level of QR code.
FSBarcodeQRErrorCorrectionLevel qr_level = FSBarcodeQRCorrectionLevelLow;
// Unit width for barcode in pixels, preferred value is 1-5 pixels.
int unitWidth = 2;
// Unit height for barcode in pixels, preferred value is >= 20 pixels.
int unitHeight = 120;

FSBarcode *barcode = [[FSBarcode alloc] init];
FSBitmap *bitmap = [barcode generateBitmap: code_string format:code_format
unit_width:unit_width unit_height:unit_height level:qr_level];
...
```

3.17 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:

3.17.1 How to encrypt a PDF file with Certificate

```
#include "FSPDFObjC.h"
...

FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:input_file];
FSErrorCode code = [doc load:nil];
if (code != FSErrSuccess) {
    return -1;
}

// Do encryption.
NSMutableArray<NSData *> *envelopes = @[].mutableCopy;
NSMutableData *initial_key = [NSMutableData new];
NSString *cert_file_path = [input_path stringByAppendingPathComponent:@"foxit.cer"];
```

```
// GetCertificateInfo is implemented in user side to get information from certificate
file.

if (!GetCertificateInfo(cert_file_path, envelopes, initial_key, true, 16)) {
    return -1;
}
FSCertificateSecurityHandler *handler = [[FSCertificateSecurityHandler alloc] init];
FSCertificateEncryptData *encrypt_data = [[FSCertificateEncryptData alloc]
initWithIs_encrypt_metadata:YES cipher:FSSecurityHandlerCipherAES envelopes:envelopes];
[handler initialize:encrypt_data encrypt_key:initial_key];

[doc setSecurityHandler:handler];
NSString *output_file = [output_directory
stringByAppendingPathComponent:@"certificate_encrypt.pdf"];
[doc saveAs:output_file save_flags:FSPDFDocSaveFlagNoOriginal];
...
```

3.17.2 How to encrypt a PDF file with Foxit DRM

```
#include "FSPDFObjC.h"
...

FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:input_file];
FSErrorCode code = [doc load:nil];
if (code != FSErrSuccess) {
    return -1;
}

// Do encryption.
FSDRMSecurityHandler *handler = [[FSDRMSecurityHandler alloc] init];
NSString *file_id = @"Simple-DRM-file-ID";
NSData *initialize_key = [@"Simple-DRM-initialize-key"
dataUsingEncoding:NSUTF8StringEncoding];
FSDRMEncryptData *encrypt_data = [[FSDRMEncryptData alloc]
initWithIs_encrypt_metadata:TRUE sub_filter:@"Simple-DRM-filter"
cipher:FSSecurityHandlerCipherAES key_length:16 is_owner:YES
user_permissions:0xfffffffc];
[handler initialize:encrypt_data file_id:file_id initial_key:initialize_key];
[doc setSecurityHandler:handler];

NSString *output_file = [output_directory
stringByAppendingPathComponent:@"foxit_drm_encrypt.pdf"];
[doc saveAs:output_file save_flags:FSPDFDocSaveFlagNoOriginal];
...
```

3.18 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:

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSPDFPage* page = [doc getPage:0];
// Parse PDF page.
[page startParse:FSPDFPageParsePageNormal pause:nil is_reparse:NO];

FSReflowPage* reflow_page = [[FSReflowPage alloc] initWithPage:page];
// Set some arguments used for parsing the relfow page.
[reflow_page setLineSpace:0];
[reflow_page setScreenMargin:margin.left top:(int)margin.top right:(int)margin.right
bottom:(int)margin.bottom];
[reflow_page setScreenSize:size.x height:size.y];
[reflow_page setZoom:100];
[reflow_page setParseFlags:FSReflowPageNormal];

// Parse reflow page.
[reflow_page startParse:nil];

// 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];

// Create a bitmap for rendering the reflow page. The bitmap size contains the margin.
FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:(int)(content_width + margin.left +
margin.right) height:(int)(content_height + margin.top + margin.bottom)
format:FSBitmapDIBArgb buffer:nil pitch:0];
[bitmap fillRect:0xFFFFFFFF rect:nil];

// Render reflow page.
FSRenderer* renderer = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:NO];
FSMatrix2D* matrix = [reflow_page getDisplayMatrix:0 offset_y:0];
[renderer startRenderReflowPage:reflow_page matrix:matrix pause:nil];
...
```

3.19 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.

