



DEVELOPER GUIDE FOXIT PDF SDK

For Go

TABLE OF CONTENTS

1	Introduction to Foxit PDF SDK	1
1.1	Why Choose Foxit PDF SDK.....	1
1.2	Foxit PDF SDK for Go API.....	2
1.3	Evaluation	2
1.4	License.....	2
1.5	About this guide	2
2	Getting Started.....	3
2.1	System Requirements.....	3
2.2	What is in the package	3
2.3	How to run a demo.....	3
2.4	How to create a simple project.....	5
3	WORKING WITH SDK API.....	8
3.1	Initialize Library	8
3.1.1	How to initialize Foxit PDF SDK.....	8
3.2	Document.....	8
3.2.1	How to create a PDF document from scratch.....	9
3.2.2	How to load an existing PDF document from file path	9
3.2.3	How to load an existing PDF document from a memory buffer	9
3.2.4	How to load an existing PDF document from a file read callback object.....	9
3.2.5	How to load PDF document and get the first page of the PDF document.....	11
3.2.6	How to save a PDF to a file.....	12
3.2.7	How to save a document into memory buffer by FileWriterCallback.....	12
3.3	Page	14
3.3.1	How to get page size.....	14
3.3.2	How to calculate bounding box of page contents.....	14

3.3.3	How to create a PDF page and set the size	14
3.3.4	How to delete a PDF page	15
3.3.5	How to flatten a PDF page.....	15
3.3.6	How to get and set page thumbnails in a PDF document.....	15
3.4	Render	16
3.4.1	How to render a page to a bitmap	16
3.4.2	How to render page and annotation	17
3.5	Attachment.....	17
3.5.1	How to export the embedded attachment file from a PDF and save it as a single file	18
3.5.2	How to remove all the attachments of a PDF	18
3.6	Text Page	19
3.6.1	How to extract text from a PDF page.....	19
3.6.2	How to get the text within a rectangle area in a PDF.....	19
3.7	Text Search	20
3.7.1	How to search a text pattern in a PDF.....	20
3.8	Search and Replace	21
3.8.1	System requirements	21
3.8.2	How to work with the search and replace function	21
3.9	Text Link.....	22
3.9.1	How to retrieve hyperlinks in a PDF page	22
3.10	Bookmark.....	22
3.10.1	How to find and list all bookmarks of a PDF.....	23
3.10.2	How to insert a new bookmark	23
3.10.3	How to create a table of contents based on bookmark information in PDFs	23
3.11	Form (AcroForm)	24
3.11.1	How to load the forms in a PDF.....	24
3.11.2	How to count form fields and get/set the properties.....	25
3.11.3	How to export the form data in a PDF to a XML file	25

3.11.4	How to import form data from a XML file	25
3.11.5	How to get coordinates of a form field.....	26
3.12	XFA Form	26
3.12.1	How to load XFADoc and represent an Interactive XFA form	27
3.12.2	How to export and import XFA form data.....	27
3.13	Form Filler	28
3.14	Form Design	28
3.14.1	How to add a text form field to a PDF	28
3.14.2	How to remove a text form field from a PDF.....	29
3.15	Annotations	29
3.15.1	General	29
3.15.2	Import annotations from or export annotations to a FDF file.....	34
3.16	Image Conversion.....	34
3.16.1	How to convert PDF pages to bitmap files	34
3.16.2	How to convert an image file to PDF file	35
3.17	Watermark.....	36
3.17.1	How to create a text watermark and insert it into the first page	36
3.17.2	How to create an image watermark and insert it into the first page.....	37
3.17.3	How to remove all watermarks from a page	37
3.18	Barcode	37
3.18.1	How to generate a barcode bitmap from a string	38
3.19	Security	38
3.19.1	How to encrypt a PDF file with Certificate.....	39
3.19.2	How to encrypt a PDF file with Foxit DRM.....	39
3.20	Reflow	40
3.20.1	How to create a reflow page and render it to a bmp file	40
3.21	Asynchronous PDF	41
3.22	Pressure Sensitive Ink	41

3.22.1	How to create a PSI and set the related properties for it.....	41
3.23	Wrapper.....	42
3.23.1	How to open a document including wrapper data	42
3.24	PDF Objects	42
3.24.1	How to remove some properties from catalog dictionary	43
3.25	Page Object	43
3.25.1	How to create a text object in a PDF page	43
3.25.2	How to add an image logo to a PDF page	44
3.26	Marked content.....	44
3.26.1	How to get marked content in a page and get the tag name	45
3.27	Layer	45
3.27.1	How to create a PDF layer	45
3.27.2	How to set all the layer nodes information.....	46
3.27.3	How to edit layer tree	46
3.28	Signature	47
3.28.1	How to sign the PDF document with a signature	47
3.28.2	How to implement signature callback function of signing.....	48
3.29	Long term validation (LTV).....	52
3.29.1	How to establish long term validation of signatures using the default signature	53
3.30	PAdES	56
3.31	PDF Action	57
3.31.1	How to create a URI action and insert to a link annot	57
3.31.2	How to create a GoTo action and insert to a link annot.....	58
3.32	JavaScript	58
3.32.1	How to add JavaScript Action to Document.....	58
3.32.2	How to add JavaScript Action to Annotation.....	59
3.32.3	How to add JavaScript Action to FormField	59
3.32.4	How to add a new annotation to PDF using JavaScript	59

3.32.5	How to get/set properties of annotations (strokeColor, fillColor, readOnly, rect, type) using JavaScript	60
3.32.6	How to destroy annotation using JavaScript.....	60
3.33	Redaction	61
3.33.1	How to redact the text "PDF" on the first page of a PDF.....	61
3.34	Comparison.....	62
3.34.1	How to compare two PDF documents and save the differences between them into a PDF file	63
3.35	Optimization	64
3.35.1	How to optimize PDF files by compressing the color/grayscale/monochrome images	64
3.36	HTML to PDF Conversion	65
3.36.1	System requirements	65
3.36.2	HTML to PDF engine files	65
3.36.3	How to run the html2pdf demo	65
3.36.4	How to work with Html2PDF API	69
3.36.5	How to get HTML data from stream and convert it to a PDF file	69
3.37	Office to PDF Conversion with third-party engines.....	71
3.37.1	System requirements	72
3.37.2	How to convert Word to PDF	72
3.37.3	How to convert Excel to PDF.....	72
3.37.4	How to convert PowerPoint to PDF.....	73
3.38	Office to PDF Conversion with Foxit's self-developed engines	73
3.38.1	System requirements	74
3.38.2	Office to PDF resource files (Foxit PDF Conversion SDK)	74
3.38.3	How to run the office2pdf demo using Foxit PDF Conversion SDK	74
3.38.4	How to convert office files to PDF with Foxit's self-developed engines	74
3.39	Output Preview	75
3.39.1	System requirements	75
3.39.2	How to run the output preview demo.....	75

3.39.3	How to do output preview using Foxit PDF SDK.....	75
3.40	Combination	76
3.40.1	How to combine several PDF files into one PDF file	76
3.41	PDF Portfolio	77
3.41.1	System requirements	77
3.41.2	How to create a new and blank PDF portfolio	77
3.41.3	How to create a Portfolio object from a PDF portfolio	77
3.41.4	How to get portfolio nodes	78
3.41.5	How to add file node or folder node	78
3.41.6	How to remove a node.....	79
3.42	Table Maker	79
3.42.1	System requirements	80
3.42.2	How to add table to a PDF document.....	80
3.43	Accessibility	81
3.43.1	System requirements	81
3.43.2	How to tag a PDF document	81
3.44	PDF to Office Conversion	81
3.44.1	System requirements	82
3.44.2	PDF to Office resource files.....	82
3.44.3	How to run the pdf2office demo	82
3.44.4	How to work with PDF2office API.....	83
3.45	DWG to PDF Conversion.....	84
3.45.1	System requirements	85
3.45.2	DWG To PDF engine files	85
3.45.3	How to run the dwg2pdf demo	85
3.45.4	How to convert DWG to PDF.....	85
3.46	OFD	86
3.46.1	System requirements	86
3.46.2	OFD engine file	86

3.46.3	How to run the ofd demo.....	87
3.46.4	How to implement the conversion between OFD file and PDF file.....	87
3.46.5	How to render OFD page	87
3.47	Paragraph Editing.....	88
3.47.1	System requirements	89
3.47.2	How to work with paragraph editing	89
FAQ	93
Appendix	95
	Supported JavaScript List	95
References	109
Support	110

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 Choose Foxit PDF SDK

Foxit is a 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 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**

Does 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 Go 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 (for C++ and .NET) 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 Go API on Linux and Mac platforms.

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

1.3 Evaluation

Foxit PDF SDK allows users to download a trial version to evaluate the 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 developers who need to integrate Foxit PDF SDK for Go API into their own applications. It aims at introducing the installation package, and the usage of SDK.

2 GETTING STARTED

It's very easy to setup Foxit PDF SDK and see it in action! This guide will provide you with a brief introduction about our SDK package. The following sections introduce the contents of system requirements, the installation package as well as how to run a demo, and create your own project.

2.1 System Requirements

Platform	System Requirement	Note
Linux	x86/x64 (32-bit and 64-bit OS) <ul style="list-style-type: none">The minimum supported version of GCC compiler is gcc5.4.The minimum supported version of GLIBC is GLIBC_2.17.	All Linux for x86/x64 samples have been tested on Ubuntu16.0 32/64 bit. Go 1.18 or later
Mac	Mac OS X 10.9 or higher (64-bit OS)	Go 1.18 or later

2.2 What is in the package

Download the Foxit PDF SDK zip for Go (Linux x86/x64 or Mac x64) package and extract it to a new directory. The release package contains the following folders:

doc:	developer guide
examples:	sample projects and demos
lib:	libraries and license files
res:	the default icc profile files used for output preview demo

2.3 How to run a demo

GCC compiler requirement

For Linux x86/x64, the minimum supported version of GCC compiler is gcc5.4. For the SDK to work properly, make sure your current GCC version is 5.4 or higher, or the libstdc++.so.6 is 6.0.20 or higher.

Foxit PDF SDK for Go API (Linux x86/x64 or Mac x64) provides several simple demos in directory "examples/simple_demo". All these demos (except html2pdf, office2pdf, output preview, pdf2office, dwg2pdf and ofd demos) can be run directly in a terminal using the "RunDemo.sh" file in directory "examples/simple_demo".

open a terminal, and then run the demos following the steps below:

1) **Install Go.**

Ensure Go 1.18 or later is installed on your Linux system. If not, download and install it from <https://golang.google.cn/dl/>.

2) **Set the library path to the environment variable.**

Assume the downloaded package is extracted to
"/home/user/Desktop/foxitpdfsdk_11_0_linux_go" for Linux x86/x64 or
"/Users/user/Desktop/foxitpdfsdk_11_0_mac_go" for Mac x64.

open a terminal and set the library path using the following command:

Using Linux x64 as an example:

```
export LD_LIBRARY_PATH=/home/user/Desktop/foxitpdfsdk_11_0_linux_go/lib/x64:$LD_LIBRARY_PATH
```

For Mac x64:

```
export DYLD_LIBRARY_PATH=/Users/user/Desktop/foxitpdfsdk_11_0_mac_go/lib:$DYLD_LIBRARY_PATH
```

3) **Run the demos.**

In the terminal, navigate to "examples/simple_demo".

- Type **"./RunAllDemo.sh"** to run all the demos.
- Type **"./RunDemo.sh demo_name"** to run a specific single demo, for example, **"./RunDemo.sh bookmark"** will run the bookmark demo.

Note: It is highly recommended to use the **RunDemo.sh** script when running the demo for the first time. This script will automatically perform the following actions:

- Create the **go.mod** file.
- Update and download all dependencies required for all the Go demos.

"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 project name under "examples/simple_demo/output_files/" folder.

HTML to PDF demo

For how to run the **html2pdf** demo, please refer to section "[HTML to PDF Conversion](#)".

Office to PDF demo (for Linux x86/64)

For **office2pdf** demo, you need to refer to section "[Office to PDF Conversion with third-party engines](#)" and "[Office to PDF Conversion with Foxit's self-developed engines](#)".

Output Preview demo

For how to run the **output preview** demo, please refer to section "[Output Preview](#)".

PDF to Office demo (for Linux x86/64)

For how to run the **pdf2office** demo, please refer to section "[PDF to Office Conversion](#)".

Dwg to PDF demo

For how to run the **dwg2pdf** demo, please refer to section "[DWG to PDF Conversion](#)".

OFD demo (for Linux x64)

For how to run the **ofd** demo, please refer to section "[OFD](#)".

2.4 How to create a simple project

In this section, we will show you how to use Foxit PDF SDK for Go (Linux x86/x64 or Mac x64) 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 new project folder named "test".
- 2) Copy the "SamplePDF.pdf" from the "example/simple_demo/input_files" to the folder "test".
- 3) For **Linux x86/x64**, copy the "lib" folder from the "foxitpdfsdk_11_0_linux_go" folder to the project "test" folder.

For **Mac x64**, copy the "lib" folder from the "foxitpdfsdk_11_0_mac_go" folder to the project "test" folder.

- 4) Set the library path referring to [Set the library path to the environment variable](#). Just replace the path of the library.
- 5) Add the following Go file "test.go" to the folder "test".

Note:

- Set the value of "sn" in test.go with the string after "SN=" from "gsdk_sn.txt".
- Set the value of "key" in test.go with the string after "Sign=" from "gsdk_key.txt".

```
import (
```

```
"fmt"
. "foxit.com/fsdk"
)
# Assuming PDFDoc doc has been loaded.

# The value of "sn" can be got from "gsdk_sn.txt" (the string after "SN=").
# The value of "key" can be got from "gsdk_key.txt" (the string after "Sign=").
const (
sn = " "
key = " "
)
func main() int {
code := LibraryInitialize(sn, key)
if code != E_ErrSuccess{
return 0
}
# Load a PDF document, and parse the first page of the document.
doc := NewPDFDoc("SamplePDF.pdf")
defer DeletePDFDoc(doc)
error_code := doc.Load("")
if error_code != E_ErrSuccess:
return 0
page := doc.GetPage(0)
page.StartParse(PDFPageE_ParsePageNormal)

width := int(page.GetWidth())
height := int(page.GetHeight())
matrix := page.GetDisplayMatrix(0, 0, width, height, page.GetRotation())

# Prepare a bitmap for rendering.
bitmap := NewBitmap(width, height, BitmapE_DIBArgb)
bitmap.FillRect(0xFFFFFFFF)
# Render page.
render := Renderer(bitmap, false)
render.StartRender(page, matrix)

# Add the bitmap to image and save the image.
img := NewImage()
img.AddFrame(bitmap)
img.SaveAs("testpage.jpg")
```

```
LibraryRelease()  
    return 0  
}
```

- 6) Initialize the Go module.

In the root directory of the "test" project, open a terminal and run the following command:

```
go mod init test
```

- 7) Edit the Go module.

Using Linux x64 as an example:

```
go mod edit -replace foxit.com/fsdk=./lib/x64  
go mod edit -require foxit.com/fsdk@v0.0.0
```

- 8) Clean up and organize Go dependencies.

```
go mod tidy
```

- 9) Run the project.

```
go run test.go
```

Then, the "testpage.jpg" will be generated in the current folder.

3 WORKING WITH SDK API

In this section, we will introduce a set of major features and list some examples for each feature to show you how to integrate powerful PDF capabilities with your applications using Foxit PDF SDK Go API.