3.20 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:

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

```
#include "FSPDFObjC.h"
...

FSPSI *psi = [[FSPSI alloc] initWithWidth:480 height:180 simulate:YES];

// 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;
FSPathPointType type = FSPathTypeMoveTo;

FSPointF *pt = [[FSPointF alloc] init];
pt.x = 121.3043f;
pt.y = 326.6846f;
[psi addPoint:pt type:type pressure:pressure];
...
```

3.21 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:

3.21.1 How to open a document including wrapper data.

```
#include "FSPDFObjC.h"
...

FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:file_name];
FSErrorCode code = [doc load:nil];
if (code != FSErrSuccess) {
    return -1;
}
if (![doc isWrapper]) {
    return -1;
}
long long offset = [doc getWrapperOffset];
FSFileRead *file_reader = [[FSFileRead alloc] initWithSourceFilePath:file_name
offset:offset];

FSPDFDoc *doc_real = [[FSPDFDoc alloc] initWithFile_read:file_reader is_async:NO];
code = [doc_real load:nil];
if (code != FSErrSuccess) {
    return -1;
}
...
```

3.22 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:

3.22.1 How to remove some properties from catalog dictionary

```
#include "FSPDFObjC.h"
...

FSPDFDictionary *catalog = [document getCatalog];
if (catalog == NULL)
    return;

NSArray *key_strings = [[NSArray alloc] initWithObjects:@"Type", @"Boolean", @"Name",
@"String", @"Array", @"Dict", nil];
for (int i = 0; i < [key_strings count]; i++) {
    if ([catalog hasKey:key_strings[i]]) {
        [catalog removeAt:key_strings[i]];
    }
}
...
```

3.23 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:

3.23.1 How to create a text object in a PDF page

```
#include "FSPDFObjC.h"
...

long position = [page getLastGraphicsObjectPosition:FSGraphicsObjectTypeText];
FSTextObject *text_object = [FSTextObject create];

text_object.fillColor = 0xFFFF7F00;

// Prepare text state
FSTextState *state = [[FSTextState alloc] init];
state.font_size = 80.0f;
FSFont *font = [[FSFont alloc] initWithName:@"Simsun" styles:FSFontStylesSmallCap
charset:FSFontCharsetGB2312 weight:0];
state.font = font;
state.textmode = FSTextStateModeFill;
[text_object setTextState:page text_state:state is_italic:false weight:750];

// Set text.
text_object.text = @"Foxit Software";
long last_position = [page insertGraphicsObject:position graphics_object:text_object];
...
```

3.23.2 How to add an image logo to a PDF page

```
#include "FSPDFObjC.h"
...

long position = [page getLastGraphicsObjectPosition:FSGraphicsObjectTypeImage];
FSImage *image = [[FSImage alloc] initWithPath:image_file];
FSImageObject *image_object = [FSImageObject create:[page getDocument]];
[image_object setImage:image frame_index:0];

float width = [image getWidth];
float height = [image getHeight];

float page_width = [page getWidth];
float page_height = [page getHeight];

// Please notice the matrix value.
image_object.matrix = [[FSMatrix2D alloc] initWithA1:width b1:0 c1:0 d1:height
e1:(page_width - width) / 2.0f f1:(page_height - height) / 2.0f];
```

```
[page insertGraphicsObject:position graphics_object:image_object];
[page generateContent];
...
```

3.24 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:

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

```
#include "FSPDFObjC.h"
...

long position = [page getFirstGraphicsObjectPosition:FSGraphicsObjectTypeText];
FSTextObject *text_obj = [[page getGraphicsObject:position] getTextObject];
FSMarkedContent *content = [text_obj getMarkedContent];
int item_count = [content getItemCount];

// Get marked content property
for (int i = 0; i < item_count; i++) {
    NSString *tag_name = [content getItemTagName:i];
    int mcid = [content getItemMCID:i];
}
...
```

3.25 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 FSLayerTree object first and then call function FSLayerTree::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:

3.25.1 How to create a PDF layer

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSLayerTree* layertree = [[FSLayerTree alloc] initWithDocument:doc];
FSLayerNode* root = [layertree getRootNode];
if ([root isEmpty]) {
    return -1;
}
...
```

3.25.2 How to set all the layer nodes information

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFDoc doc has been loaded.
...

FSLayerTree* layertree = [[FSLayerTree alloc] initWithDocument:doc];
FSLayerNode* root = [layertree getRootNode];
if ([root isEmpty]) {
    return -1;
}
setAllLayerNodesInformation(root);

void setAllLayerNodesInformation(FSLayerNode* layer_node) {
    if ([layer_node hasLayer]) {
        [layer_node setDefaultVisible:YES];
        [layer_node setExportUsage:FSLayerTreeStateUndefined];
        [layer_node setViewUsage:FSLayerTreeStateOFF];
        FSLayerPrintData* print_data = [[FSLayerPrintData alloc]
initWithSubtype:@"subtype_print" print_state:FSLayerTreeStateON];
        [layer_node setPrintUsage:print_data];
        FSLayerZoomData* zoom_data = [[FSLayerZoomData alloc] initWithMin_factor:1
max_factor:10];
        [layer_node setZoomUsage:zoom_data];
        NSString* new_name = [NSString
stringWithFormat:@"%@%@",@"[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++) {
        FSLayerNode* child = [layer_node getChild:i];
        setAllLayerNodesInformation(child);
    }
}
...
```

3.25.3 How to edit layer tree

```
#include "FSPDFObjC.h"
...
```

```
FSLayerTree* layertree = [[FSLayerTree alloc] initWithDocument:doc];
FSLayerNode* root = [layertree getRootNode];
if ([root isEmpty]) {
    return -1;
}
int children_count = [root getChildrenCount];
[root removeChild:children_count-1];
FSLayerNode* child = [root getChild:children_count-2];
FSLayerNode* child0 = [root getChild:0];
[child moveTo:child0 index:0];
[child addChild:0 name:@"AddedLayerNode" has_Layer:YES];
[child addChild:0 name:@"AddedNode" has_Layer:NO];
...
```

3.26 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:

3.26.1 How to sigh the PDF document with a signature

```
#include "FSPDFObjC.h"
...

NSString *filter = @"Adobe.PPKLite";
NSString *sub_filter = @"adbe.pkcs7.detached";

if (!use_default) {
    InitializeOpenssl();
    sub_filter = @"adbe.pkcs7.sha1";
    SignatureCallback *sig_callback = [[SignatureCallback alloc]
initWithSubFilter:sub_filter];
    [FSLibrary registerSignatureCallback:filter sub_filter:sub_filter
signature_callback:sig_callback];
}
```

```
[filter UTF8String], [sub_filter UTF8String]);
FSPDFPage *pdf_page = [pdf_doc getPage:0];
// Add a new signature to first page.
FSSignature *new_signature = AddSiganture(pdf_page, sub_filter);
// Set filter and subfilter for the new signature.
[new_signature setFilter:filter];
[new_signature setSubFilter:sub_filter];

// Sign the new signature.
NSString *signed_pdf_path = [output_directory
stringByAppendingPathComponent:@"signed_newsignature.pdf"];
if (use_default)
    signed_pdf_path = [output_directory
stringByAppendingPathComponent:@"signed_newsignature_default_handler.pdf"];

NSString *cert_file_path = [input_path
stringByAppendingPathComponent:@"foxit_all.pfx"];
NSString *cert_file_password = @"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:cert_file_path cert_password:cert_file_password
digest_algorithm:FSSignatureDigestSHA1 save_path:signed_pdf_path client_data:nil
pause:nil];

// Open the signed document and verify the newly added signature (which is the last
one).
FSPDFDoc *signed_pdf_doc = [[FSPDFDoc alloc] initWithPath:signed_pdf_path];
FSErrorCode error_code = [signed_pdf_doc load:nil];
if (FSErrSuccess != error_code) {
    return;
}
// Get the last signature which is just added and signed.
int sig_count = [signed_pdf_doc getSignatureCount];
FSSignature *signed_signature = [signed_pdf_doc getSignature:sig_count - 1];
// Verify the signature.
[signed_signature startVerify:nil pause:nil];
...
```