3.1 Initialize Library

It is necessary for applications to initialize Foxit PDF SDK before calling any APIs. The function [LibraryInitialize](#) is provided to initialize Foxit PDF SDK. A license should be purchased for the application and pass unlock key and code to get proper support. When there is no need to use Foxit PDF SDK any more, please call function [LibraryRelease](#) 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

```
package main
import (
    . "foxit.com/fsdk"
)
// Define serial number and key
const (
    sn = ""
    key = ""
)
.....
errCode := LibraryInitialize(sn, key)
if errCode != E_ErrSuccess {
    return false
}
```

3.2 Document

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

Example:

3.2.1 How to create a PDF document from scratch

```
package main
import (
    . "foxit.com/fsdk"
)
...
doc := NewPDFDoc()
defer DeletePDFDoc(doc)
```

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

3.2.2 How to load an existing PDF document from file path

```
package main
import (
    . "foxit.com/fsdk"
)
...
doc := NewPDFDoc("Sample.pdf")
defer DeletePDFDoc(doc)
error_code := doc.Load("")
if error_code != E_ErrSuccess{
    return 0
}
```

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

```
package main
import (
    . "foxit.com/fsdk"
    "io/ioutil"
    "unsafe"
)
...
fileData, err := ioutil.ReadFile("blank.pdf")
if err != nil {
    return 0
}
doc := NewPDFDoc(uintptr(unsafe.Pointer(&fileData[0])), int64(len(fileData)))
defer DeletePDFDoc(doc)
error_code := doc.Load()
if error_code != E_ErrSuccess {
    return 0
}
```

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

```
package main
```

```
import (
    . "foxit.com/fsdk"
    "os"
    "unsafe"
)

...
// FileReaderImpl implements the FileReaderCallback interface
type FileReaderImpl struct {
    file    *os.File
    filePath string
    FileReaderCallback
}

// NewFileReader creates a new FileReaderImpl instance
func NewFileReader() *FileReaderImpl {
    return &FileReaderImpl{}
}

// LoadFile loads a file
func (fr *FileReaderImpl) LoadFile(filePath string) bool {

    fr.filePath = filePath

    file, err := os.Open(filePath)
    if err != nil {
        return false
    }
    fr.file = file
    return true
}

// GetSize returns the size of the file
func (fr FileReaderImpl) GetSize() int64 {
    fileInfo, err := os.Stat(fr.filePath)
    if err != nil {
        return 0
    }
    return fileInfo.Size()
}

// ReadBlock reads a block of data from the file
func (fr FileReaderImpl) ReadBlock(buffer uintptr, offset int64, size int64) bool {
    // First, move the file pointer to the specified offset
    _, err := fr.file.Seek(offset, 0)
    if err != nil {
        return false
    }

    // Create a temporary buffer to hold the data to be read
    tempBuffer := make([]byte, size)
```

```
// Read the specified amount of data from the file
n, err := fr.file.Read(tempBuffer)
if err != nil || int64(n) != size {
    return false
}

for i := 0; i < n; i++ {
    *(*byte)(unsafe.Pointer(buffer + uintptr(i))) = tempBuffer[i]
}
return true
}

// Release releases resources
func (fr FileReaderImpl) Release() {
    if fr.file != nil {
        fr.file.Close()
        fr.file = nil
    }
}

...
input_pdf_path := "Sample.pdf"
fileReader := FileReaderImpl {}
if !fileReader.LoadFile(fileName) {
    return false
}

// Create callback adapter for fileReader
fileReader_ := NewDirectorFileReaderCallback(fileReader)

doc := NewPDFDoc(SwgcptrFileReaderCallback(fileReader_.Swgcptr()))
defer DeletePDFDoc(doc)
error_code := doc.Load()
if error_code != E_ErrSuccess{
    return 0
}
```

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

```
package main
import (
    . "foxit.com/fsdk"
)

...
doc := NewPDFDoc("Sample.pdf")
defer DeletePDFDoc(doc)
error_code := doc.Load("")
if error_code != E_ErrSuccess{
    return 0
}

page := doc.GetPage(0)
page.StartParse(uint(PDFPageE_ParsePageNormal))
```

3.2.6 How to save a PDF to a file

```
package main
import (
    . "foxit.com/fsdk"
)

...
doc := NewPDFDoc("Sample.pdf")
defer DeletePDFDoc(doc)
error_code := doc.Load("")
if error_code != E_ErrSuccess {
    return 0
}
doc.SaveAs("new_Sample.pdf", uint(PDFDocE_SaveFlagNoOriginal))
```

3.2.7 How to save a document into memory buffer by FileWriterCallback

```
package main
import (
    . "foxit.com/fsdk"
    "os"
    "unsafe"
)

// FileWriterImpl implements the FileWriterCallback interface
type FileWriterImpl struct {
    file *os.File
    FileWriterCallback
}

// NewFileWriter creates a new FileWriterImpl instance
func NewFileWriter() *FileWriterImpl {
    return &FileWriterImpl{}
}

// LoadFile loads a file for writing
func (fw *FileWriterImpl) LoadFile(filePath string) bool {
    file, err := os.Create(filePath)
    if err != nil {
        return false
    }
    fw.file = file
    return true
}

// WriteBlock writes a block of data to the file
func (fw FileWriterImpl) WriteBlock(buffer uintptr, offset int64, size int64) bool {
    if fw.file == nil {
        return false
    }
}
```

```
// First, seek to the specified offset in the file
_, err := fw.file.Seek(offset, 0)
if err != nil {
    return false
}

// Create a temporary buffer to hold the data from the memory pointer
tempBuffer := make([]byte, size)

// Copy data from the memory pointer to the temporary buffer
for i := int64(0); i < size; i++ {
    tempBuffer[i] = *(*byte)(unsafe.Pointer(buffer + uintptr(i)))
}

// Write the temporary buffer to the file
n, err := fw.file.Write(tempBuffer)
if err != nil || int64(n) != size {
    return false
}

return true
}

// GetSize returns the current size of the file
func (fw FileWriterImpl) GetSize() int64 {
    if fw.file == nil {
        return 0
    }
    currentPos, _ := fw.file.Seek(0, os.SEEK_CUR)
    size, _ := fw.file.Seek(0, os.SEEK_END)
    fw.file.Seek(currentPos, os.SEEK_SET)
    return size
}

// Flush flushes the file buffer
func (fw FileWriterImpl) Flush() bool {
    if fw.file == nil {
        return false
    }
    return fw.file.Sync() == nil
}

// Release releases resources
func (fw FileWriterImpl) Release() {
    if fw.file != nil {
        fw.file.Close()
        fw.file = nil
    }
}

fileWriter := FileWriterImpl{}
```

```
if !fileWriter.LoadFile(outputFile) {
    fmt.Printf("Failed to create output file: %s\n", outputFile)
    return
}
fileWriter_ := NewDirectorFileWriterCallback(fileWriter)

# Assuming PDFDoc doc has been loaded.
doc.StartSaveAs(fileWriter_, uint(PDFDocE_SaveFlagNoOriginal))
...
```

3.3 Page

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

Example:

3.3.1 How to get page size

```
package main
import (
    . "foxit.com/fsdk"
)

...

# Assuming PDFPage page has been loaded and parsed.
width := int(page.GetWidth())
height := int(page.GetHeight())
```

3.3.2 How to calculate bounding box of page contents

```
package main
import (
    . "foxit.com/fsdk"
)

...

# Assuming PDFDoc doc has been loaded.
# Assuming PDFPage page has been loaded and parsed.
ret := page.CalcContentBBox(PDFPageE_CalcContentsBox)
...
```

3.3.3 How to create a PDF page and set the size

```
package main
import (
```

```
. "foxit.com/fsdk"
)
...
# Assuming PDFDoc doc has been loaded.
page := doc.InsertPage(index, PageWidth, PageHeight)
```

3.3.4 How to delete a PDF page

```
package main
import (
. "foxit.com/fsdk"
)
...
# Assuming PDFDoc doc has been loaded.

# Remove a PDF page by page index.
doc.RemovePage(index)

# Remove a specified PDF page.
doc.RemovePage(page)
...
```

3.3.5 How to flatten a PDF page

```
package main
import (
. "foxit.com/fsdk"
)

page := NewPDFPage()
defer DeletePDFPage(page)
...
# Assuming PDFPage page has been loaded and parsed.
# Flatten all contents of a PDF page.
page.Flatten(true, uint(PDFPageE_FlattenAll))

# Flatten a PDF page without annotations.
page.Flatten(true, uint(PDFPageE_FlattenNoAnnot))

# Flatten a PDF page without form controls.
page.Flatten(true, uint(PDFPageE_FlattenNoFormControl))

# Flatten a PDF page without annotations and form controls (Equals to nothing to be flattened).
page.Flatten(true, uint(PDFPageE_FlattenNoAnnot | PDFPageE_FlattenNoFormControl))
...
```

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

```
package main
import (
. "foxit.com/fsdk"
)
```

```

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

bmp := NewBitmap()
defer DeleteBitmap(bmp)

# Write bitmap data to the bmp object.
...
# Set thumbnails to the page.
page.SetThumbnail(bmp)
# Load thumbnails in the page.
bitmap := page.LoadThumbnail()
...

```

3.4 Render

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

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

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

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

Example:

3.4.1 How to render a page to a bitmap

```

package main
import (
    "foxit.com/fsdk"

```



```
)  
  
# Assuming PDFPage page has been loaded and parsed.  
  
width := int(page.GetWidth())  
height := int(page.GetHeight())  
matrix := page.GetDisplayMatrix(0, 0, width, height, page.GetRotation())  
  
# Prepare a bitmap for rendering.  
bitmap := NewBitmap(width, height, BitmapE_DIBArgb)  
defer DeleteBitmap(bitmap)  
  
bitmap.FillRect(uint(0xFFFFFFFF))  
# Render page.  
render := NewRenderer(bitmap, false)  
render.StartRender(page, matrix)  
...
```

3.4.2 How to render page and annotation

```
package main  
import (  
    . "foxit.com/fsdk"  
)  
  
# Assuming PDFPage page has been loaded and parsed.  
  
width := int(page.GetWidth())  
height := int(page.GetHeight())  
matrix := page.GetDisplayMatrix(0, 0, width, height, page.GetRotation())  
  
# Prepare a bitmap for rendering.  
bitmap := NewBitmap(width, height, BitmapE_DIBArgb)  
defer DeleteBitmap(bitmap)  
bitmap.FillRect(uint(0xFFFFFFFF))  
  
render := NewRenderer(bitmap, false)  
defer DeleteRenderer(render)  
dwRenderFlag := RendererE_RenderAnnot | RendererE_RenderPage  
render.SetRenderContentFlags(uint(dwRenderFlag))  
render.StartRender(page, matrix)  
...
```

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

```
package main
import (
    . "foxit.com/fsdk"
)

# Assuming PDFDoc doc has been loaded.

# Get information of attachments.
attachments := NewAttachments(doc)
defer DeleteAttachments(attachments)

count := attachments.GetCount()
for i := 0; i < count ; i++){
    key := attachments.GetKey(i)
    file_spec := attachments.GetEmbeddedFile(key)
    if !file_spec.IsEmpty(){
        name := file_spec.GetFileName()
    }
    if file_spec.IsEmbedded(){
        exFilePath := "output_directory"
        file_spec.ExportToFile(exFilePath)
    }
}
...
```

3.5.2 How to remove all the attachments of a PDF

```
package main
import (
    . "foxit.com/fsdk"
)

# Assuming PDFDoc doc has been loaded.

# Get information of attachments.
attachments :=NewAttachments(doc)
defer DeleteAttachments(attachments)
count = attachments.GetCount()
for i := 0; i < count ; i++){
    key := attachments.GetKey(i)
    attachments.RemoveEmbeddedFile(key)
}
...
```

3.6 Text Page

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

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

Example:

3.6.1 How to extract text from a PDF page

```
package main
import (
    . "foxit.com/fsdk"
    "os"
)
...

# Assuming PDFPage page has been loaded and parsed.

# Get the text page object.
text_page := NewTextPage(page)
defer DeleteTextPage(text_page)
count := text_page.GetCharCount()
if count > 0{
    text = text_page.GetChars()
    file.WriteString(text)
}
...
```

3.6.2 How to get the text within a rectangle area in a PDF

```
package main
import (
    . "foxit.com/fsdk"
    "os"
)
...

rect := NewRectF()
rect.SetLeft(90)
rect.SetRight(450)
```

```
rect.SetTop(595)
rect.SetBottom(580)
textPage := NewTextPage(page, int(TextPageE_ParseTextNormal))
textPage.GetTextInRect(rect)
...
```

3.7 Text Search

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

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

Example:

3.7.1 How to search a text pattern in a PDF

```
package main
import (
    . "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.

# Search for all pages of doc.
search := NewTextSearch(doc)
defer DeleteTextSearch(search)

start_index := 0
end_index := doc.GetPageCount() - 1
search.SetStartPage(start_index)
search.SetEndPage(end_index)

pattern := "Foxit"
search.SetPattern(pattern)

flags := TextSearchE_SearchNormal
search.SetSearchFlags(uint(flags))
...
match_count := 0
for search.FindNext() {
    rect_array := search.GetMatchRects()
```

```
match_count = match_count + 1
}
...
```

3.8 Search and Replace

The Search and Replace feature allows you to search for specific text content within a PDF document and replace it with new content.

3.8.1 System requirements

Platform: Windows, Linux, Mac

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

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

SDK Version: Foxit PDF SDK (C, C++, C#, Java, Python, Objective-C) 9.0 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.8.2 How to work with the search and replace function

```
package main
import (
    . "foxit.com/fsdk"
)

doc := NewPDFDoc(input_file)
defer DeletePDFDoc(doc)
error_code := doc.Load("")

# Instantiate a TextSearchReplace object.
searchreplace := NewTextSearchReplace(doc)
defer DeleteTextSearchReplace(searchreplace)

# Configure search options, match whole words only, whether to set match only whole words and match case.
find_option := NewFindOption(true, true)
defer DeleteFindOption(find_option)

# Set replacing callback function.
callback := ReplaceCallbackImpl{ }
callback_ := NewDirectorReplaceCallback(callback)
searchreplace.SetReplaceCallback(callback_)

# Set keywords and page index to do searching and replacing.
searchreplace.SetPattern(pattern, 0, find_option)

# Replace with new text.
for searchreplace.ReplaceNext("PDC") == true{
```

3.9 Text Link

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

Example:

3.9.1 How to retrieve hyperlinks in a PDF page

```
package main
import (
    . "foxit.com/fsdk"
)

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

# Get the text page object.
text_page := NewTextPage(page)
defer DeleteTextPage(text_page)
pageTextLink := NewPageTextLinks(text_page)
defer DeletePageTextLinks(pageTextLink)
textLink := pageTextLink.GetTextLink(index)
strURL := textLink.GetURI()
...
```

3.10 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 [PDFDoc.GetRootBookmark](#) must be called first to get the root of the whole bookmark tree before accessing to the bookmark tree. Here, "root bookmark" is an abstract object which can only have some child bookmarks without next sibling bookmarks and any data (includes bookmark data, destination data and action data). It cannot be shown on the application UI since it has no data. Therefore, a root bookmark can only call function [Bookmark.GetFirstChild](#).

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

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

Example:

3.10.1 How to find and list all bookmarks of a PDF

```
package main
import (
    . "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.
root := doc.GetRootBookmark()
first_bookmark := root.GetFirstChild()

func TraverseBookmark(root Bookmark, iLevel int){
    if !root.IsEmpty():
        child := root.GetFirstChild()
        for !child.IsEmpty(){
            TraverseBookmark(child, iLevel + 1)
            child = child.GetNextSibling()
        }
}

if first_bookmark != nil{
    TraverseBookmark(first_bookmark, 0)
}

...
```

3.10.2 How to insert a new bookmark

```
package main
import (
    . "foxit.com/fsdk"
    "fmt"
)

# Assuming PDFDoc doc has been loaded.
root := doc.GetRootBookmark()
if root.IsEmpty(){
    root = doc.CreateRootBookmark()
}

dest := DestinationCreateFitPage(doc, 0)
ws_title := fmt.Sprintf("A bookmark to a page (index: %d)", 0)
child := root.Insert(ws_title, BookmarkE_PosLastChild)
child.SetDestination(dest)
child.SetColor(uint(0xF68C21))
.....
DeleteDestination(dest)
```

3.10.3 How to create a table of contents based on bookmark information in PDFs

```
package main
```

```

import (
    . "foxit.com/fsdk"
)
...
func AddTOCToPDF(doc PDFDoc){
    # Set the table of contents configuration.
    intarray := NewInt32Array()
    depth := doc.GetBookmarkLevelDepth()
    if depth > 0{
        for i := 1; i < depth; i++{
            intarray.Add(i)
        }
    }
    title := ""
    toc_config := NewTableOfContentsConfig(title, intarray, true, false)
    defer DeleteTableOfContentsConfig(toc_config)
    # Add the table of contents
    doc.AddTableOfContents(toc_config)
}

```

3.11 Form (AcroForm)

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

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

To import form data from a FDF/XFDF file or export such data to a FDF/XFDF file, please refer to functions [PDFDoc.ImportFromFDF](#) and [PDFDoc.ExportToFDF](#).

Example:

3.11.1 How to load the forms in a PDF

```

import (
    . "foxit.com/fsdk"
)
...
# Assuming PDFDoc doc has been loaded.

```



```
hasForm := doc.HasForm()
if hasForm{
  form := NewForm(doc)
  defer DeleteForm(form)
}
...
```

3.11.2 How to count form fields and get/set the properties

```
import (
  "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.

form := NewForm(doc)
defer DeleteForm(form)
countFields := form.GetFieldCount("")
for i:=0;i< countFields;i++){
  field := form.GetField(i, filter)
  type := field.GetType()
  org_alternateName := field.GetAlternateName()
  field.SetAlternateName("signature")
}
```

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

```
import (
  "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.
form := NewForm(doc)
defer DeleteForm(form)

...
form.ExportToXML(XMLFilePath)
```

3.11.4 How to import form data from a XML file

```
import (
  "foxit.com/fsdk"
)

# Assuming PDFDoc doc has been loaded.
form := NewForm(doc)
defer DeleteForm(form)

...
form.ImportFromXML(XMLFilePath)
```

3.11.5 How to get coordinates of a form field

1. Load PDF file by [PDFDoc](#).
2. Traverse the [form fields](#) of the [PDFDoc](#) to get the [field](#) object of [form](#).
3. Traverse the [form controls](#) of the [field](#) object to get the [form control](#) object.
4. Get the related [widget annotation](#) object by [form control](#).
5. Call the [GetRect](#) of the [widget annotation](#) object to get the coordinate of the form.

```
import (
    "foxit.com/fsdk"
    "fmt"
)
...

# Load a document
doc := NewPDFDoc(input_file)
defer DeletePDFDoc(doc)
error_code := doc.Load("")
if error_code != E_ErrSuccess{
    fmt.Printf("The PDFDoc %s Error: %d", input_file, error_code)
    return 1
}
if !doc.HasForm(){return 1}
form := NewForm(doc)
defer DeleteForm(form)
for i:=0; i< form.GetFieldCount("");i++){
    field := form.GetField(i, "")
    if field.IsEmpty(){
        continue
    }
    for j:=0;j< field.GetControlCount();j++ {
        control := field.GetControl(j)
        widget := control.GetWidget()
        # Get rectangle of the annot widget.
        rect := widget.GetRect()
    }
}
...
```

3.12 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 users 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 [AppProviderCallback](#) and [DocProviderCallback](#) to represent the callback objects as an XFA document provider and an XFA application provider respectively. All the functions in those classes are used as callback functions. Pure virtual functions should be implemented by users.
- To use the XFA form feature, please make sure the license key has the permission of the 'XFA' module.