3.26.2 How to implement signature callback function of signing

```
#include "FSPDFObjC.h"
#include "openssl/rsa.h"
#include "openssl/evp.h"
#include "openssl/objects.h"
#include "openssl/x509.h"
#include "openssl/err.h"
#include "openssl/pem.h"
#include "openssl/ssl.h"
#include "openssl/pkcs12.h"
#include "openssl/rand.h"
#include "openssl/pkcs7.h"

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#include <netdb.h>

#include <string>

// some base type declarations
typedef std::string String;

// Used for implementing SignatureCallback.
class DigestContext {
public:
    DigestContext(id<FSFileReaderCallback> file_read_callback, NSArray<NSNumber *>
*byte_range_array)
    : file_read_callback_(file_read_callback), byte_range_array_(byte_range_array) {}
    ~DigestContext() {}

    id<FSFileReaderCallback> GetFileReadCallback() {
        return file_read_callback_;
    }
    NSUInteger GetByteRangeSize() {
        return byte_range_array_.count;
    }
    unsigned int GetByteRangeElement(NSUInteger index) {
        if (!byte_range_array_)
            return 0;
        return [byte_range_array_[index] unsignedIntValue];
    }

    SHA_CTX sha_ctx_;

protected:
    id<FSFileReaderCallback> file_read_callback_;
    NSArray<NSNumber *> *byte_range_array_;
};

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

    void Release() {
        delete this;
    }
    bool StartCalcDigest(id<FSFileReaderCallback> file, NSArray<NSNumber *>
*byte_range_array, FSSignature *signature, const void *client_data);
    FSProgressiveState ContinueCalcDigest(const void *client_data, id<FSPauseCallback>
pause);
    NSData *GetDigest(const void *client_data);
    NSData *Sign(const void *digest, uint32 digest_length, std::string cert_path,
                 std::string password, FSSignatureDigestAlgorithm digest_algorithm,
                 void *client_data);
    FSSignatureStates VerifySigState(const void *digest, uint32 digest_length,
                                     const void *signed_data, uint32 signed_data_len,
                                     void *client_data);
    bool IsNeedPadData() { return false; }

protected:
    bool GetTextFromFile(unsigned char *plainString);

    unsigned char *PKCS7Sign(std::string cert_file_path, String cert_file_password,
```

```
                              String plain_text, int &signed_data_size);
    bool PKCS7VerifySignature(String signed_data, String plain_text);
    bool ParseP12File(std::string cert_file_path, String cert_file_password,
                      EVP_PKEY **pkey, X509 **x509, STACK_OF(X509) * *ca);
    ASN1_INTEGER *CreateNonce(int bits);

private:
    std::string sub_filter_;
    DigestContext *digest_context_;

    std::string cert_file_path_;
    std::string cert_file_password_;
};

#define FREE_CERT_KEY if(pkey)\
EVP_PKEY_free(pkey);\
if(x509)\
X509_free(x509);\
if(ca)\
sk_X509_free(ca);

void InitializeOpenssl() {
    //      SSLeay_add_all_algorithms();
}

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;
    id<FSFileReaderCallback> file_read = digest_context_->GetFileReadCallback();
    NSData *data = [file_read readBlock:digest_context_->GetByteRangeElement(0)
size:digest_context_->GetByteRangeElement(1)];
    [data getBytes:file_buffer length:data.length];

    data = [file_read readBlock:digest_context_->GetByteRangeElement(2)
size:digest_context_->GetByteRangeElement(3)];
    [data getBytes:file_buffer + (digest_context_->GetByteRangeElement(1) -
digest_context_->GetByteRangeElement(0)) length:data.length];
    return true;
}

bool SignatureCallbackImpl::StartCalcDigest(id<FSFileReaderCallback> file,
NSArray<NSNumber *> *byte_range_array, FSSignature *signature, const void *client_data)
{
    if (digest_context_) {
        delete digest_context_;
        digest_context_ = NULL;
    }
    digest_context_ = new DigestContext(file, byte_range_array);
    if (!SHA1_Init(&digest_context_->sha_ctx_)) {
        delete digest_context_;
        digest_context_ = NULL;
        return false;
    }
    return true;
```

```
}

FSProgressiveState SignatureCallbackImpl::ContinueCalcDigest(const void *client_data,
id<FSPauseCallback> pause) {
    if (!digest_context_)
        return FSProgressiveError;

    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 FSProgressiveError;

    SHA1_Update(&digest_context_->sha_ctx_, file_buffer, file_length);
    free(file_buffer);
    return FSProgressiveFinished;
}

NSData *SignatureCallbackImpl::GetDigest(const void *client_data) {
    if (!digest_context_)
        return nil;
    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 nil;
    NSData *digest = [NSData dataWithBytes:reinterpret_cast<const void *>(md)
length:SHA_DIGEST_LENGTH];
    OPENSSL_free(md);
    return digest;
}

NSData *SignatureCallbackImpl::Sign(const void *digest, uint32 digest_length,
std::string cert_path,
                                    std::string password, FSSignatureDigestAlgorithm
digest_algorithm,
                                    void *client_data) {
    if (!digest_context_)
        return nil;
    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, password,
                                                  plain_text, signed_data_length);
    if (!signed_data_buffer)
        return nil;

    NSData *signed_data = [NSData dataWithBytes:(const void *) signed_data_buffer
length:signed_data_length];
    free(signed_data_buffer);
    return signed_data;
}

FSSignatureStates 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 FSSignatureStateVerifyErrorData;
    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 FSSignatureStateUnknown;
    }

    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 ? FSSignatureStateVerifyValid : FSSignatureStateVerifyInvalid;
}

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(std::string 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, @"rb");
#else
    file = fopen(cert_file_path.c_str(), "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, cert_file_password.c_str(), pkey, x509, ca)) {
        return false;
    }

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

unsigned char *SignatureCallbackImpl::PKCS7Sign(std::string 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.size() > 32)
        PKCS7_ctrl(p7, PKCS7_OP_SET_DETACHED_SIGNATURE, 1, NULL);

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

# define CHECKED_STACK_OF(type, p) \
((_STACK*) (1 ? p : (STACK_OF(type)*)0))
    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.c_str(), (int) plain_text.size());
    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 = (int) 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.c_str(), (int) signed_data.size());
    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.c_str(), (int)
plain_text.size());
        X509 *x509 = PKCS7_cert_from_signer_info(p7, sign_info);
        x509 = NULL;
        if (1 == PKCS7_verify(p7, NULL, NULL, p7bio, NULL, PKCS7_NOVERIFY))
            bSigAppr = true;
        BIO_free(p7bio);
    }
    PKCS7_free(p7);
    BIO_free(vin);
    return bSigAppr;
}
...
```

3.27 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:

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

```
#include "FSPDFObjC.h"
...

// Add link annotation
FSRectF* annot_rect = [[FSRectF alloc] initWithLeft1:350 bottom1:350 right1:380
top1:400];
FSLink* link = [[FSLink alloc] initWithAnnot:[page addAnnot:FSAnnotLink
rect:annot_rect]];
[link setHighlightingMode:FSAnnotHighlightingToggle];
```

```
// Add action for link annotation
FSPDFDoc* doc = [page getDocument];
FSURIAction* action = [[FSURIAction alloc] initWithAction:[FSAction create:doc
action_type:FSActionTypeURI]];
[action setTrackPositionFlag:YES];
[action setURI:@"www.foxitsoftware.com"];
[link setAction:action];
// Appearance should be reset.
[link resetAppearanceStream];
...
```

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

```
#include "FSPDFObjC.h"
...

// Assuming FSPDFPage page has been loaded.
...

// Add link annotation
FSRectF* annot_rect = [[FSRectF alloc] initWithLeft1:350 bottom1:350 right1:380
top1:400];
FSLink* link = [[FSLink alloc] initWithAnnot:[page addAnnot:FSAnnotLink
rect:annot_rect]];
[link setHighlightingMode:FSAnnotHighlightingToggle];

// Add action for link annotation
FSPDFDoc* doc = [page getDocument];
FSGotoAction* action = [[FSGotoAction alloc] initWithAction:[FSAction create:doc
action_type: FSActionTypeGoto]];
action.destination = [FSDestination createFitPage:doc page_index:0];
// Appearance should be reset.
[link resetAppearanceStream];
...
```

3.28 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 FSJavaScriptAction is derived from FSAction and offers functions to get/set JavaScript action data.

Example:

3.28.1 How to add JavaScript Action to Document

```
#include "FSPDFObjC.h"
...
```

```
// Load Document doc.
...

FSJavaScriptAction *javascript_action = [[FSJavaScriptAction alloc]
initWithAction:[FSAction create:[form getDocument]
action_type:FSActionTypeJavaScript]];
javascipt_action.script = @"app.alert(\"Hello Foxit \");";
FSAdditionalAction *additional_act = [[FSAdditionalAction alloc] initWithDoc:doc
pdf_dict:nil];
[additional_act setAction:FSAdditionalActionTriggerDocWillClose
action:javascipt_action];
...
```

3.28.2 How to add JavaScript Action to Annotation

```
#include "FSPDFObjC.h"
...

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

FSJavaScriptAction *javascript_action = [[FSJavaScriptAction alloc]
initWithAction:[FSAction create:[form getDocument]
action_type:FSActionTypeJavaScript]];
javascipt_action.script = @"app.alert(\"Hello Foxit \");";
FSAdditionalAction *additional_act = [[FSAdditionalAction alloc]
initWithAnnot:widget_annot];
[additional_act setAction:FSAdditionalActionTriggerAnnotMouseButtonPressed
action:javascipt_action];
...
```

3.28.3 How to add JavaScript Action to FormField

```
#include "FSPDFObjC.h"
...

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

FSJavaScriptAction *javascript_action = [[FSJavaScriptAction alloc]
initWithAction:[FSAction create:[form getDocument]
action_type:FSActionTypeJavaScript]];
javascipt_action.script = @"AFSimple_Calculate(\"SUM\", new Array (\"Text Field0\",
\"Text Field1\"));";
FSAdditionalAction *additional_act = [[FSAdditionalAction alloc] initWithField:field];
[additional_act setAction:FSAdditionalActionTriggerFieldRecalculateValue
action:javascipt_action];
...
```

3.29 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 FSRedaction to create a redaction module. If module "Redaction" is not defined in the license information which is used in function FSLibrary::initialize, that means user has no right in using redaction related functions and this constructor will throw exception FSErrInvalidLicense.
- Then call function FSRedaction::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 FSRedaction.apply to apply redaction in marked areas: remove the text or graphics under marked areas permanently.