Example:

3.12.1 How to load XFADoc and represent an Interactive XFA form

```
import (
    . "foxit.com/fsdk"
)

pXFAAppHandler := XFAAppHandler{}
pXFAAppHandler_ := NewDirectorAppProviderCallback(pXFAAppHandler)

# implement from AppProviderCallback
Library.RegisterXFAAppProviderCallback(pXFAAppHandler_)
input_file := input_path + "xfa_dynamic.pdf"
doc := NewPDFDoc(input_file)
defer DeletePDFDoc(doc)
error_code := doc.Load("")
if error_code != fdsk.E_ErrSuccess {
    return 1
}

pXFADocHandler := XFADocHandler{}
pXFADocHandler_ := NewDirectorDocProviderCallback(pXFADocHandler)
# implement from DocProviderCallback
xfa_doc = NewXFADoc(doc, pXFADocHandler_)
defer DeleteXFADoc(xfa_doc)
xfa_doc.StartLoad("")
...
```

3.12.2 How to export and import XFA form data

```
import (
    . "foxit.com/fsdk"
)

# Assuming FSXFADoc xfa_doc has been loaded.

xfa_doc.ExportData("xfa_form.xml", XFADocE_ExportDataTypeXML)

xfa_doc.ResetForm()
```

```
doc.SaveAs("xfa_dynamic_resetform.pdf")

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

3.13 Form Filler

Form filler is the most commonly used feature for users. Form filler allows applications to fill forms dynamically. The key point for applications to fill forms is to construct some callback functions for PDF SDK to call. To fill the form, please construct a **Filler** object by current **Form** object or retrieve the **Filler** object by function **Form.GetFormFiller** if such object has been constructed. (There should be only one form filler object for an interactive form).

3.14 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:

3.14.1 How to add a text form field to a PDF

```
import (
    . "foxit.com/fsdk"
)

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

# Add text field
control := form.AddControl(page, "Text Field0", FieldE_TypeTextField, NewRectF(float32(50), float32(600),
float32(90), float32(640)))
control.GetField().SetValue("3")
# Update text field's appearance.
control.GetWidget().ResetAppearanceStream()

control1 = form.AddControl(page, "Text Field1", FieldE_TypeTextField, NewRectF(float32(100), float32(600),
float32(140), float32(640)))
control1.GetField().SetValue("123")
# Update text field's appearance.
control1.GetWidget().ResetAppearanceStream()
...
```

3.14.2 How to remove a text form field from a PDF

```
import (
    . "foxit.com/fsdk"
)
...
# Assuming PDFDoc doc has been loaded.

form := NewForm(doc)
defer DeleteForm(form)
filter := "text1"
countFields := form.GetFieldCount("")
for i:=0; i<countFields; i++){
    field := form.GetField(i, filter)
    if field.GetType() == FieldE_TypeTextField{
        form.RemoveField(field)
    }
}
...
```

3.15 Annotations

3.15.1 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:

3.15.1.1 How to add a link annotation to a PDF page

```
import (
    . "foxit.com/fsdk"
)

...
# Assuming PDFPage page has been loaded and parsed.
# Assuming the annots in the page have been loaded.
```

```
# Add link annotation.
link := NewLink(page.AddAnnot(AnnotE_Link, NewRectF(float32(350),float32(350),float32(380),float32(400))))
defer DeleteLink(link)
link.SetHighlightingMode(AnnotE_HighlightingToggle)
```

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

```
import (
    . "foxit.com/fsdk"
)
...
# Assuming PDFPage page has been loaded and parsed.
# Assuming the annots in the page have been loaded.

# Add highlight annotation.
highlight := NewHighlight(page.AddAnnot(AnnotE_Highlight,NewRectF(10,450,100,550)))
defer DeleteHighlight(highlight)
highlight.SetContent("Highlight")
quad_points := NewQuadPoints()
quadPoints.SetFirst(NewPointF(float32(10), float32(500)))
quadPoints.SetSecond(NewPointF(float32(90), float32(500)))
quadPoints.SetThird(NewPointF(float32(10), float32(480)))
quadPoints.SetFourth(NewPointF(float32(90), float32(480)))
quad_points_array := NewQuadPointsArray()
quad_points_array.Add(quad_points)
highlight.SetQuadPoints(quad_points_array)
highlight.SetSubject("Highlight")
highlight.SetTitle("Foxit SDK")
highlight.SetCreationDateTime(GetLocalDateTime())
highlight.SetModifiedDateTime(GetLocalDateTime())
highlight.SetUniqueID(RandomUID())

# Appearance should be reset.
highlight.ResetAppearanceStream()
...
```

3.15.1.3 How to set the popup information when creating markup annotations

```
import (
    . "foxit.com/fsdk"
)
...
# Assuming PDFPage page has been loaded and parsed.
# Assuming the annots in the page have been loaded.

# Create a new note annot and set the properties for it.
note := NewNote(page.AddAnnot(AnnotE_Note, NewRectF(float32(10), float32(350), float32(50), float32(400))))

note.SetIconName("Comment")
note.SetSubject("Note")
note.SetTitle("Foxit SDK")
note.SetContent("Note annotation")
```

```
note.SetCreationDateTime(GetLocalDateTime())
note.SetModifiedDateTime(GetLocalDateTime())
note.SetUniqueID(RandomUID())

# Create a new popup annot and set it to the new note annot.
popup := NewPopup(page.AddAnnot(AnnotE_Popup, NewRectF(float32(300), float32(450), float32(500),
float32(550))))

popup.SetBorderColor(0x00FF00)
popup.SetOpenStatus(false)
popup.SetModifiedDateTime(GetLocalDateTime())
note.SetPopup(popup)
```

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

```
import (
. "foxit.com/fsdk"
)

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

width := int(page.GetWidth())
height := int(page.GetHeight())

# Get page transformation matrix.
displayMatrix:= page.GetDisplayMatrix(0, 0, width, height, page.GetRotation())
iAnnotCount := page.GetAnnotCount()

for i :=0; i< iAnnotCount; i++){
    pAnnot := page.GetAnnot(i)
    if AnnotE_Popup == pAnnot.GetType(){
        continue
    }
    annotRect := pAnnot.GetDeviceRect(False, displayMatrix)
    pt := NewPointF()
    tolerance := 1

    # Get the same annot (pAnnot) using annotRect.
    pt.SetX(float32(annotRect.GetLeft() + tolerance))
    pt.SetY(float32( (annotRect.GetTop() - annotRect.GetBottom())/2 + annotRect.GetBottom()))
    gAnnot := page.GetAnnotAtDevicePoint(pt, float32(tolerance), displayMatrix)
}

...
```

3.15.1.5 How to extract the texts under text markup annotations

```
import (
. "foxit.com/fsdk"
)
```



```
...
# Assuming PDFDoc doc has been loaded.
...

page := doc.GetPage(0)
# Parse the first page.
page.StartParse(PDFPageE_ParsePageNormal)
annot_count := page.GetAnnotCount()
text_page := NewTextPage(page)
for i := 0; i < annot_count; i++ {
    annot := page.GetAnnot(i)
    text_markup := NewTextMarkup(annot)
    if !text_markup.IsEmpty() {
        # Get the texts which intersect with a text markup annotation.
        text := text_page.GetTextUnderAnnot(text_markup)
    }
}
```

3.15.1.6 How to add richtext for freetext annotation

```
import (
    . "foxit.com/fsdk"
)

...
# Make sure that SDK has already been initialized successfully.
# Load a PDF document, get a PDF page and parse it.

# Add a new freetext annotation, as text box.
freetext := NewFreeText(pdf_page.AddAnnot(AnnotE_FreeText, NewRectF(50, 50, 150, 100)))
# Set annotation's properties.

# Add/insert richtext string with style.
richtext_style := NewRichTextStyle()
richtext_style.SetFont(NewFont("Times New Roman", 0, FontE_CharsetANSI, 0))
richtext_style.SetText_color(0xFF0000)
richtext_style.SetText_size(10)
freetext.AddRichText("Textbox annotation ", richtext_style)

richtext_style.SetText_color(0x00FF00)
richtext_style.SetIs_underline(true)
freetext.AddRichText("1-underline", richtext_style)

richtext_style.SetFont(NewFont("Calibri", 0, FontE_CharsetANSI, 0))
richtext_style.SetText_color(0x0000FF)
richtext_style.SetIs_underline(false)
richtext_style.SetIs_strikethrough(true)
richtext_count := freetext.GetRichTextCount()
freetext.InsertRichText(richtext_count - 1, "2_strikethrough", richtext_style)

# Appearance should be reset.
freetext.ResetAppearanceStream()
```

3.15.2 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:

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

```
import (
    . "foxit.com/fsdk"
    "io/ioutil"
    "unsafe"
)

buffer, err := ioutil.ReadFile(fdfFile)
bufferPtr := uintptr(unsafe.Pointer(&buffer[0]))

fdf_doc := NewFDFDoc(bufferPtr, int64(len(buffer)))
pdf_doc.ImportFromFDF(fdf_doc, PDFDocE_Annots)
```

3.16 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:

3.16.1 How to convert PDF pages to bitmap files

```
import (
    . "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.
...

# Get page count
nPageCount := doc.GetPageCount()
for i := 0; i < nPageCount; i++ {
    page := doc.GetPage(i)
}

# Parse page.
page.StartParse(uint(PDFPageE_ParsePageNormal))
```

```
width := int(page.GetWidth())
height := int(page.GetHeight())
matrix := page.GetDisplayMatrix(0, 0, width, height, page.GetRotation())

# Prepare a bitmap for rendering.
bitmap := NewBitmap(width, height, BitmapE_DIBArgb)
bitmap.FillRect(0xFFFFFFFF)

# Render page.
render := NewRenderer(bitmap, false)
render.StartRender(page, matrix)
image.AddFrame(bitmap)
...
```

Note: For pdf2image functionality, if the PDF file contains images larger than 1G, it is recommended to process the images using tiled rendering. Otherwise, it may occur exceptions. Following is a brief implementation of tiled rendering.

```
import (
    . "foxit.com/fsdk"
)
...
# Parse page.
page.StartParse(PDFPageE_ParsePageNormal)

width := int(page.GetWidth())
height := int(page.GetHeight())

render_sum := 10
width_scale := 1
height_scale := 1
little_width := width * width_scale
little_height := height / render_sum * height_scale
for i := 0; i < render_sum; i++ {
    # According to Matrix, do module rendering for large PDF files.
    matrix := page.GetDisplayMatrix(0, -1 * i * little_height, little_width, height * height_scale, page.GetRotation())
    # Prepare a bitmap for rendering.
    bitmap := NewBitmap(little_width, little_height, BitmapE_DIBArgb)
    bitmap.FillRect(0xFFFFFFFF)
    render := NewRenderer(bitmap, false)
    render.StartRender(page, matrix)
    # The bitmap data will be added to the end of image file after rendering.
}
...
```

3.16.2 How to convert an image file to PDF file

```
import (
    . "foxit.com/fsdk"
)
...
```

```
image := NewImage(input_file)
count := image.GetFrameCount()

doc := NewPDFDoc()
for i := 0; i < count; i++{
    page := doc.InsertPage(i)
    page.StartParse(PDFPageE_ParsePageNormal)
    # Add image to page.
    page.AddImage(image, i, NewPointF(0, 0), page.GetWidth(), page.GetHeight(), true)
}
doc.SaveAs(output_file, PDFDocE_SaveFlagNoOriginal)
...
```

3.17 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:

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

```
import (
    . "foxit.com/fsdk"
)
...

# Assuming PDFDoc doc has been loaded.

settings := NewWatermarkSettings()
settings.SetFlags(WatermarkSettingsE_FlagASPageContents | WatermarkSettingsE_FlagOnTop)
settings.SetOffset_x(0)
settings.SetOffset_y(0)
settings.SetOpacity(90)
settings.SetPosition(E_PosTopRight)
settings.SetRotation(-45.0)
settings.SetScale_x(1.0)
settings.SetScale_y(1.0)

text_properties := NewWatermarkTextProperties()
text_properties.SetAlignment(E_AlignmentCenter)
text_properties.SetColor(0xF68C21)
text_properties.SetFont_style(WatermarkTextPropertiesE_FontStyleNormal)
text_properties.SetLine_space(1)
text_properties.SetFont_size(12.0)
text_properties.SetFont(NewFont(FontE_StdIDTimesB))

watermark := NewWatermark(doc, "Foxit PDF SDK\nwww.foxitsoftware.com", text_properties, settings)
```

```
watermark.InsertToPage(doc.GetPage(0))  
  
# Save document to file  
...
```

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

```
import (  
  . "foxit.com/fsdk"  
)  
...  
  
# Assuming PDFDoc doc has been loaded.  
  
settings := NewWatermarkSettings()  
settings.flags = WatermarkSettingsE_FlagASPageContents | WatermarkSettingsE_FlagOnTop  
settings.SetOffset_x(0.0)  
settings.SetOffset_y(0.0)  
settings.SetOpacity(20)  
settings.SetPosition(E_PosCenter)  
settings.SetRotation(0.0)  
  
image := NewImage(image_file)  
bitmap := image.GetFrameBitmap(0)  
settings.SetScale_x(page.GetWidth() * 0.618 / bitmap.GetWidth())  
settings.SetScale_y(settings.GetScale_x())  
  
watermark := NewWatermark(doc, image, 0, settings)  
watermark.InsertToPage(doc.GetPage(0))  
  
# Save document to file.  
...
```

3.17.3 How to remove all watermarks from a page

```
import (  
  . "foxit.com/fsdk"  
)  
...  
# Assuming PDFPage page has been loaded and parsed.  
...  
page.RemoveAllWatermarks()  
...  
# Save document to file
```

3.18 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 PDF SDK provides applications to generate a barcode bitmap from a given string. The barcode types that Foxit PDF SDK supports are listed in Table 3-2.

Table 3-2

Barcode Type	Code39	Code128	EAN 8	UPC A	EAN13	ITF	PDF417	QR
Dimension	1D	1D	1D	1D	1D	1D	2D	2D

Example:

3.18.1 How to generate a barcode bitmap from a string

```
import (
    . "foxit.com/fsdk"
)
...

# Strings used as barcode content.
sz_code_string := "TEST-SHEET"

# Barcode format types.
code_format := BarcodeE_FormatCode39

#Format error correction level of QR code.
sz_qr_level := BarcodeE_QRCorrectionLevelLow

#Image names for the saved image files for QR code.
bmp_qr_name := "/QR_CODE_TestForBarcodeQrCode_L.bmp"

# Unit width for barcode in pixels, preferred value is 1-5 pixels.
unit_width := 2

# Unit height for barcode in pixels, preferred value is >= 20 pixels.
unit_height := 120

barcode := NewBarcode()
bitmap := barcode.GenerateBitmap(sz_code_string, code_format, unit_width, unit_height, sz_qr_level)
```

3.19 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:

3.19.1 How to encrypt a PDF file with Certificate

```
import (
    . "foxit.com/fsdk"
)
...
doc := NewPDFDoc(input_file)
error_code := doc.Load()
if error_code != E_ErrSuccess{
    return false
}

# Do encryption.
envelopes := NewStringArray()
initial_key := ""
cert_file_path := input_path + "foxit.cer"
if ! GetCertificateInfo(cert_file_path, envelopes, initial_key, true, 16){
    return false
}

handler := NewCertificateSecurityHandler()
encrypt_data := NewCertificateEncryptData(true, SecurityHandlerE_CipherAES, envelopes)
handler.Initialize(encrypt_data, initial_key)

doc.SetSecurityHandler(handler)
output_file := output_directory + "certificate_encrypt.pdf"
doc.SaveAs(output_file, PDFDocE_SaveFlagNoOriginal)
```

3.19.2 How to encrypt a PDF file with Foxit DRM

```
import (
    . "foxit.com/fsdk"
)
...
doc := NewPDFDoc(input_file)
error_code := doc.Load()
if error_code != E_ErrSuccess {
    return false
}
```

```
}  
  
# Do encryption.  
handler := NewDRMSecurityHandler()  
file_id := "Simple-DRM-file-ID"  
initialize_key := "Simple-DRM-initialize-key"  
encrypt_data := NewDRMEncryptData(true, "Simple-DRM-filter", SecurityHandlerE_CipherAES, 16, true,  
0xffffffffc)  
handler.Initialize(encrypt_data, file_id, initialize_key)  
doc.SetSecurityHandler(handler)  
  
output_file := output_directory + "foxit_drm_encrypt.pdf"  
doc.SaveAs(output_file, PDFDocE_SaveFlagNoOriginal)
```

3.20 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:

3.20.1 How to create a reflow page and render it to a bmp file

```
import (  
    . "foxit.com/fsdk"  
)  
...  
  
# Assuming PDFDoc doc has been loaded.  
  
page := doc.GetPage(0)  
# Parse PDF page.  
page.StartParse(PDFPageE_ParsePageNormal)  
  
reflow_page := NewReflowPage(page)  
  
# Set some arguments used for parsing the reflow page.  
reflow_page.SetLineSpace(0)  
reflow_page.SetZoom(100)  
reflow_page.SetParseFlags(ReflowPageE_Normal)  
  
# Parse reflow page.  
reflow_page.StartParse()  
  
# Get actual size of content of reflow page. The content size does not contain the margin.  
content_width := reflow_page.GetContentWidth()  
content_height := reflow_page.GetContentHeight()
```



```
# Assuming Bitmap bitmap has been created.

# Render reflow page.
renderer := NewRenderer(bitmap, false)
matrix := reflow_page.GetDisplayMatrix(0, 0)
renderer.StartRenderReflowPage(reflow_page, matrix)
```

3.21 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.

3.22 Pressure Sensitive Ink

Pressure Sensitive Ink (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:

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

```
import (
    . "foxit.com/fsdk"
)
...
psi := NewPSI(480, 180, True)

# Set ink diameter.
psi.SetDiameter(9)

# Set ink color.
psi.SetColor(0x434236)

# Set ink opacity.
psi.SetOpacity(0.8)
```

```
# Add points to pressure sensitive ink.  
x := 121.3043  
y := 326.6846  
pressure := 0.0966  
type := PathE_TypeMoveTo  
psi.AddPoint(NewPointF(x, y), type, pressure)
```

3.23 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:

3.23.1 How to open a document including wrapper data

```
import (  
    . "foxit.com/fsdk"  
)  
  
# file_name is PDF document which includes wrapper data.  
doc := NewPDFDoc(file_name)  
code := doc.Load()  
if code != E_ErrSuccess:  
    return false  
  
if ! doc.IsWrapper(){  
    return false  
}  
offset := doc.GetWrapperOffset()  
  
file_reader := FileReader{offset}  
file_reader.LoadFile(file_name)  
...
```

3.24 PDF Objects

There are eight types of objects 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:

3.24.1 How to remove some properties from catalog dictionary

```
import (
    . "foxit.com/fsdk"
)
...