Example:

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

```
#include "FSPDFObjC.h"
...

// Assuming that FSPDFDoc doc has been loaded.
...

FSRedaction *redaction = [[FSRedaction alloc] initWithDocument:doc];
// Parse PDF page.
FSPDFPage *page = [doc getPage:0];
[page startParse:FSPDFPageParsePageNormal pause:nil is_reparse:NO];
FSTextPage *text_page = [[FSTextPage alloc] initWithPage:page
flags:FSTextPageParseTextNormal];
FSTextSearch *text_search = [[FSTextSearch alloc] initWithText_page:text_page];
[text_search setPattern:@"PDF"];
FSRectFArray *rect_array = [[FSRectFArray alloc] init];
while ([text_search findNext]) {
    FSRectFArray *matchrects = [text_search getMatchRects];
    for (int z = 0; z < [matchrects getSize]; z++) {
        FSRectF *temp_rect = [matchrects getAt:z];
        [rect_array add:temp_rect];
    }
}
if ([rect_array getSize] > 0) {
    FSRedact *redact = [redaction markRedactAnnot:page rects:rect_array];
    [redact resetAppearanceStream];
    [doc saveAs:[output_directory
stringByAppendingString:@"AboutFoxit_redected_default.pdf"]
save_flags:FSPDFDocSaveFlagNormal];

    // set border color to Green.
    [redact setBorderColor:0x00FF00];
    // set fill color to Blue.
```

```
    [redact setFillColor:0x0000FF];
    // set rollover fill color to Red.
    [redact setApplyFillColor:0xFF0000];
    [redact resetAppearanceStream];
    [doc saveAs:[output_directory
stringByAppendingString:@"AboutFoxit_redected_setColor.pdf"]
save_flags:FSPDFDocSaveFlagNormal];

    [redact setOpacity:0.5];
    [redact resetAppearanceStream];
    [doc saveAs:[output_directory
stringByAppendingString:@"AboutFoxit_redected_setOpacity.pdf"]
save_flags:FSPDFDocSaveFlagNormal];
}
```

3.30 Comparison

Comparison feature lets you see the differences in two versions of a PDF. Foxit PDF SDK provides APIs to compare two PDF documents page by page, the differences between the two documents will be returned. For now, it only supports comparing the text of the PDF document, and in the next release, more PDF elements will be supported.

The differences can be defined into three types: delete, insert and replace. You can save these differences into a PDF file and mark them as annotations.

***Note**: To use the comparison feature, please make sure the license key has the permission of the 'Comparison' module*

Example:

3.30.1 How to compare two PDF documents and save the differences between them into a PDF file?

```
#include "FSPDFObjC.h"
...

FSPDFDoc *base_doc = [[FSPDFDoc alloc] initWithPath:@"input_base_file"];
errorCode = [base_doc load:@""];
if (errorCode != FSErrSuccess) {
    return -1;
}

FSPDFDoc *compared_doc = [[FSPDFDoc alloc] initWithPath:@"input_compared_file"];
errorCode = [compared_doc load:@""];
if (errorCode != FSErrSuccess) {
    return -1;
}

FSComparison* comparison = [[FSComparison alloc] initWithBase_doc:base_doc
compared_doc:compared_doc];

// Start comparison.
FSCompareResults* result = [comparison doCompare:0 compared_page_index:0
compare_flags:FSComparisonCompareTypeText];
```

```
int oldInfoSize = [result.results_base_doc getSize];
int newInfoSize = [result.results_compared_doc getSize];
FSPDFPage* page = [compared_doc getPage:0];
for (int i=0; i<newInfoSize; i++)
{
    FSCompareResultInfo* item = [result.results_compared_doc getAt:i];
    FSCompareResultInfoCompareResultType type = item.type;
    if (type == FSCompareResultInfoCompareResultTypeDeleteText)
    {
        NSString* res_string = [NSString stringWithFormat:@"\"%@\"",
item.diff_contents];

        // Add stamp to mark the "delete" type differences between the two documents.
        createDeleteTextStamp(page, item.rect_array, 0xff0000, res_string, @"Compare :
Delete", @"Text");
    }
    else if (type == FSCompareResultInfoCompareResultTypeInsertText)
    {
        NSString* res_string = [NSString stringWithFormat:@"\"%@\"",
item.diff_contents];

        // Highlight the "insert" type differences between the two documents.
        createHighlightRect(page, item.rect_array, 0x0000ff, res_string, @"Compare :
Insert", @"Text");
    }
    else if (type == FSCompareResultInfoCompareResultTypeReplaceText)
    {
        NSString* res_string = [NSString stringWithFormat:@"[Old]: \"%@\"\r\n[New]:
\"%@\"",[result.results_base_doc getAt:i].diff_contents,item.diff_contents];

        // Highlight the "replace" type differences between the two documents.
        createHighlightRect(page, item.rect_array, 0xe7651a, res_string, @"Compare :
Replace", @"Text");
    }
}

// Save the comparison result to a PDF file.
[compared_doc saveAs:[output_directory stringByAppendingString:@"result.pdf"]
save_flags:FSPDFDocSaveFlagNormal];
```