# Assuming PDFDoc doc has been loaded.

catalog := doc.GetCatalog()
if catalog == nil{
    return
}

key_strings := []string{"Type", "Boolean", "Name", "String", "Array", "Dict"}

for _, key := range key_strings {
    if catalog.HasKey(key) {
        catalog.RemoveAt(key)
    }
}
...
```

3.25 Page Object

Page object is a feature that allows novice users having limited knowledge of PDF objects (see 3.24 for details of PDF objects) to be able to work with text, path, image, and canvas 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:

3.25.1 How to create a text object in a PDF page

```
import (
    . "foxit.com/fsdk"
)
...

# Assuming PDFPage page has been loaded and parsed.

position := page.GetLastGraphicsObjectPosition(GraphicsObjectE_TypeText)
text_object := TextObjectCreate()

text_object.SetFillColor(0xFFFF7F00)

# Prepare text state.
```

```
state := NewTextState()
state.SetFont_size(80.0)
state.SetFont(NewFont("Simsun", FontE_StylesSmallCap, FontE_CharsetGB2312, 0))
state.SetTextmode(TextStateE_ModeFill)
text_object.SetTextState(page, state, false, 750)

# Set text.
text_object.SetText("Foxit Software")
last_position := page.InsertGraphicsObject(position, text_object)
...
```

3.25.2 How to add an image logo to a PDF page

```
import (
    . "foxit.com/fsdk"
)
...
# Assuming PDFPage page has been loaded and parsed.

position := page.GetLastGraphicsObjectPosition(GraphicsObjectE_TypeImage)
image := NewImage(image_file)
image_object := ImageObjectCreate(page.GetDocument())
image_object.SetImage(image, 0)

width := float(image.GetWidth())
height := float(image.GetHeight())

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

# Please notice the matrix value.
image_object.SetMatrix(NewMatrix2D(width, 0, 0, height, (page_width - width) / 2.0, (page_height - height) / 2.0))

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

3.26 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:

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

```
import (
    . "foxit.com/fsdk"
)

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

position := page.GetFirstGraphicsObjectPosition(GraphicsObjectE_TypeText)
text_obj := page.GetGraphicsObject(position)
content := text_obj.GetMarkedContent()
item_count := content.GetItemCount()

# Get marked content property
for i := 0; i < item_count; i++{
    tag_name := content.GetItemTagName(i)
    mcid := content.GetItemMCID(i)
}
...
```

3.27 Layer

PDF Layers, in other words, Optional Content Groups (OCG), are supported in Foxit PDF SDK. Users can selectively view or hide the contents in different layers of a multi-layer PDF document. Multi-layers are widely used in many application domains such as CAD drawings, maps, layered artwork and multi-language document, etc.

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

Example:

3.27.1 How to create a PDF layer

```
import (
    . "foxit.com/fsdk"
    "fmt"
)

...
# Assuming PDFDoc doc has been loaded.

layertree := NewLayerTree(doc)
root := layertree.GetRootNode()
if root.IsEmpty(){
```

```
    fmt.Printf("No layer information!\r\n")
    return
}
...
```

3.27.2 How to set all the layer nodes information

```
import (
    . "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.

func SetAllLayerNodesInformation(layer_node LayerNode){
    if layer_node.HasLayer(){
        layer_node.SetDefaultVisible(true)
        layer_node.SetExportUsage(LayerTreeE_StateUndefined)
        layer_node.SetViewUsage(LayerTreeE_StateOFF)
        print_data = LayerPrintData("subtype_print", LayerTreeE_StateON)
        layer_node.SetPrintUsage(print_data)
        zoom_data := LayerZoomData(1, 10)
        layer_node.SetZoomUsage(zoom_data)
        new_name := "[View_OFF_Print_ON_Export_Undefined]" + layer_node.GetName()
        layer_node.SetName(new_name)
    }
    count := layer_node.GetChildrenCount()
    for i:=0;i< count;i++){
        child := layer_node.GetChild(i)
        SetAllLayerNodesInformation(child)
    }
}

layertree := NewLayerTree(doc)
root := layertree.GetRootNode()
SetAllLayerNodesInformation(root)
...
```

3.27.3 How to edit layer tree

```
import (
    . "foxit.com/fsdk"
)

...
# Assuming PDFDoc doc has been loaded.

# edit layer tree
doc := NewPDFDoc(input_file)
error_code := doc.Load("")
layertree := NewLayerTree(doc)
root := layertree.GetRootNode()
children_count := root.GetChildrenCount()
root.RemoveChild(children_count - 1)
```

```
child := root.GetChild(children_count - 2)
child0 := root.GetChild(0)
child.MoveTo(child0, 0)
child.AddChild(0, "AddedLayerNode", true)
child.AddChild(0, "AddedNode", false)
```

3.28 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:

3.28.1 How to sign the PDF document with a signature

```
import (
    . "foxit.com/fsdk"
    "fmt"
)
...
filter := "Adobe.PPKLite"
sub_filter := "adbe.pkcs7.detached"

if ! use_default{
    sub_filter = "adbe.pkcs7.sha1"
    sig_callback := SignatureCallbackImpl{sub_filter}
    sig_callback_ := NewDirectorSignatureCallback(sig_callback)
    Library.RegisterSignatureCallback(filter, sub_filter, sig_callback_)
}
fmt.Printf(
    "Use signature callback object for filter \"%s\" and sub-filter \"%s\\r\\n\",filter, sub_filter)
pdf_page := pdf_doc.GetPage(0)
# Add a new signature to first page.
new_signature := AddSignature(pdf_page, sub_filter)
```

```
# Set filter and subfilter for the new signature.
new_signature.SetFilter(filter)
new_signature.SetSubFilter(sub_filter)
is_signed := new_signature.IsSigned()
sig_state := new_signature.GetState()
fmt.Printf("[Before signing] Signed?:%t\t State:%s\n", is_signed,
    TransformSignatureStateToString(sig_state))

# Sign the new signature.
signed_pdf_path := output_directory + "signed_newssignature.pdf"
if use_default {
    signed_pdf_path = output_directory + "signed_newssignature_default_handle.pdf"
}
cert_file_path := input_path + "foxit_all.pfx"
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_file_password,
    SignatureE_DigestSHA1, signed_pdf_path)
fmt.Printf("[Sign] Finished!\n")
is_signed = new_signature.IsSigned()
sig_state = new_signature.GetState()
fmt.Printf("[After signing] Signed?:%t\tState:%s\n",is_signed,
    TransformSignatureStateToString(sig_state))

# Open the signed document and verify the newly added signature (which is the last one).
fmt.Printf("Signed PDF file: %s\n",signed_pdf_path)
signed_pdf_doc := NewPDFDoc(signed_pdf_path)
error_code := signed_pdf_doc.Load("")
if E_ErrSuccess != error_code{
    fmt.Printf("Fail to open the signed PDF file.\n")
    return
}

# Get the last signature which is just added and signed.
sig_count := signed_pdf_doc.GetSignatureCount()
signed_signature := signed_pdf_doc.GetSignature(sig_count - 1)
# Verify the integrity of signature.
signed_signature.StartVerify("")
fmt.Printf("[Verify] Finished!\n")
is_signed = signed_signature.IsSigned()
sig_state = signed_signature.GetState()
fmt.Printf("[After verifying] Signed?: %t\tState:%s\n",is_signed,
    TransformSignatureStateToString(sig_state))
```

3.28.2 How to implement signature callback function of signing

```
import (
    "foxit.com/fsdk"
    "fmt"
    "golang.org/x/crypto/pkcs12"
    "unsafe"
    "crypto"
```



```
"crypto/sha1"
"hash"
"crypto/x509"
"encoding/pem"
)
...
# Implementation of pdf.SignatureCallback
type DigestContext struct {
    fileReadCallback FileReaderCallback
    byteRangeArray []uint32
    byteRangeArraySize int
    hasher hash.Hash
    digest string
    digestBytes []byte
}
func (ctx DigestContext) GetFileReadCallback() FileReaderCallback {
    return ctx.fileReadCallback
}
func (ctx DigestContext) GetByteRangeSize() int {
    return ctx.byteRangeArraySize
}
func (ctx DigestContext) GetByteRangeElement(index int) uint32 {
    if ctx.byteRangeArray != nil && index < ctx.byteRangeArraySize {
        return ctx.byteRangeArray[index]
    }
    return 0
}
func (ctx DigestContext) HashInit() bool {
    fmt.Println("HashInit called")
    ctx.hasher = sha1.New()
    if ctx.hasher == nil {
        fmt.Println("Failed to create hasher")
    }
    return ctx.hasher != nil
}
func (ctx DigestContext) HashUpdate(fileBuffer []byte) {
    fmt.Printf("HashUpdate called with buffer length: %d\n", len(fileBuffer))
    if ctx.hasher != nil {
        fmt.Printf("Updating hash with buffer: %s\n", string(fileBuffer))
        ctx.hasher.Write(fileBuffer)
    }
}
func (ctx DigestContext) HashDigest() string {
    if ctx.hasher != nil {
        return string(ctx.hasher.Sum(nil))
    }
    return ""
}
type SignatureCallbackImpl struct {
    digestContext *DigestContext
    subFilter string
    SignatureCallback
}
```

```
}
func (sc *SignatureCallbackImpl) Release() {
}
func (sc *SignatureCallbackImpl) GetTextFromFile(buffer uintptr) bool {
    if sc.digestContext == nil || sc.digestContext.GetFileReadCallback() == nil {
        return false
    }

    fileRead := sc.digestContext.GetFileReadCallback()

    // Read first block
    start1 := sc.digestContext.GetByteRangeElement(0)
    length1 := sc.digestContext.GetByteRangeElement(1) - start1
    if !fileRead.ReadBlock(buffer, int64(start1), int64(sc.digestContext.GetByteRangeElement(1))) {
        return false
    }
    start2 := sc.digestContext.GetByteRangeElement(2)
    length2 := sc.digestContext.GetByteRangeElement(3) - start2
    if !fileRead.ReadBlock(buffer+uintptr(length1), int64(start2), int64(length2)) {
        return false
    }
    return true
}
func (sc *SignatureCallbackImpl) StartCalcDigest(file FileReaderCallback, byteRangeArray []uint32, signature
Signature, clientData uintptr) bool {
    if sc.digestContext != nil {
        sc.digestContext = nil
    }
    destSlice := make([]uint32, len(byteRangeArray))

    copy(destSlice, byteRangeArray)
    sc.digestContext = &DigestContext{
        fileReadCallback: file,
        byteRangeArray: destSlice,
        byteRangeArraySize: len(destSlice),
    }

    sc.digestContext.hasher = sha1.New()
    return true
}
func (sc *SignatureCallbackImpl) ContinueCalcDigest(clientData uintptr, pause PauseCallback)
FoxitCommonProgressive_State {
    if sc.digestContext == nil {
        return ProgressiveE_Error
    }
    fileLength := sc.digestContext.GetByteRangeElement(1) + sc.digestContext.GetByteRangeElement(3)
    fileBuffer := make([]byte, fileLength)
    if fileBuffer == nil || !sc.GetTextFromFile(uintptr(unsafe.Pointer(&fileBuffer[0]))) {
        return ProgressiveE_Error
    }

    sc.digestContext.hasher.Write(fileBuffer)
```

```
    return ProgressiveE_Finished
}
func (sc *SignatureCallbackImpl) GetDigest(clientData uintptr) string {
    digest := sc.digestContext.HashDigest()
    return digest
}
func (sc *SignatureCallbackImpl) Sign__SWIG_0(digest uintptr, digestLength uint, certPath string, password
string, digestAlgorithm FoxitPdfSignature_DigestAlgorithm, clientData uintptr) string {
    var plainText []byte
    if sc.subFilter == "adbe.pkcs7.sha1" {
        plainText = C.GoBytes(unsafe.Pointer(digest), C.int(digestLength))
    }
    fmt.Printf("certPath: %s, password: %s\n", certPath, password)

    // Load the pfx/p12 file
    pfxData, err := ioutil.ReadFile(certPath)
    if err != nil {
        fmt.Printf("Error reading certificate file: %v\n", err)
        return ""
    }

    // Parse the pfx/p12 file
    privateKey, certificate, err := pkcs12.Decode(pfxData, password)
    if err != nil {
        fmt.Printf("Error decoding PKCS12 data: %v\n", err)
        return ""
    }

    // Create a PKCS7 signature
    pkcs7, err := createPKCS7Signature(plainText, privateKey, certificate)
    if err != nil {
        fmt.Printf("Error creating PKCS7 signature: %v\n", err)
        return ""
    }

    return string(pkcs7)
}

func (sc *SignatureCallbackImpl) Sign__SWIG_1(arg2 uintptr, arg3 uint, arg4 StreamCallback, arg5 string, arg6
FoxitPdfSignature_DigestAlgorithm, arg7 uintptr) string {
    return ""
}

func createPKCS7Signature(plainText []byte, privateKey crypto.PrivateKey, certificate *x509.Certificate) ([]byte,
error) {
    signature, err := privateKey.(crypto.Signer).Sign(nil, plainText, crypto.SHA1)
    if err != nil {
        return nil, fmt.Errorf("error signing data: %v", err)
    }

    certPEM := pem.EncodeToMemory(&pem.Block{
        Type: "CERTIFICATE",
        Bytes: certificate.Raw,
```

```
    })
    result := append(certPEM, signature...)
    return result, nil
}
func (sc *SignatureCallbackImpl) VerifySigState(digest uintptr, digestLength uint, signedData uintptr,
signedDataLen uint, clientData uintptr) uint {
    if sc.digestContext == nil {
        return uint(SignatureE_StateVerifyErrorData)
    }

    if sc.subFilter != "adbe.pkcs7.sha1" {
        return uint(SignatureE_StateUnknown)
    }

    return uint(SignatureE_StateVerifyNoChange)
}
func (sc *SignatureCallbackImpl) IsNeedPadData() bool {
    return false
}
func (sc *SignatureCallbackImpl) CheckCertificateValidity(cert_path string, pw string, cert_data uintptr)
FoxitPdfSignatureCallback_CertValidity {
    return SignatureCallbackE_CertValid
}
}
```

3.29 Long term validation (LTV)

Foxit PDF SDK provides APIs to establish long term validation of signatures, which is mainly used to solve the verification problem of signatures that have already expired. LTV requires DSS (Document Security Store) which contains the verification information of the signatures, as well as DTS (Document Timestamp Signature) which belongs to the type of time stamp signature.

In order to support LTV, Foxit PDF SDK provides:

- Support for adding the signatures of time stamp type, and provides a default signature callback for the subfilter "ETSI.RFC3161".
- TimeStampServerMgr and TimeStampServer classes, which are used to set and manager the server for time stamp. The default signature callback for the subfilter "ETSI.RFC3161" will use the default time stamp server.
- LTVVerifier class which offers the functionalities of verifying signatures and adding DSS information to documents. It also provides a basic default RevocationCallback which is required by LTVVerifier.

Following lists an example about how to establish long term validation of signatures using the default signature callback for subfilter "ETSI.RFC3161" and the default RevocationCallback. For more

details, please refer to the simple demo "ltv" in the "examples/simple_demo" folder of the download package.