3.31 Compliance (PDF/A)

PDF/A is an ISO-standardized version of the PDF specialized for use in the archiving and long-term preservation of electronic documents. PDF/A differs from PDF by prohibiting features unsuitable for long-term archiving, such as font linking (as opposed to font embedding), encryption, JavaScript, audio, video and so on.

Foxit PDF SDK provides APIs to convert a PDF to be compliance with PDF/A standard, or verify whether a PDF is compliance with PDF/A standard. It supports the PDF/A version including PDF/A-1a, PDF/A-1b, PDF/A-2a, PDF/A-2b, PDF/A-2u, PDF/A-3a, PDF/A-3b, PDF/A-3u (ISO 19005- 1, 19005 -2 and 19005-3).

This section will provide instructions on how to set up your environment for running the 'compliance' demo.

3.31.1 System requirements

Platform: Windows, Linux, Mac

Programming Language: C++, Java, C#, Objective-C

License Key requirement: 'Compliance' module permission in the license key

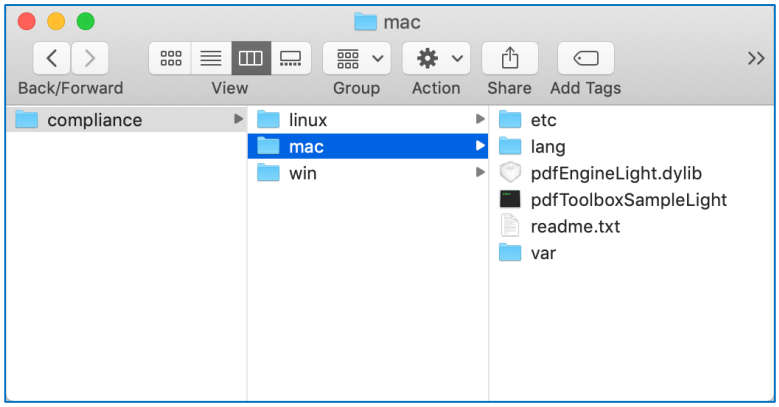
SDK Version: Foxit PDF SDK 6.4

3.31.2 Compliance resource files

Please contact Foxit support team or sales team to get the Compliance (PDF-A) resource files package.

After getting the package, extract it to a desired directory (for example, extract the package to a directory named "compliance"), and you can see the resource files for Compliance are as follows:

For **Mac**:



3.31.3 How to run the Compliance demo

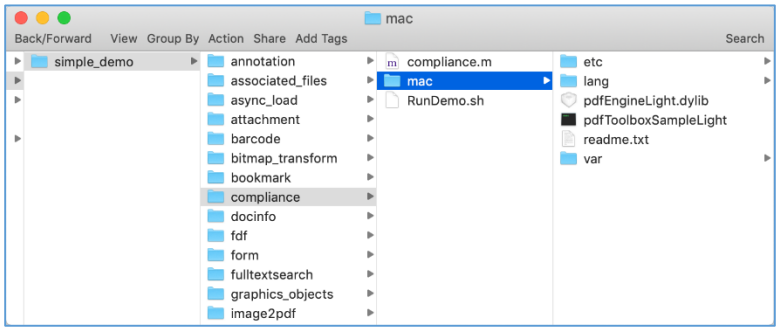
Foxit PDF SDK provides a Compliance demo located in the "examples/simple_demo/compliance" folder to show you how to use Foxit PDF SDK to verify whether a PDF is compliance with PDF/A standard, and convert a PDF to be compliance with PDF/A standard.

3.31.3.1 Build a Compliance resource directory

Before running the Compliance demo, you should first build a compliance resource directory, and then pass the directory to Foxit PDF SDK API **ComplianceEngine::initialize** to initialize compliance engine.

For Mac platform, you can directly use the "compliance/mac" resource folder as the compliance resource directory.

Put the "mac" folder under "compliance" directory into "\examples\simple_demo\compliance" folder (see the following picture), and then follow the contents below to configure the demo.



3.31.3.2 Configure the demo

Configure the demo in the "\examples\simple_demo\compliance**compliance.m**" file.

Specify the Compliance resource directory

Add the compliance resource directory as follows, which will be used to initialize the compliance engine.

```
157         @try {
158             // "compliance_resource_folder_path" is the path of compliance resource folder. Please refer to Developer Guide for more details.
159             NSString* compliance_resource_folder_path = @"mac";
160             // If you use an authorization key for Foxit PDF SDK, please set a valid unlock code string to compliance_engine_unlockcode for FSComplianceEngine.
161             // If you use a trial key for Foxit PDF SDK, just keep compliance_engine_unlockcode as an empty string.
162             NSString* compliance_engine_unlockcode = @"";
163
164             if ([compliance_resource_folder_path length] < 1) {
165                 NSLog(@"compliance_resource_folder_path is still empty. Please set it with a valid path to compliance resource folder path.");
166                 return -1;
167             }
168             // Initialize compliance engine.
169             errorCode = [FSComplianceEngine initialize:compliance_resource_folder_path compliance_engine_unlockcode:compliance_engine_unlockcode];
```

***Note**: If you have purchased an authorization license key (includes 'Compliance' module permission), Foxit sales team will send you an extra unlock code for initializing compliance engine.*

(Optional) Set language for compliance engine

ComplianceEngine::setLanguage function is used to set language for compliance engine. The default language is "English", and the supported languages are as follows:

"Czech", "Danish", "Dutch", "English", "French", "Finnish", "German", "Italian", "Norwegian", "Polish", "Portuguese", "Spanish", "Swedish", "Chinese-Simplified", "Chinese-Traditional", "Japanese", "Korean".

For example, uncomment the **ComplianceEngine::setLanguage** method, and set the language to "Chinese-Simplified".

```
186             // Set language. If not set language to FSComplianceEngine, "English" will be used as default.
187             [FSComplianceEngine setLanguage:@"Chinese-Simplified"];
```

(Optional) Set a temp folder for compliance engine

ComplianceEngine::setTempFolderPath function is used to set a temp folder to store several files for proper processing (e.g verifying or converting). If no custom temp folder is set by this function, the default temp folder in system will be used.

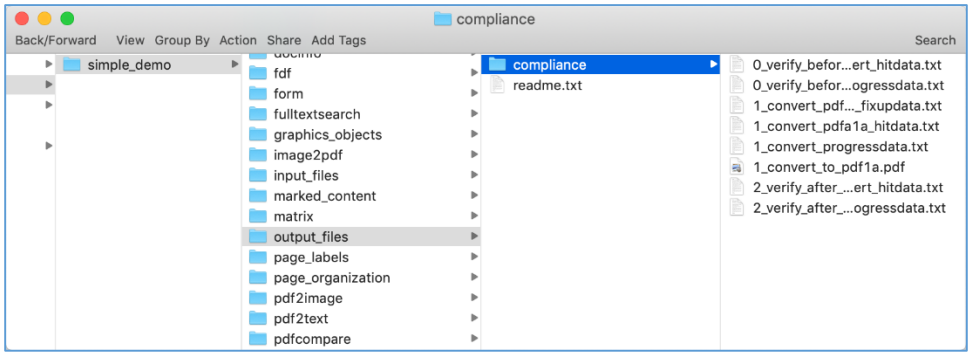
For example, uncomment the **ComplianceEngine::setTempFolderPath** method, and set the path to "compliance_temp". (Assume that you have created a folder name "compliance_temp" in the "\examples\simple_demo\compliance" folder.)

```
183             // Set custom temp folder path for FSComplianceEngine.
184             [FSComplianceEngine setTempFolderPath:@"compliance_temp"];
```

3.31.3.3 Run the demo

The demo will verify whether the PDF ("\examples\simple_demo\input_files\AboutFoxit.pdf") is compliance with PDF/A-1a standard, and convert the PDF to be compliance with PDF/A-1a standard.

Once you run the demo successfully, the output files are located in "\examples\simple_demo\output_files\compliance" folder as follows:



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 OC API reference
sdk_folder/doc/Foxit PDF SDK OC 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