Example:

3.29.1 How to establish long term validation of signatures using the default signature

```
import (
    "foxit.com/fsdk"
    "fmt"
    "golang.org/x/crypto/pkcs12"
    "unsafe"
    "crypto"
    "crypto/sha1"
    "hash"
    "crypto/x509"
    "encoding/pem"
)
...
# Implementation of pdf.SignatureCallback
type DigestContext struct {
    fileReadCallback FileReadCallback
    byteRangeArray []uint32
    byteRangeArraySize int
    hasher hash.Hash
    digest string
    digestBytes []byte
}
func (ctx DigestContext) GetFileReadCallback() FileReadCallback {
    return ctx.fileReadCallback
}
func (ctx DigestContext) GetByteRangeSize() int {
    return ctx.byteRangeArraySize
}
func (ctx DigestContext) GetByteRangeElement(index int) uint32 {
    if ctx.byteRangeArray != nil && index < ctx.byteRangeArraySize {
        return ctx.byteRangeArray[index]
    }
    return 0
}
func (ctx DigestContext) HashInit() bool {
    fmt.Println("HashInit called")
    ctx.hasher = sha1.New()
    if ctx.hasher == nil {
        fmt.Println("Failed to create hasher")
    }
    return ctx.hasher != nil
}
func (ctx DigestContext) HashUpdate(fileBuffer []byte) {
    fmt.Printf("HashUpdate called with buffer length: %d\n", len(fileBuffer))
    if ctx.hasher != nil {
```

```
    fmt.Printf("Updating hash with buffer: %s\n", string(fileBuffer))
    ctx.hasher.Write(fileBuffer)
}
}
func (ctx DigestContext) HashDigest() string {
    if ctx.hasher != nil {
        return string(ctx.hasher.Sum(nil))
    }
    return ""
}
type SignatureCallbackImpl struct {
    digestContext *DigestContext
    subFilter     string
    SignatureCallback
}
func (sc *SignatureCallbackImpl) Release() {
}
func (sc *SignatureCallbackImpl) GetTextFromFile(buffer uintptr) bool {
    if sc.digestContext == nil || sc.digestContext.GetFileReadCallback() == nil {
        return false
    }

    fileRead := sc.digestContext.GetFileReadCallback()

    // Read first block
    start1 := sc.digestContext.GetByteRangeElement(0)
    length1 := sc.digestContext.GetByteRangeElement(1) - start1
    if !fileRead.ReadBlock(buffer, int64(start1), int64(sc.digestContext.GetByteRangeElement(1))) {
        return false
    }
    start2 := sc.digestContext.GetByteRangeElement(2)
    length2 := sc.digestContext.GetByteRangeElement(3) - start2
    if !fileRead.ReadBlock(buffer+uintptr(length1), int64(start2), int64(length2)) {
        return false
    }
    return true
}
func (sc *SignatureCallbackImpl) StartCalcDigest(file FileReaderCallback, byteRangeArray []uint32, signature
Signature, clientData uintptr) bool {
    if sc.digestContext != nil {
        sc.digestContext = nil
    }
    destSlice := make([]uint32, len(byteRangeArray))

    copy(destSlice, byteRangeArray)
    sc.digestContext = &DigestContext{
        fileReadCallback: file,
        byteRangeArray:   destSlice,
        byteRangeArraySize: len(destSlice),
    }

    sc.digestContext.hasher = sha1.New()
}
```

```
    return true
}
func (sc *SignatureCallbackImpl) ContinueCalcDigest(clientData uintptr, pause PauseCallback)
FoxitCommonProgressive_State {
    if sc.digestContext == nil {
        return ProgressiveE_Error
    }
    fileLength := sc.digestContext.GetByteRangeElement(1) + sc.digestContext.GetByteRangeElement(3)
    fileBuffer := make([]byte, fileLength)
    if fileBuffer == nil || !sc.GetTextFromFile(uintptr(unsafe.Pointer(&fileBuffer[0]))) {
        return ProgressiveE_Error
    }

    sc.digestContext.hasher.Write(fileBuffer)
    return ProgressiveE_Finished
}
func (sc *SignatureCallbackImpl) GetDigest(clientData uintptr) string {
    digest := sc.digestContext.HashDigest()
    return digest
}
func (sc *SignatureCallbackImpl) Sign__SWIG_0(digest uintptr, digestLength uint, certPath string, password
string, digestAlgorithm FoxitPdfSignature_DigestAlgorithm, clientData uintptr) string {
    var plainText []byte
    if sc.subFilter == "adbe.pkcs7.sha1" {
        plainText = C.GoBytes(unsafe.Pointer(digest), C.int(digestLength))
    }
    fmt.Printf("certPath: %s, password: %s\n", certPath, password)

    // Load the pfx/p12 file
    pfxData, err := ioutil.ReadFile(certPath)
    if err != nil {
        fmt.Printf("Error reading certificate file: %v\n", err)
        return ""
    }

    // Parse the pfx/p12 file
    privateKey, certificate, err := pkcs12.Decode(pfxData, password)
    if err != nil {
        fmt.Printf("Error decoding PKCS12 data: %v\n", err)
        return ""
    }

    // Create a PKCS7 signature
    pkcs7, err := createPKCS7Signature(plainText, privateKey, certificate)
    if err != nil {
        fmt.Printf("Error creating PKCS7 signature: %v\n", err)
        return ""
    }

    return string(pkcs7)
}
```

```
func (sc *SignatureCallbackImpl) Sign__SWIG_1(arg2 uintptr, arg3 uint, arg4 StreamCallback, arg5 string, arg6
FoxitPdfSignature_DigestAlgorithm, arg7 uintptr) string {
    return ""
}
func createPKCS7Signature(plainText []byte, privateKey crypto.PrivateKey, certificate *x509.Certificate) ([]byte,
error) {
    signature, err := privateKey.(crypto.Signer).Sign(nil, plainText, crypto.SHA1)
    if err != nil {
        return nil, fmt.Errorf("error signing data: %v", err)
    }

    certPEM := pem.EncodeToMemory(&pem.Block{
        Type: "CERTIFICATE",
        Bytes: certificate.Raw,
    })
    result := append(certPEM, signature...)
    return result, nil
}
func (sc *SignatureCallbackImpl) VerifySigState(digest uintptr, digestLength uint, signedData uintptr,
signedDataLen uint, clientData uintptr) uint {
    if sc.digestContext == nil {
        return uint(SignatureE_StateVerifyErrorData)
    }

    if sc.subFilter != "adbe.pkcs7.sha1" {
        return uint(SignatureE_StateUnknown)
    }

    return uint(SignatureE_StateVerifyNoChange)
}
func (sc *SignatureCallbackImpl) IsNeedPadData() bool {
    return false
}
func (sc *SignatureCallbackImpl) CheckCertificateValidity(cert_path string, pw string, cert_data uintptr)
FoxitPdfSignatureCallback_CertValidity {
    return SignatureCallbackE_CertValid
}
```

3.30 PAdES

Foxit PDF SDK also supports PAdES (PDF Advanced Electronic Signature) which is the application for CAdES signature in the field of PDF. CAdES is a new standard for advanced digital signature, its default subfilter is "ETSI.CAdES.detached". PAdES signature includes four levels: B-B, B-T, B-LT, and B-LTA.

- B-B: Must include the basic attributes.
- B-T: Must include document time stamp or signature time stamp to provide trusted time for existing signatures, based on B-B.

- B-LT: Must include DSS/VRI to provide certificates and revocation information, based on B-T.
- B-LTA: Must include the trusted time DTS for existing revocation information, based on B-LT.

Foxit PDF SDK provides a default signature callback for the subfilter "ETSI.CAdES.detached" to sign and verify the signatures (with subfilter "ETSI.CAdES.detached"). It also provides `TimeStampServerMgr` and `TimeStampServer` classes to set and manager the server for time stamp. The default signature callback for the subfilter "ETSI.CAdES.detached" will use the default time stamp server.

Foxit PDF SDK provides functions to get the level of PAdES from signature, and application level can also judge and determine the level of PAdES according to the requirements of each level. For more details about how to add, sign and verify a PAdES signature in PDF document, please refer to the simple demo "**pades**" in the "examples/simple_demo" folder of the download package.

3.31 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:

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

```
import (
    . "foxit.com/fsdk"
)
...
# Assuming PDFPage page has been loaded and parsed.
# Assuming the annots in the page have been loaded.
...

# Add link annotation
link := NewLink(page.AddAnnot(AnnotE_Link, NewRectF(350,350,380,400)))
link.SetHighlightingMode(AnnotE_HighlightingToggle)
# Add action for link annotation

action := NewURIAction(ActionCreate(page.GetDocument(), ActionE_TypeURI))
action.SetTrackPositionFlag(true)
action.SetURI("www.foxitsoftware.com")
link.SetAction(action)
# Appearance should be reset.
link.ResetAppearanceStream()
```

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

```
import (
    . "foxit.com/fsdk"
)

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

# Add link annotation
link := NewLink(page.AddAnnot(AnnotE_Link, NewRectF(350,350,380,400)))
link.SetHighlightingMode(AnnotE_HighlightingToggle)

action := ActionCreate(page.GetDocument(), ActionE_TypeGoto)
newDest := DestinationCreateXYZ(page.GetDocument(), 0,0,0,0)
action.SetDestination(newDest)
```

3.32 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 [JavaScriptAction](#) is derived from [Action](#) and offers functions to get/set JavaScript action data.

The JavaScript methods and properties supported by Foxit PDF SDK are listed in the [appendix](#).

Example:

3.32.1 How to add JavaScript Action to Document

```
import (
    . "foxit.com/fsdk"
)

...
# Load Document doc.
...

javascript_action := NewJavaScriptAction(ActionCreate(doc, ActionE_TypeJavaScript))
javascript_action.SetScript("app.alert(\"Hello Foxit \");")
additional_act := AdditionalAction(doc)
additional_act.SetAction(AdditionalActionE_TriggerDocWillClose,javascript_action)
additional_act.DoJSAction(AdditionalActionE_TriggerDocWillClose)
```

...

3.32.2 How to add JavaScript Action to Annotation

```
import (
. "foxit.com/fsdk"
)
...
# Load Document and get a widget annotation.
...

javascript_action := NewJavaScriptAction(ActionCreate(page.GetDocument(), ActionE_TypeJavaScript))
javascript_action.SetScript("app.alert(\"Hello Foxit \");")
additional_act := NewAdditionalAction(annot)
additional_act.SetAction(AdditionalActionE_TriggerAnnotMouseButtonPressed, javascript_action)
additional_act.DoJSAction(AdditionalActionE_TriggerAnnotMouseButtonPressed)
...
```

3.32.3 How to add JavaScript Action to FormField

```
import (
. "foxit.com/fsdk"
)
...
# Load Document and get a form field.
...

# Add text field
control := form.AddControl(page, "Text Field0", FieldE_TypeTextField, NewRectF(50, 600, 90, 640))
control.GetField().SetValue("3")
# Update text field's appearance.
control.GetWidget().ResetAppearanceStream()

control1 = form.AddControl(page, "Text Field1", FieldE_TypeTextField, RectF(100, 600, 140, 640))
control1.GetField().SetValue("23")
# Update text field's appearance.
control1.GetWidget().ResetAppearanceStream()

control2 := form.AddControl(page, "Text Field2", FieldE_TypeTextField, NewRectF(150, 600, 190, 640))
javascript_action := NewJavaScriptAction(ActionCreate(form.GetDocument(), ActionE_TypeJavaScript))
javascript_action.SetScript("AFSimple_Calculate(\"SUM\", new Array (\"Text Field0\", \"Text Field1\"))")
field2 := control2.GetField()
additional_act := NewAdditionalAction(field2)
additional_act.SetAction(AdditionalActionE_TriggerFieldRecalculateValue, javascript_action)
# Update text field's appearance.
control2.GetWidget().ResetAppearanceStream()
```

3.32.4 How to add a new annotation to PDF using JavaScript

```
import (
. "foxit.com/fsdk"
```

```
)
...
# Load Document and get form field, construct a Form object and a Filler object.
...

javascript_action := NewJavaScriptAction(ActionCreate(form.GetDocument(), ActionE_TypeJavaScript))
javascript_action.SetScript("var annot = this.addAnnot({ page : 0, type : \"Square\", rect : [ 0, 0, 100, 100 ], name : \"UniqueID\", author : \"A. C. Robot\", contents : \"This section needs revision.\" });")
additional_act := NewAdditionalAction(field)
additional_act.SetAction(AdditionalActionE_TriggerAnnotCursorEnter,javascript_action)
additional_act.DoJSAction(AdditionalActionE_TriggerAnnotCursorEnter)
...
```

3.32.5 How to get/set properties of annotations (strokeColor, fillColor, readOnly, rect, type) using JavaScript

```
import (
."foxit.com/fsdk"
)
...
# Load Document and get form field, construct a Form object and a Filler object.
...

# Get properties of annotations.
javascript_action := NewJavaScriptAction(ActionCreate(form.GetDocument(), ActionE_TypeJavaScript))
javascript_action.SetScript("var ann = this.getAnnot(0, \" UniqueID \"); if (ann != null) { console.println(\"Found it! type: \" + ann.type); console.println(\"readOnly: \" + ann.readOnly); console.println(\"strokeColor: \" + ann.strokeColor);console.println(\"fillColor: \" + ann.fillColor); console.println(\"rect: \" + ann.rect);}")
additional_act := NewAdditionalAction(field)
additional_act.SetAction(AdditionalActionE_TriggerAnnotCursorEnter,javascript_action)
additional_act.DoJSAction(AdditionalActionE_TriggerAnnotCursorEnter)

# Set properties of annotations (only take strokeColor as an example).
javascript_action1 := NewJavaScriptAction(Action.Create(form.GetDocument(), ActionE_TypeJavaScript))
javascript_action1.SetScript("var ann = this.getAnnot(0, \"UniqueID\");if (ann != null) { ann.strokeColor = color.blue; }")
additional_act1 := NewAdditionalAction(field1)
additional_act1.SetAction(AdditionalActionE_TriggerAnnotCursorEnter,javascript_action1)
additional_act1.DoJSAction(AdditionalActionE_TriggerAnnotCursorEnter)
...
```

3.32.6 How to destroy annotation using JavaScript

```
import (
."foxit.com/fsdk"
)
...
# Load Document and get form field, construct a Form object and a Filler object.
...

javascript_action := NewJavaScriptAction(ActionCreate(form.GetDocument(), ActionE_TypeJavaScript))
javascript_action.SetScript("var ann = this.getAnnot(0, \" UniqueID \"); if (ann != null) { ann.destroy(); } ")
```

```
additional_act := NewAdditionalAction(field)
additional_act.SetAction(AdditionalActionE_TriggerAnnotCursorEnter,javascript_action)
additional_act.DoJSAction(AdditionalActionE_TriggerAnnotCursorEnter)
...
```

3.33 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 [NewRedaction](#) to create a redaction module. If module "Redaction" is not defined in the license information which is used in function [LibraryInitialize](#), it means user has no right in using redaction related functions and this constructor will throw exception [E_ErrInvalidLicense](#).
- Then call function [Redaction.MarkRedactAnnot](#) to create a redaction object and mark page contents (text object, image object, and path object) which are to be redacted.
- Finally call function [Redaction.Apply](#) to apply redaction in marked areas: remove the text or graphics under marked areas permanently.

Note: To use the redaction feature, please make sure the license key has the permission of the 'Redaction' module.

Example:

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

```
import (
    "foxit.com/fsdk"
    "fmt"
)
...
redaction := NewRedaction(doc)
# Parse PDF page.
page := doc.GetPage(0)
page.StartParse(PDFPageE_ParsePageNormal)
text_page := NewTextPage(page)
text_search := NewTextSearch(text_page)
```

```
text_search.SetPattern("PDF")
rect_array := RectFArray()
for text_search.FindNext() {
    itemArray := text_search.GetMatchRects()
    rect_array.InsertAt(rect_array.GetSize(), itemArray)
}
if rect_array.GetSize() > 0{
    redact := redaction.MarkRedactAnnot(page, rect_array)
    redact.ResetAppearanceStream()
    doc.SaveAs(output_directory + "AboutFoxit_redected_default.pdf")

    # 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 + "AboutFoxit_redected_setColor.pdf")

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

    if redaction.Apply(){
        fmt.Printf("Redact page(0) succeed.")
    }
    else{
        fmt.Printf("Redact page(0) failed.")
    }
}
doc.SaveAs(output_directory + "AboutFoxit_redected_apply.pdf")
```

3.34 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.

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:

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

```
import (
    "foxit.com/fsdk"
)

...
base_doc := NewPDFDoc(input_base_file)
error_code := base_doc.Load("")
if error_code != E_ErrSuccess{
    fmt.Printf("The Doc [%s] Error: %d\n",input_base_file, error_code)
    return 1
}
compared_doc := NewPDFDoc(input_compared_file)
error_code = compared_doc.Load("")
if error_code != E_ErrSuccess{
    fmt.Printf("The Doc [%s] Error: %d\n",input_base_file, error_code)
    return 1
}
comparison := NewComparison(base_doc, compared_doc)
result := comparison.DoCompare(0, 0, ComparisonE_CompareTypeText)
oldInfo := result.GetBase_doc_results()
newInfo := result.GetCompared_doc_results()
oldInfoSize := oldInfo.GetSize()
newInfoSize := newInfo.GetSize()
page := compared_doc.GetPage(0)
for i :=0; i<newInfoSize;i++){
    item := newInfo.GetAt(i)
    type := item.GetType()
    if type == CompareResultInfoE_CompareResultTypeDeleteText{
        res_string := fmt.Sprintf("\'%s\'", item.GetDiff_contents())
        CreateDeleteTextStamp(page, item.GetRect_array(), 0xff0000,
            res_string, "Compare : Delete", "Text")
    }
    else if type == CompareResultInfoE_CompareResultTypeInsertText{
        res_string := fmt.Sprintf("\'%s\'", item.GetDiff_contents())
        CreateDeleteText(page, item.GetRect_array(), 0x0000ff, res_string,
            "Compare : Insert", "Text")
    }
    else if type == CompareResultInfoE_CompareResultTypeReplaceText{
        res_string := fmt.Sprintf("[New]: \\'%s\\\'\\r\\n[Old]: \\'%s\\\'",
            newInfo.GetAt(int64(i)).GetDiff_contents(), item.GetDiff_contents())
        CreateSquigglyRect(page, item.GetRect_array(), 0xe7651a, res_string,
            "Compare : Replace", "Text")
    }
}
# Save the comparison result to a PDF file.
compared_doc.SaveAs(output_directory + "result.pdf")
```

Note: for *CreateDeleteTextStamp*, *CreateDeleteText* and *CreateSquigglyRect* functions, please refer to the simple demo "**pdfcompare**" located in the "examples/simple_demo" folder of the download package.

3.35 Optimization

Optimization feature can reduce the size of PDF files to save disk space and make files easier to send and store, through compressing images, deleting redundant data, discarding useless user data and so on. Optimization module also provides functions to compress the color/grayscale/monochrome images in PDF files to reduce the size of the PDF files.

Note: To use the Optimization feature, please make sure the license key has the permission of the 'Optimization' module.

Example:

3.35.1 How to optimize PDF files by compressing the color/grayscale/monochrome images

```
import (
    . "foxit.com/fsdk"
    "fmt"
)

...
doc := NewPDFDoc(input_file)
error_code := doc.Load("input_pdf_file")
if error_code != E_ErrSuccess:
    fmt.Printf("The Doc [%s] Error: %d\n", input_file, error_code)
    return 1

pause := Optimization_Pause{0, true}
pause_ := NewDirectorPauseCallback(pause)
settings := NewOptimizerSettings()
settings.SetOptimizerOptions(OptimizerSettingsE_OptimizerCompressImages)
progressive := OptimizerOptimize(doc, settings, pause_)
progress_state := ProgressiveE_ToBeContinued
for ProgressiveE_ToBeContinued == progress_state {
    progress_state = progressive.Continue()
    percent := progressive.GetRateOfProgress()
    fmt.Printf("Optimize progress percent: %d %", percent)
    if ProgressiveE_Finished == progress_state {
        doc.SaveAs(output_directory + "ImageCompression_Optimized.pdf",
PDFDocE_SaveFlagRemoveRedundantObjects)
    }
}
fmt.Printf("Optimized Finish.")
```


3.36 HTML to PDF Conversion

For some large HTML files or a webpage which contain(s) many contents, it is not convenient to print or archive them directly. Foxit PDF SDK provides APIs to convert the online webpage or local HTML files like invoices or reports into PDF file(s), which makes them easier to print or archive. In the process of conversion from HTML to PDF, Foxit PDF SDK supports to create and add PDF Tags based on the organizational structure of HTML. In addition, Foxit PDF SDK also supports to provide the generated files after HTML2PDF conversion in the form of file stream.

For HTML to PDF module, it supports HTML5, CSS3 and JavaScript.

Foxit PDF SDK for Go API supports to convert HTML to PDF on Linux (x86/x64) and Mac x64 platforms. But for HTML to PDF engine (Linux), the version of `libnss` should be 3.22.

This section will provide instructions on how to set up your environment for running the 'html2pdf' demo.

3.36.1 System requirements

Platform: Windows, Linux (x86 and x64), Mac

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

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

SDK Version: Foxit PDF SDK (C++, Java, C#, Objective-C) 7.0 or higher; Foxit PDF SDK (C) 7.4 or higher; Foxit PDF SDK (Python) 8.3 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.36.2 HTML to PDF engine files

Please contact Foxit support team or sales team to get the HTML to PDF engine files package.

After getting the package, extract it to a desired directory (for example, extract the package to a directory: "**htmltopdf/linux**" for Linux, and "**htmltopdf/mac**" for Mac).

3.36.3 How to run the html2pdf demo

Foxit PDF SDK provides a html2pdf demo located in the "examples/simple_demo/html2pdf" folder to show how to use Foxit PDF SDK to convert from html to PDF.

3.36.3.1 Prepare a HTML2PDF engine directory

Before running the html2pdf demo, you should first extract engine package to a desired directory (for example, extract the package to a directory: "**/home/user/Desktop/htmltopdf/linux/**" for Linux), and then pass the engine file path to the API **ConvertFromHTML** to convert html to PDF file.

3.36.3.2 Configure the demo

For html2pdf demo, you can configure the demo in the "examples/simple_demo/html2pdf/html2pdf.go" file, or you can configure the demo with parameters directly in a terminal. Following will configure the demo in "html2pdf.go" file on Linux for example. For Mac platform, do the same configuration with Linux.

Specify the html2pdf engine directory

In the "html2pdf.go" file, add the path of the engine file "fxhtml2pdf.exe" as follows, which will be used to convert html files to PDF files.

```
# "engine_path" is the path of the engine file "fxhtml2pdf" which is used to convert html to pdf.  
engine_path := "/home/user/Desktop/htmltopdf/linux/fxhtml2pdf"
```

(Optional) Specify cookies file path

Add the path of the cookies file exported from the web pages that you want to convert. For example,

```
# "cookies_path" is the path of the cookies file exported from the web pages.  
cookies_path := "/home/user/Desktop/cookies.txt"
```

3.36.3.3 Run the demo

Run the demo without parameters

In a terminal, navigate to "examples/simple_demo", type `./RunDemo.sh html2pdf` to run the html2pdf demo.

Run the demo with parameters

After building the demo successfully, open a terminal, navigate to "examples/simple_demo/html2pdf", type `go run html2pdf.go --help` for example to see how to use the parameters to execute the program.

For example, convert the URL web page "www.foxit.com" into a PDF with setting the page width to 900 points and the page height to 300 points:

```
go run html2pdf.go -html www.foxit.com -w 900 -h 300
```

The output file is located in "examples/simple_demo/output_files/html2pdf" folder.

Parameters Description

Basic Syntax:

```
html2pdf_xxx <-html <The url or html path>> <-o <output pdf path>> <-engine <htmltopdf engine path>>  
[-w <page width>] [-h <page height>] [-ml <margin left>] [-mr <margin right>]
```

*[-mt <margin top>] [-mb <margin bottom>] [-r <page rotation degree>] [-mode <page mode>] [-scale <scaling mode>] [-link <whether to convert link>]
 [-tag <whether to generate tag>] [-bookmarks <whether to generate bookmarks>]
 [-print_background <whether to print background>]
 [-optimize_tag <whether to optimize tag tree>] [-media <media style>] [-encoding <HTML encoding format>] [-render_images <Whether to render images>]
 [-remove_underline_for_link <Whether to remove underline for link>]
 [-headerfooter <Whether to generate headerfooter>] [-headerfooter_title <headerfooter title>] [-headerfooter_url <headerfooter url>] [-bookmark_root_name <bookmark root name>] [-resize_objects <Whether to enable the JavaScripts related resizing of the objects>]
 [-cookies <cookies file path>] [-timeout <timeout>] [--help<Parameter usage>]*

Note:

- <> required
- [] optional

Parameters	Description
--help	The usage description of the parameters.
-html	The url or html file path. For example, '-html www.foxitsoftware.com'.
-o	The path of the output PDF file.
-engine	The path of the engine file "fxhtml2pdf.exe".
-w	The page width of the output PDF file in points.
-h	The page height of the output PDF file in points.
-r	The page rotation for the output PDF file. <ul style="list-style-type: none"> • 0 : 0 degree. • 1 : 90 degree. • 2 : 180 degree. • 3 : 270 degree.
-ml	The left margin of the pages for the output PDF file.
-mr	The right margin of the pages for the output PDF file.
-mt	The top margin of the pages for the output PDF file.
-mb	The bottom margin of the pages for the output PDF file.
-mode	The page mode for the output PDF file. <ul style="list-style-type: none"> • 0 : Single page mode. • 1 : Multiple pages mode.
-scale	The scaling mode.

Parameters	Description
	<ul style="list-style-type: none">• 0 : No need to scale pages.• 1 : Scale pages.• 2 : Enlarge page.
-link	Whether to convert links. <ul style="list-style-type: none">• 'yes' : Convert links.• 'no' : No need to convert links.
-tag	Whether to generate tag. <ul style="list-style-type: none">• 'yes' : Generate tag.• 'no' : No need to generate tag.
-bookmarks	Whether to generate bookmarks. <ul style="list-style-type: none">• 'yes' : Generate bookmarks .• 'no' : No need to generate bookmarks.
-print_background	Whether to print background. <ul style="list-style-type: none">• 'yes' : Print bookmarks .• 'no' : No need to print bookmarks.
-optimize_tag	Whether to optimize tag tree. <ul style="list-style-type: none">• 'yes' : Optimize tag tree .• 'no' : No need to optimize tag tree.
-media	The media style. <ul style="list-style-type: none">• 0 : Screen media style.• 1 : Print media style.
-encoding	The HTML encoding format. <ul style="list-style-type: none">• 0 : Auto encoding .• 1-73 : Other encodings.
-render_images	Whether to render images. <ul style="list-style-type: none">• 'yes' : Render images.• 'no' : No need to render images.
-remove_underline_for_link	Whether to remove underline for link. <ul style="list-style-type: none">• 'yes' : Remove underline for link.• 'no' : No need to remove underline for link.
-headerfooter	Whether to generate headerfooter. <ul style="list-style-type: none">• 'yes' : Generate headerfooter.• 'no' : No need to generate headerfooter.
-headerfooter_title	The headerfooter title.

Parameters	Description
-headerfooter_url	The headerfooter url.
-bookmark_root_name	The bookmark root name.
-resize_objects	Whether to enable the JavaScripts related resizing of the objects during rendering process. <ul style="list-style-type: none"> 'yes' : Enable. 'no' : Disable.
-cookies	The path of the cookies file exported from a URL that you want to convert.
-timeout	The timeout of loading webpages.

3.36.4 How to work with Html2PDF API

```
import (
    . "foxit.com/fsdk"
)
...
pdf_setting_data := NewHTML2PDFSettingData()
pdf_setting_data.SetIs_convert_link(true)
pdf_setting_data.SetIs_generate_tag(true)
pdf_setting_data.SetTo_generate_bookmarks(true)
pdf_setting_data.SetRotate_degrees(E_Rotation0)
pdf_setting_data.SetPage_height(640)
pdf_setting_data.SetPage_width(900)
pdf_setting_data.SetPage_mode(HTML2PDFSettingDataE_PageModeSinglePage)
pdf_setting_data.SetScaling_mode(HTML2PDFSettingDataE_ScalingModeScale)
pdf_setting_data.SetTo_print_background(true)
pdf_setting_data.SetTo_optimize_tag_tree(false)
pdf_setting_data.SetMedia_style(HTML2PDFSettingDataE_MediaStyleScreen)
...
ConvertFromHTML(url_or_html, engine_path, cookies_path, pdf_setting_data, output_path, time_out)
```

3.36.5 How to get HTML data from stream and convert it to a PDF file

1. Defines a [FileRead](#) class inherited from [FileReaderCallback](#) used to get html data from stream or memory. And defines a [FileWriter](#) class inherited from [FileWriterCallback](#) used to do file writing. For the implementations of [FileRead](#) and [FileWriter](#) classes, please refer to the **html2pdf** demo in the "examples/simple_demo/html2pdf" folder.
2. Get html data from stream and set resources related to source html.
3. Call the [ConvertFromHTML](#) function to convert it to a PDF file.

```
import (
    . "foxit.com/fsdk"
```

```
"fmt"
)
pdf_setting_data := NewHTML2PDFSettingData()
pdf_setting_data.SetIs_convert_link(true)
pdf_setting_data.SetIs_generate_tag(true)
pdf_setting_data.SetTo_generate_bookmarks(true)
pdf_setting_data.SetRotate_degrees(E_Rotation0)
pdf_setting_data.SetPage_height(640)
pdf_setting_data.SetPage_width(900)
pdf_setting_data.SetPage_mode(HTML2PDFSettingDataE_PageModeSinglePage)
pdf_setting_data.SetScaling_mode(HTML2PDFSettingDataE_ScalingModeScale)

pdf_setting_data := NewHTML2PDFSettingData()
pdf_setting_data.SetPage_height(650)
pdf_setting_data.SetPage_width(950)
pdf_setting_data.SetIs_to_page_scale(false)
pdf_setting_data.SetPage_margin(NewRectF(18, 18, 18, 18))
pdf_setting_data.SetIs_convert_link(true)
pdf_setting_data.SetRotate_degrees(E_Rotation0)
pdf_setting_data.SetIs_generate_tag(true)
pdf_setting_data.SetPage_mode(HTML2PDFSettingDataE_PageModeSinglePage)
pdf_setting_data.SetScaling_mode(HTML2PDFSettingDataE_ScalingModeScale)
pdf_setting_data.SetTo_generate_bookmarks(true)
pdf_setting_data.SetEncoding_format(0)
pdf_setting_data.SetTo_render_images(true)
pdf_setting_data.SetTo_remove_underline_for_link(false)
pdf_setting_data.SetTo_set_headerfooter(false)
pdf_setting_data.SetHeaderfooter_title("")
pdf_setting_data.SetHeaderfooter_url("")
pdf_setting_data.SetBookmark_root_name("abcde")
pdf_setting_data.SetTo_resize_objects(true)
pdf_setting_data.SetTo_print_background(false)
pdf_setting_data.SetTo_optimize_tag_tree(false)
pdf_setting_data.SetMedia_style(0)
pdf_setting_data.SetTo_load_active_content(false)

pdf_setting_data.SetBookmark_root_name("abcde")
pdf_setting_data.SetTo_resize_objects(true)
pdf_setting_data.SetTo_print_background(false)
pdf_setting_data.SetTo_optimize_tag_tree(false)
pdf_setting_data.SetMedia_style(0)
pdf_setting_data.SetTo_load_active_content(false)
```

```
output_path := output_directory + "html2pdf_filestream_result.pdf"
filewrite := FileWriter{}
filewrite.LoadFile(output_path)
filewrite_ := NewDirectorFileWriterCallback(filewrite)

# "htmlfile" is the path of the html file to be loaded. For example: "C:/aaa.html". The method
# of "FromHTML" will load this file as a stream.
htmlfile := ""
filereader := FileReader{}
filereader.LoadFile(htmlfile, false)
filereader_ := NewDirectorFileReaderCallback(filereader)

# "htmlfilepng" is the path of the png resource file to be loaded. For example: "C:/aaa.png". set "htmlfilepng" in
# the related_resource_file of HTML2PDFRelatedResource.
htmlfilepng := ""
filereader1 := FileReader{}
filereader1.LoadFile(htmlfilepng, false)
filereader1_ := NewDirectorFileReaderCallback(filereader1)

# "relativefilepath" is the resource file's relative path. For example: "./aaa.png".
relativefilepath := ""
html2PDFRelatedResourceArray := NewHTML2PDFRelatedResourceArray()
html2PDFRelatedResource := NewHTML2PDFRelatedResource(filereader1_, relativefilepath)
html2PDFRelatedResourceArray.Add(html2PDFRelatedResource)
ConvertFromHTML(filereader_, html2PDFRelatedResourceArray, engine_path, None, pdf_setting_data,
filewrite_, 30)
fmt.Printf("Convert HTML to PDF successfully by filestream.")
```

3.37 Office to PDF Conversion with third-party engines

Foxit PDF SDK provides APIs to convert Microsoft Office documents (Word, Excel and PowerPoint) into professional-quality PDF files on Windows and Linux (x86, x64 and armv8) platforms.

Foxit PDF SDK for Go API supports Linux x86/x64 platforms.

For using this feature, please note that:

- For Linux x86/x64, make sure that LibreOffice is already installed on your Linux system.

Note: When using LibreOffice 7.0 or a higher version, if you encounter an error like "An unknown error has occurred", you can try to set an environment variable before running the program as follows:

```
"export URE_BOOTSTRAP=vnd.sun.star.pathname:/opt/libreoffice7.x/program/fundamentalrc"
```

Where, 'x' represents the LibreOffice version.

3.37.1 System requirements

Platform: Windows, Linux (x86, x64 and armv8)

Programming Language: C, C++, Python, Java, C#, Node.js, Go

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

SDK Version: Word and Excel (Foxit PDF SDK (C++, C#, Java) 7.3 or higher; Foxit PDF SDK (C) 7.4 or higher; Foxit PDF SDK (Python) 8.3 or higher); PowerPoint (Foxit PDF SDK (C, C++, C#, Java) 7.4 or higher; Foxit PDF SDK (Python) 8.3 or higher); Word/Excel/PowerPoint (Foxit PDF SDK (Node.js) 10.0 or higher); Word/Excel/PowerPoint (Foxit PDF SDK (Go) 11.0)

Example:

Note: For Linux x86/x64, the parameter "`engine_path`" in the following sample code represents the path of LibreOffice engine. To get the installed path of LibreOffice, you can input the command "**locate soffice.bin**" in a terminal, then the path will be shown, for example, `/usr/lib/libreoffice/program/soffice.bin`. Then the value of "`engine_path`" parameter is set to `/usr/lib/libreoffice/program`.

3.37.2 How to convert Word to PDF

```
import (
    . "foxit.com/fsdk"
    "fmt"
    "runtime"
)

...
# Make sure that SDK has already been initialized successfully.

word_file_path := "test.doc"
output_path := "saved.pdf"

# Use default Word2PDFSettingData values.
word_convert_setting_data := NewWord2PDFSettingData()
if runtime.GOOS == "linux" {
    ConvertFromWord(word_file_path, "", output_path, engine_path, word_convert_setting_data)
} else{
    ConvertFromWord(word_file_path, "", output_path, word_convert_setting_data)
}
```

3.37.3 How to convert Excel to PDF

```
import (
```



```
. "foxit.com/fsdk"
"fmt"
"runtime"
)
...
# Make sure that SDK has already been initialized successfully.

excel_file_path := "test.xls"
output_path := "saved.pdf"

# Use default Excel2PDFSettingData values.
excel_convert_setting_data := NewExcel2PDFSettingData()
if runtime.GOOS == "linux"{
    ConvertFromExcel(excel_file_path, "", output_path, engine_path, excel_convert_setting_data)
}
else{
    ConvertFromExcel(excel_file_path, "", output_path, excel_convert_setting_data)
}
```

3.37.4 How to convert PowerPoint to PDF

```
import (
. "foxit.com/fsdk"
"fmt"
)
...
# Make sure that SDK has already been initialized successfully.

ppt_file_path := "test.ppt"
output_path := "saved.pdf"

# Use default PowerPoint2PDFSettingData values.
ppt_convert_setting_data := NewPowerPoint2PDFSettingData()
if runtime.GOOS == "linux"{
    ConvertFromPowerPoint(ppt_file_path, "", output_path, engine_path, ppt_convert_setting_data)
}
else{
    ConvertFromPowerPoint(ppt_file_path, "", output_path, ppt_convert_setting_data)
}
```

3.38 Office to PDF Conversion with Foxit's self-developed engines

Foxit PDF SDK offers the capability to convert Microsoft Office documents (Word, Excel and PowerPoint) into professional-quality PDF files without any third-party engine. This feature is available through the Foxit PDF Conversion SDK on Windows and Linux x86/x64 platforms.

Foxit PDF SDK for Go API supports Linux x86/x64 platform.

3.38.1 System requirements

Platform: Windows, Linux (x86, and x64)

Programming Language: C, C++, Python, Java, C#, Node.js, Go

License Key requirement: 'Office2PDF' module permission in the license key

SDK Version: Foxit PDF SDK for Windows 10.1 or higher; Foxit PDF SDK for Linux x86/x64 11.0

3.38.2 Office to PDF resource files (Foxit PDF Conversion SDK)

Please contact Foxit support team or sales team to get the Foxit PDF Conversion SDK (C++).

After getting Foxit PDF Conversion SDK package, extract it to a desired directory, for example, extract the package to a directory: `"/home/user/Desktop/foxitpdfconversionsdk_*_Linux/"` for Linux x86/x64.

3.38.3 How to run the office2pdf demo using Foxit PDF Conversion SDK

Before running the office2pdf demo in the "examples/simple_demo/office2pdf" folder using Foxit PDF Conversion SDK, you should first add the Foxit PDF Conversion SDK library in the demo code, for example:

```
# If you want to convert office files to PDF whitout other third-party engines, you can use the Office2PDF module.
library_path := "/home/user/Desktop/foxitpdfconversionsdk_*_Linux/lib/libfpdconversionsdk_linux32.so"
```

Then, specify the office2pdf resource data files:

```
# A valid path of a folder which contains resource data files.
office2pdfSettingData.SetResource_folder_path("/home/user/Desktop/foxitpdfconversionsdk_*_Linux/res/office2pdf")
```

Finally, run the demo following the steps as the other demos.

3.38.4 How to convert office files to PDF with Foxit's self-developed engines

```
# If you want to convert office files to PDF whitout other third-party engines, you can use the Office2PDF module.
library_path := "" # Path of Foxit PDF Conversion SDK library, please ensure the path is valid.
# Initialize the Office2PDF module.
Office2PDFInitialize(library_path)
# Use default Office2PDFSettingData values.
office2pdf_setting_data := NewOffice2PDFSettingData()
# A valid path of a folder which contains resource data files.
office2pdf_setting_data.SetResource_folder_path("")
# Conver Word file to PDF file.
output_path := output_directory + "word2pdf_result_foxit.pdf"
Office2PDFConvertFromWord(word_file_path, "", output_path, office2pdf_setting_data)
```

```
# Conver Excel file to PDF file.
output_path = output_directory + "excel2pdf_result_foxit.pdf"
Office2PDFConvertFromExcel(excel_file_path, "", output_path, office2pdf_setting_data)
# Conver PowerPoint file to PDF file.
output_path = output_directory + "ppt2pdf_result_foxit.pdf"
Office2PDFConvertFromPowerPoint(ppt_file_path, "", output_path, office2pdf_setting_data)
# Release the Office2PDF module.
Office2PDFRelease()
```

3.39 Output Preview

Foxit PDF SDK supports output preview feature which can preview color separations and test different color profiles.

Note: Currently, the output preview feature is not supported on the Linux ARM platform.

3.39.1 System requirements

Platform: Windows, Linux (x86 and x64), Mac (x64)

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

License Key requirement: valid license key

SDK Version: Foxit PDF SDK (C, C++, Java, C#, Objective-C) 7.4 or higher; Foxit PDF SDK (Python) 8.3 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.39.2 How to run the output preview demo

Before running the output preview demo in the "examples/simple_demo/output_preview" folder, you should first set the folder path of "res/icc_profile" in the SDK package to the variable **default_icc_folder_path**. For example:

```
# "default_icc_folder_path" is the path of the folder which contains default icc profile files.
default_icc_folder_path := "/home/user/Desktop/foxitpdfsdk_X_X_linux_go/res/icc_profile"
```

Then, run the demo following the steps as the other demos.

3.39.3 How to do output preview using Foxit PDF SDK

```
import (
    . "foxit.com/fsdk"
    "fmt"
)
...
# Make sure that SDK has already been initialized successfully.

# Set folder path which contains default icc profile files.
LibrarySetDefaultICCPProfilesPath(default_icc_folder_path)
```

```
# Load a PDF document; Get a PDF page and parse it.
# Prepare a Renderer object and the matrix for rendering.

output_preview := NewOutputPreview(pdf_doc)
simulation_icc_file_path := input_path + "icc_profile/USWebCoatedSWOP.icc"
output_preview.SetSimulationProfile(simulation_icc_file_path)
output_preview.SetShowType(OutputPreviewE_ShowAll)
process_plates := output_preview.GetPlates(OutputPreviewE_ColorantTypeProcess)
spot_plates := output_preview.GetPlates(OutputPreviewE_ColorantTypeSpot)
process_size := int(process_plates.GetSize())
# Set check status of process plate to be true, if there's any process plate.
for i := 0; i < process_size; i++{
    output_preview.SetCheckStatus(process_plates.GetAt(i), true)
}
spot_size := int(spot_plates.GetSize())
# Set check status of spot plate to be true, if there's any spot plate.
for i := 0; i < spot_size; i++{
    output_preview.SetCheckStatus(spot_plates.GetAt(i), true)
}
# Generate preview bitmap
preview_bitmap := output_preview.GeneratePreviewBitmap(pdf_page, display_matrix, renderer)
```

3.40 Combination

Combination feature is used to combine several PDF files into one PDF file.

3.40.1 How to combine several PDF files into one PDF file

```
import (
    . "foxit.com/fsdk"
    "fmt"
)
...
# Make sure that SDK has already been initialized successfully.

info_array := NewCombineDocumentInfoArray()
info_array.Add(NewCombineDocumentInfo(input_path + "AboutFoxit.pdf", ""))
info_array.Add(NewCombineDocumentInfo(input_path + "Annot_all.pdf", ""))
info_array.Add(NewCombineDocumentInfo(input_path + "SamplePDF.pdf", ""))

savepath := output_directory + "Test_Combined.pdf"
option := CombinationE_CombineDocsOptionBookmark | CombinationE_CombineDocsOptionAcroformRename
| \
    CombinationE_CombineDocsOptionStructTree | CombinationE_CombineDocsOptionOutputIntents | \
    CombinationE_CombineDocsOptionOCProperties | CombinationE_CombineDocsOptionMarkInfos | \
    CombinationE_CombineDocsOptionPageLabels | CombinationE_CombineDocsOptionNames | \
    CombinationE_CombineDocsOptionObjectStream | CombinationE_CombineDocsOptionDuplicateStream

progress := CombinationStartCombineDocuments(savepath, info_array, option)
progress_state := ProgressiveE_ToBeContinued
for ProgressiveE_ToBeContinued == progress_state{
```

```
progress_state = progress.Continue()
}
```

3.41 PDF Portfolio

PDF portfolios are a combination of files with different formats. Portfolio file itself is a PDF document, and files with different formats can be embedded into this kind of PDF document.

3.41.1 System requirements

Platform: Windows, Linux, Mac

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

License Key requirement: valid license key

SDK Version: Foxit PDF SDK (C, C++, Java, C#, Objective-C) 7.6 or higher; Foxit PDF SDK (Python) 8.3 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

Example:

3.41.2 How to create a new and blank PDF portfolio

```
import (
    . "foxit.com/fsdk"
    "fmt"
)
...
# Make sure that SDK has already been initialized successfully.

new_portfolio = PortfolioCreatePortfolio()

# Set properties, add file/folder node to the new portfolio.
...

# Get portfolio PDF document object.
portfolio_pdf_doc := new_portfolio.GetPortfolioPDFDoc()
```

3.41.3 How to create a Portfolio object from a PDF portfolio

```
import (
    . "foxit.com/fsdk"
    "fmt"
)
...
# Make sure that SDK has already been initialized successfully.

portfolio_pdf_doc := NewPDFDoc("portfolio.pdf")
error_code := portfolio_pdf_doc.Load("")
if e_ErrSuccess == error_code{
    if portfolio_pdf_doc.IsPortfolio(){
```

```
    existed_portfolio := PortfolioCreatePortfolio(portfolio_pdf_doc)
  }
}
```

3.41.4 How to get portfolio nodes

```
import (
. "foxit.com/fsdk"
"fmt"
)
...
# Make sure that SDK has already been initialized successfully.

# Portfolio object has been created, assume it is named "portfolio".
...

root_node := portfolio.GetRootNode()
root_folder := NewPortfolioFolderNode(root_node)
sub_nodes := root_folder.GetSortedSubNodes()
sub_size := sub_nodes.GetSize()
for index := 0; index < sub_size; index++ {
    node := sub_nodes[index]
    note_type := node.GetNodeType()
    if note_type == PortfolioNodeE_TypeFolder {
        folder_node := NewPortfolioFolderNode(node)

        # Use PortfolioFolderNode's getting method to get some properties.
        ...
        sub_nodes_2 := folder_node.GetSortedSubNodes()
        ...
    }
    else if note_type == PortfolioNodeE_TypeFile {
        file_node := PortfolioFileNode(node)
        # Get file specification object from this file node, and then get/set information from/to this file
        specification object.
        file_spec := file_node.GetFileSpec()
        ...
    }
}
```

3.41.5 How to add file node or folder node

```
import (
. "foxit.com/fsdk"
"fmt"
)
...
# Make sure that SDK has already been initialized successfully.

# Portfolio object has been created, and the root folder node has been retrieved, assume it is named
"root_folder".
...
```

```
# Add file from path.
path_to_a_file := "directory/Sample.txt"
new_file_node_1 := root_folder.AddFile(path_to_a_file)

# User can update properties of file specification for new_file_node_1 if necessary.
...

# Add file from MyStreamCallback which is inherited from StreamCallback and implemented by user.
my_stream_callback := MyStreamCallback{
my_stream_callback_ := NewDirectorStreamCallback(my_stream_callback)
new_file_node_2 := root_folder.AddFile(my_stream_callback_, "file_name")

# Please get file specification of new_file_node_2 and update properties of the file specification by its setting
methods.
...

# Add a loaded PDF file.
# Open and load a PDF file, assume it is named "test_pdf_doc".
...

new_file_node_3 := root_folder.AddPDFDoc(test_pdf_doc, "pdf_file_name")

# User can update properties of file specification for new_file_node_3 if necessary.
...

# Add a sub folder in root_folder.
new_sub_foldernode := root_folder.AddSubFolder("Sub Folder-1")

# User can add file or folder node to new_sub_foldernode.
...
```

3.41.6 How to remove a node

```
import (
. "foxit.com/fsdk"
)
...

# Make sure that SDK has already been initialized successfully.

# Remove a child folder node from its parent folder node.
parent_folder_node.RemoveSubNode(child_folder_node)
# Remove a child file node from its parent folder node.
parent_folder_node.RemoveSubNode(child_file_node)
```

3.42 Table Maker

Foxit PDF SDK supports to add table to PDF files.

3.42.1 System requirements

Platform: Windows, Mac, Linux

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

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

SDK Version: Foxit PDF SDK (C, C++, C#, Java, Python, Objective-C) 8.4 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.42.2 How to add table to a PDF document

Foxit PDF SDK provides an electronictable demo located in the "examples/simple_demo/electronictable" folder to show you how to use Foxit PDF SDK to add table to PDF document.

```
import (
    . "foxit.com/fsdk"
)
...
index := 0
cell_array := NewTableCellDataArray()
for row:= 0; row< 4; row++){
    col_array := NewTableCellDataColArray()
    for col := 0 ;col< 3; col++){
        style := GetTableTextStyle(index)
        cell_text := GetTableCellText(index)
        cell_data := NewTableCellData(style, cell_text, NewImage(), NewRectF())
        col_array.Add(cell_data)
        index = index + 1
    }
    cell_array.Add(col_array)
}
page_width := pdf_page.GetWidth()
page_height := pdf_page.GetHeight()
outside_border_left := NewTableBorderInfo()
outside_border_left.SetLine_width(1)
outside_border_left.SetTable_border_style(TableBorderInfoE_TableBorderStyleSolid)
outside_border_right := NewTableBorderInfo()
outside_border_right.SetLine_width(1)
outside_border_right.SetTable_border_style(TableBorderInfoE_TableBorderStyleSolid)
outside_border_top := NewTableBorderInfo()
outside_border_top.SetLine_width(1)
outside_border_top.SetTable_border_style(TableBorderInfoE_TableBorderStyleSolid)
outside_border_bottom := NewTableBorderInfo()
outside_border_bottom.SetLine_width(1)
outside_border_bottom.SetTable_border_style(TableBorderInfoE_TableBorderStyleSolid)

inside_border_row_info := NewTableBorderInfo()
inside_border_row_info.SetLine_width(1)
inside_border_row_info.SetTable_border_style(TableBorderInfoE_TableBorderStyleSolid)
```



```
inside_border_col_info := NewTableBorderInfo()
inside_border_col_info.SetLine_width(1)
inside_border_col_info.SetTable_border_style(TableBorderInfoE_TableBorderStyleSolid)

data := NewTableData(NewRectF(100, 550, page_width - 100, page_height - 100), 4, 3, outside_border_left,
outside_border_right,
    outside_border_top, outside_border_bottom, inside_border_row_info, inside_border_col_info,
NewTableCellIndexArray(), NewFloatArray(), NewFloatArray())
TableGeneratorAddTableToPage(pdf_page, data, cell_array)
...
```

3.43 Accessibility

Foxit PDF SDK supports to tag PDF files.

3.43.1 System requirements

Platform: Windows, Mac, Linux

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

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

SDK Version: Foxit PDF SDK (C, C++, C#, Java, Python, Objective-C) 8.4 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.43.2 How to tag a PDF document

Foxit PDF SDK provides a taggedpdf demo located in the "examples/simple_demo/taggedpdf" folder to show you how to use Foxit PDF SDK to tag a PDF document.

```
import (
    . "foxit.com/fsdk"
)
....
pdfDoc := NewPDFDoc(input_file)
pdfDoc.Load("")
taggedpdf := NewTaggedPDF(pdfDoc)
progressive := taggedpdf.StartTagDocument()
progressState := ProgressiveE_ToBeContinued
for (ProgressiveE_ToBeContinued == progressState){
    progressState = progressive.Continue()
}

pdfDoc.SaveAs(output_file_path, uint(PDFDocE_SaveFlagNoOriginal))
```

3.44 PDF to Office Conversion

Foxit PDF SDK provides APIs to convert PDF files to MS office suite formats while maintaining the layout and format of your original documents on Windows and Linux platforms.

Foxit PDF SDK for Go API supports Linux x86/x64 platform.

3.44.1 System requirements

Platform: Windows, Linux

Programming Language: C, C++, Java, Python, C#, Node.js, Go

License Key requirement: 'PDF2Office' module permission in the license key

SDK Version: Foxit PDF SDK for Windows (C, C++, Java, Python, C#) 9.0 or higher; Foxit PDF SDK for Linux (C, C++, Java, Python, C#) 9.1 or higher; Foxit PDF SDK (Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.44.2 PDF to Office resource files

Please contact Foxit support team or sales team to get the PDF to Office resource files package naming Foxit PDF Conversion SDK (C++).

Note:

- For version 11.0, it requires Foxit PDF Conversion SDK 3.0.

After getting Foxit PDF Conversion SDK package, extract it to a desired directory, for example, **"/foxitpdfconversionsdk*_Linux/"** for Linux x86/x64.

3.44.3 How to run the pdf2office demo

Foxit PDF SDK provides a pdf2office demo located in the "examples/simple_demo/pdf2office" folder to show you how to use Foxit PDF SDK to convert PDF files to MS office suite formats.

3.44.3.1 Prepare a PDF2Office resource directory

Before running the pdf2office demo, you should first extract the PDF to Office resource files (Foxit PDF Conversion SDK) package to a desired directory (for example, extract the package to a directory: **"/home/user/Desktop/foxitpdfconversionsdk*_Linux/"** for Linux), and then pass the engine file located in "lib" folder to the API **PDF2OfficeInitialize** to initialize PDF2Office engine.

3.44.3.2 Configure the demo

For pdf2office demo, you can configure the demo in the "examples/simple_demo/pdf2office/**pdf2office.go**" file.

Specify the pdf2office engine directory

In the "pdf2office.go" file, add the path of the engine file "pdf2office" as follows, which will be used to convert PDF files to office files.

```
# Please ensure the path is valid.
```

```
PDF2OfficeInitialize("/home/user/Desktop/foxitpdfconversionsdk*_Linux/lib/libpdfconversionsdk_linux64.so",  
    "")
```

(Optional) Specify whether to enable machine learning-based recognition functionality

```
setting_data.SetEnable_ml_recognition(false)
```

(Optional) Specify the page range to be converted

```
setting_data.SetPage_range(NewRange())
```

(Optional) Specify whether to convert the comments in the PDF documents

```
setting_data.SetInclude_pdf_comments(true)
```

(Optional) Specify whether to retain the page layout for PDF to Word conversion

```
setting_data.GetWord_setting_data().SetEnable_retain_page_layout(false)
```

Note: The PDF to Office Conversion engine supports a timeout parameter. This parameter defines the maximum time allowed for the conversion process to complete. If the conversion exceeds the specified time, it will be terminated. The timeout must be a non-negative value. A value of 0 disables the timeout, allowing the conversion to proceed without any time limitation.

3.44.3.3 Run the demo

Once you run the demo successfully, the output files are located in "examples/simple_demo/output_files/pdf2office" folder.

3.44.4 How to work with PDF2office API

```
import (  
    "foxit.com/fsdk"  
    "fmt"  
)  
type CustomConvertCallback struct {  
    ConvertCallback  
}  
  
// Release releases resources  
func (callback CustomConvertCallback) Release() {  
}
```

```
// NeedToPause checks if the conversion process needs to pause
func (callback CustomConvertCallback) NeedToPause() bool {
    return true
}

// ProgressNotify notifies progress of conversion
func (callback CustomConvertCallback) ProgressNotify(convertedCount int, totalCount int) {
    // Can be implemented to track progress
}

callback := CustomConvertCallback{}
callback_ := NewDirectorConvertCallback(callback)
progressive := PDF2OfficeStartConvertToWord(input_path + "word.pdf", "", output_directory +
"pdf2word_result.docx", setting_data, callback_)
if progressive.GetRateOfProgress() != 100 {
    state := ProgressiveE_ToBeContinued
    for (ProgressiveE_ToBeContinued == state){
        state = progressive.Continue()
    }
}
fmt.Printf("Convert PDF file to Word format file with path.")

progressive = PDF2OfficeStartConvertToExcel(input_path + "excel.pdf", "", output_directory +
"pdf2excel_result.xlsx", setting_data, callback_)
if progressive.GetRateOfProgress() != 100{
    state := ProgressiveE_ToBeContinued
    for (ProgressiveE_ToBeContinued == state){
        state = progressive.Continue()
    }
}
fmt.Printf("Convert PDF file to Excel format file with path.")

progressive = PDF2OfficeStartConvertToPowerPoint(input_path + "powerpoint.pdf", "", output_directory +
"pdf2powerpoint_result.pptx", setting_data, callback_)
if progressive.GetRateOfProgress() != 100 {
    state := ProgressiveE_ToBeContinued
    for ProgressiveE_ToBeContinued == state {
        state = progressive.Continue()
    }
}
fmt.Printf("Convert PDF file to PowerPoint format file with path.")
```

3.45 DWG to PDF Conversion

Foxit PDF SDK supports to convert DWG files to PDF files. If you want to use this feature, you should contact Foxit support team or sales team to get the engine files package.

3.45.1 System requirements

Platform: Windows, Linux (x86 and x64), Mac(x64)

Programming Language: C, C++, Java, C#, Python, Objective-C, Node.js, Go

License Key requirement: 'DWG2PDF' module permission in the license key

SDK Version: Foxit PDF SDK (C, C++, Java, C#, Python, Objective-C, Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.45.2 DWG To PDF engine files

Please contact Foxit support team or sales team to get the DWG to PDF engine files package.

After getting the package, extract it to the desired directory. For example, extract the package to a directory: `"/home/user/Desktop/dwgtopdf/linux"` for Linux, and

`"/Users/user/Desktop/dwgtopdf/mac"` for Mac.

3.45.3 How to run the dwg2pdf demo

Before running the dwg2pdf demo in the "examples/simple_demo/dwg2pdf" folder, you should first add the dwg2pdf engine file in the demo code, for example:

```
# "engine_path" is the path of the engine file "dwg2pdf" which is used to convert dwg to pdf.  
engine_path := "/home/user/Desktop/dwgtopdf/linux"
```

Note: For Linux (x86 and x64) and Mac x64, before running the demo, you should configure environment variables.

- For Linux x86 and x64, add the path of the dwg2pdf engine file to `LD_LIBRARY_PATH` environment variable.

```
export LD_LIBRARY_PATH=/dwgtopdf/linux:$LD_LIBRARY_PATH
```

- For Mac x64, add the path of the dwg2pdf engine file to `LD_LIBRARY_PATH` environment variable.

```
export LD_LIBRARY_PATH=/dwgtopdf/mac:$LD_LIBRARY_PATH
```

Then, run the demo following the steps as the other demos.

3.45.4 How to convert DWG to PDF

```
import (  
  . "foxit.com/fsdk"  
  "fmt"  
)  
  
pdf_setting_data := NewDWG2PDFSettingData()
```

```
pdf_setting_data.SetExport_flags(DWG2PDFSettingDataE_FlagEmbeddedTTF)
pdf_setting_data.SetExport_hatches_type(DWG2PDFSettingDataE_DWG2PDFExportHatchesTypeBitmap)
pdf_setting_data.SetOther_export_hatches_type(DWG2PDFSettingDataE_DWG2PDFExportHatchesTypeBitmap)
pdf_setting_data.SetGradient_export_hatches_type(DWG2PDFSettingDataE_DWG2PDFExportHatchesTypeBitmap)
pdf_setting_data.SetSearchable_text_type(DWG2PDFSettingDataE_DWG2PDFSearchableTextTypeNoSearch)
pdf_setting_data.SetIs_active_layout(false)
pdf_setting_data.SetPaper_width(640)
pdf_setting_data.SetPaper_height(900)
ConvertFromDWG(engine_path, dwg_file_path, output_path, pdf_setting_data)
```

3.46 OFD

OFD files, standing for Open Financial Document, are used for storing and exchanging digital financial documents. They are open and XML-based, making them specifically designed for financial documents like contracts, invoices, and statements.

OFD files contain structured data and graphical elements defining the document's layout and content, including text, images, vector graphics, annotations, and other related information. The XML format facilitates easy interpretation, manipulation, and rendering of the document's content.

OFD files provide various benefits, including document integrity, security, and interoperability. They can be digitally signed to ensure authenticity and can be encrypted to protect sensitive information. OFD files also support interactive features like form fields and digital signatures.

To work with OFD files, OFD viewer or editor software that supports the OFD standard is needed. These tools allow you to display, edit, convert, and print the contents of OFD documents.

In essence, OFD files provide a standardized and efficient method for representing financial documents digitally, simplifying the exchange, storage, and management of financial information.

3.46.1 System requirements

Platform: Windows, Linux (x64 and armv8)

Programming Language: C, C++, Java, C#, Python, Node.js, Go

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

SDK Version: Foxit PDF SDK (C, C++, Java, C#, Python, Node.js) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.46.2 OFD engine file

Please contact Foxit support team or sales team to get the OFD engine file package.

After getting the package, extract it to the desired directory. For example, extract the package to a directory: `"/home/user/Desktop/ofd/linux64"` for Linux x64.

3.46.3 How to run the ofd demo

Before running the ofd demo in the "examples/simple_demo/ofd" folder, you should first add the ofd engine file in the demo code, for example:

```
# "engine_path" is the path of ofd engine file. Please refer to Developer Guide for more details.  
engine_path := "/home/user/Desktop/ofd/linux64" # For linux x64
```

Then, run the demo following the steps as the other demos.

3.46.4 How to implement the conversion between OFD file and PDF file

```
import (  
    . "foxit.com/fsdk"  
)  
  
# Initialize OFD engine.  
LibraryInitializeOFDEngine(engine_path)  
src_ofd_path := input_path + "wm_txttiled.ofd"  
src_pdf_path := input_path + "test.pdf"  
# Convert PDF document to OFD document, and convert OFD document to PDF document.  
convert_param := NewOFDConvertParam(false)  
# Convert OFD document to PDF document.  
ConvertFromOFD(src_ofd_path, "", output_directory + "ofd2pdf.pdf", convert_param)  
# Convert PDF document to OFD document.  
ConvertToOFD(src_pdf_path, "", output_directory + "pdf2ofd.ofd", convert_param)  
# Release OFD engine.  
LibraryReleaseOFDEngine()
```

3.46.5 How to render OFD page

```
import (  
    . "foxit.com/fsdk"  
)  
  
# Initialize OFD engine.  
LibraryInitializeOFDEngine(engine_path)  
# Render OFD document to bitmap.  
render_file_path := input_path + "content_flag.ofd"  
doc := NewOFDDoc(render_file_path, "")  
ofd_page := doc.GetPage(0)  
# Get the size of the page.  
w := int(ofd_page.GetWidth())  
h := int(ofd_page.GetHeight())  
bitmap := NewBitmap(w, h, BitmapE_DIBArgb)  
fRect := NewRectI(0, h, w, 0)  
bitmap.FillRect(0xFFFFFFFF, fRect)
```

```
# Get the display matrix of the page.
matrix_1 := ofd_page.GetDisplayMatrix(0, 0, w, h, E_Rotation0)

ofd_render := NewOFDRenderer(bitmap)
progressive := ofd_render.StartRender(ofd_page, matrix_1)
sSaveFilePath := output_directory + "renderBitmap.bmp"
# Add the bitmap to image and save the image.
image := NewImage()
image.AddFrame(bitmap)
image.SaveAs(sSaveFilePath)
ofd_render.Release()
ofd_page.Release()
doc.Release()
# Release OFD engine.
LibraryReleaseOFDEngine()
```

3.47 Paragraph Editing

Foxit PDF SDK offers a versatile set of tools for developers to fine-tune and customize text in PDF documents. The paragraph editing module provides complex adjustments, joining, and splitting functions that allow users to have precise control over the content of the document. The features are complemented by an intuitive UI implementation that facilitates efficient editing, ensuring a seamless and customized experience for managing text paragraphs.

The paragraph editing functionality revolves around two core modules, the **ParagraphEditing** module, and the **JoinSplit** module.

The **ParagraphEditing** module is designed to offer a variety of text editing operations, enabling users to easily perform the following actions according to their specific requirements:

- **Insert Text:** Insert new content at specific locations, allowing for customization of the document's precise layout.
- **Delete Text:** Delicately remove paragraphs or characters, enabling highly customized content trimming.
- **Modify Text:** Adjust existing text, including its content and formatting, to suit different editing styles.
- **Format Adjustment:** Support fine adjustments to paragraph formats and text styles, allowing for more accurate typesetting.

The **JoinSplit** module contains four vital operation types to support more complex text processing requirements:

- Join: Integrate multiple text blocks, enhancing content layout and overall document consistency.
- Split: Finely split text blocks, providing flexibility to manage various sections of the document.
- Link: Establish connections between text blocks, ensuring consistency in associated content.
- Unlink: Disconnect links between text blocks, offering more control over editing.

3.47.1 System requirements

Platform: Windows, Linux, Mac

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

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

SDK Version: Foxit PDF SDK (C, C++, Java, C#, Python, Objective-C) 10.0 or higher; Foxit PDF SDK (Go) 11.0

3.47.2 How to work with paragraph editing

```
import (
    . "foxit.com/fsdk"
)

...

type FxParagraphEditingProviderCallback struct {
    page PDFPage
}

# Release releases resources
func (callback FxParagraphEditingProviderCallback) Release() {
}

# GetRenderMatrix returns the render matrix
func (callback FxParagraphEditingProviderCallback) GetRenderMatrix(arg2 PDFDoc, arg3 int) Matrix2D {
    width := int(callback.page.GetWidth())
    height := int(callback.page.GetHeight())
    matrix := callback.page.GetDisplayMatrix(0, 0, width, height, callback.page.GetRotation())
    return matrix
}

# GetPageViewHandle returns the page view handle
func (callback FxParagraphEditingProviderCallback) GetPageViewHandle(arg2 PDFDoc, arg3 int) uintptr {
    return uintptr(0)
}

# GetClientRect returns the client rectangle
func (callback FxParagraphEditingProviderCallback) GetClientRect(arg2 PDFDoc) RectF {
    rect := NewRectF(float32(0), float32(0), float32(0), float32(0))
    return rect
}
```

```
}

# GetScale returns the scale
func (callback FxParagraphEditingProviderCallback) GetScale(arg2 PDFDoc, arg3 int) float32 {
    return 1.0
}

# GotoPageView navigates to the page view
func (callback FxParagraphEditingProviderCallback) GotoPageView(arg2 PDFDoc, arg3 int, arg4 float32, arg5
float32) bool {
    return true
}

# GetVisiblePageIndexArray returns an array of visible page indices
func (callback FxParagraphEditingProviderCallback) GetVisiblePageIndexArray(arg2 PDFDoc) Int32Array {
    pageArray := NewInt32Array()
    pageIndex := callback.page.GetIndex()
    pageArray.Add(pageIndex)
    return pageArray
}

# GetPageVisibleRect returns the visible rectangle of the page
func (callback FxParagraphEditingProviderCallback) GetPageVisibleRect(arg2 PDFDoc, arg3 int) RectF {
    rect := NewRectF(float32(0), float32(0), float32(0), float32(0))
    return rect
}

# GetPageRect returns the page rectangle
func (callback FxParagraphEditingProviderCallback) GetPageRect(arg2 PDFDoc, arg3 int) RectF {
    width := int(callback.page.GetWidth())
    height := int(callback.page.GetHeight())
    rect := NewRectF(float32(0), float32(height), float32(width), float32(0))
    return rect
}

# GetCurrentPageIndex returns the current page index
func (callback FxParagraphEditingProviderCallback) GetCurrentPageIndex(arg2 PDFDoc) int {
    return callback.page.GetIndex()
}

# GetRotation returns the rotation
func (callback FxParagraphEditingProviderCallback) GetRotation(arg2 PDFDoc, arg3 int) FoxitCommon_Rotation
{
    return E_Rotation0
}

# InvalidateRect invalidates the specified rectangle
func (callback FxParagraphEditingProviderCallback) InvalidateRect(arg2 PDFDoc, arg3 int, arg4 RectFArray) {
    // Not implemented for this example
}

# AddUndoItem adds an undo item
```

```
func (callback FxParagraphEditingProviderCallback) AddUndoItem(arg2 ParagraphEditingUndoItem) {
    # Not implemented for this example
}

# SetDocChangeMark sets the document change mark
func (callback FxParagraphEditingProviderCallback) SetDocChangeMark(arg2 PDFDoc) {
    # Not implemented for this example
}

# NotifyTextInputReachLimit notifies when text input reaches the limit
func (callback FxParagraphEditingProviderCallback) NotifyTextInputReachLimit(arg2 PDFDoc, arg3 int) {
    # Not implemented for this example
}

...

page := doc.GetPage(0)
page.StartParse(0)
height := page.GetHeight()
callback := FxParagraphEditingProviderCallback{ page: page;}
callback_ := NewDirectorParagraphEditingProviderCallback(callback)
editingMgr := NewParagraphEditingMgr(callback_, doc)

# Paragraph_editing
editing := editingMgr.GetParagraphEditing()
editing.Activate()
point_insert := NewPointF(95, height - 728)
editing.StartEditing(0, point_insert, point_insert)
editing.SetFontSize(24)
editing.SetUnderline(true)
editing.InsertText("InsertText_Paragraph_editing")
editing.Deactivate()
output_path := output_directory + "Paragraph_editing.pdf"
doc.SaveAs(output_path, PDFDocE_SaveFlagNoOriginal)

# Join&split
jionsplit := editingMgr.GetJoinSplit()
jionsplit.Activate()
point := NewPointF(289, 659)
jionsplit.OnLButtonDown(0, point)
jionsplit.OnLButtonUp(0, point)
jionsplit.SplitBoxes()
jionsplit.Deactivate()
output_path := output_directory + "Split_Boxes.pdf"
doc.SaveAs(output_path, PDFDocE_SaveFlagNoOriginal)

jionsplit.Activate()
point = NewPointF(307, height - 637)
jionsplit.OnLButtonDown(0, point)
jionsplit.OnLButtonUp(0, point)
point = NewPointF(307, height - 453)
jionsplit.OnLButtonDown(0, point)
jionsplit.OnLButtonUp(0, point)
```

```
jionsplit.JoinBoxes()  
jionsplit.Deactivate()
```

FAQ

1. How do I get text objects in a specified position of a PDF and change the contents of the text objects?

To get text objects in a specified position of a PDF and change the contents of the text objects using Foxit PDF SDK, you can follow the steps below:

- 1) Open a PDF file.
- 2) Load PDF pages and get the page objects.
- 3) Use **PDFPage.GetGraphicsObjectAtPoint** to get the text object at a certain position.
Note: use the page object to get rectangle to see the position of the text object.
- 4) Change the contents of the text objects and save the PDF document.

Following is the sample code:

```
import (
    . "foxit.com/fsdk"
    "fmt"
)
...

func ChangeTextObjectContent() bool {
    input_file := input_path + "AboutFoxit.pdf"
    doc := NewPDFDoc(input_file)
    error_code := doc.Load("")
    if error_code != E_ErrSuccess{
        print("The PDFDoc {} Error: {}".format(input_file, error_code))
        return false
    }

    # Get original shading objects from the first PDF page.
    original_page := doc.GetPage(0)
    original_page.StartParse(PDFPageE_ParsePageNormal)
    pointf := NewPointF()
    pointf.SetX(92)
    pointf.SetY(762)
    arr := original_page.GetGraphicsObjectsAtPoint(pointf, 10, GraphicsObjectE_TypeText)
    arr_size := arr.GetSize()
    for i := 0; i < arr_size; i++{
        graphobj := arr.GetAt(i)
        textobj := graphobj.GetTextObject()
        textobj.SetText("Foxit Test")
    }
    original_page.GenerateContent()
    output_directory := output_path + "graphics_objects/"
```

```
output_file := output_directory + "After_revise.pdf"
doc.SaveAs(output_file, uint(PDFDocE_SaveFlagNorma))
return true
}
```

2. Can I change the DPI of an embedded TIFF image?

No, you cannot change it. The DPI of the images in PDF files is static, so if the images already exist, Foxit PDF SDK does not have functions to change its DPI.

The solution is that you can use third-party library to change the DPI of an image, and then add it to the PDF file.

Note: Foxit PDF SDK provides a function "Image.SetDPIs" which can set the DPI property of an image object. However, it only supports the images that are created by Foxit PDF SDK or created by function "Image.AddFrame", and it does not support the image formats of JPX, GIF and TIF.

APPENDIX

Supported JavaScript List

Note: The minimum supported SDK Version listed in the tables is based on the C++ programming language.

Objects' property or method

Object	Properties/Method Names	Minimum Supported SDK Version
annotation properties	alignment	V7.0
	author	V7.0
	contents	V7.0
	creationDate	V7.0
	fillColor	V7.0
	hidden	V7.0
	modDate	V7.0
	name	V7.0
	opacity	V7.0
	page	V7.0
	readOnly	V7.0
	rect	V7.0
	richContents	V7.1
	rotate	V7.0
	strokeColor	V7.0
	textSize	V7.0
	type	V7.0
	AP	V9.0
	arrowBegin	V9.0
	arrowEnd	V9.0
	attachIcon	V9.0
	attachment	V9.0
	borderEffectIntensity	V9.0
	borderEffectStyle	V9.0
	callout	V9.0
	caretSymbol	V9.0
	dash	V9.0
	delay	V9.0
	doc	V9.0
	doCaption	V9.0
	gestures	V9.0

Object	Properties/Method Names	Minimum Supported SDK Version
	inReplyTo	V9.0
	intent	V9.0
	leaderExtend	V9.0
	leaderLength	V9.0
	lineEnding	V9.0
	lock	V9.0
	notelcon	V9.0
	noView	V9.0
	point	V9.0
	points	V9.0
	popupOpen	V9.0
	popupRect	V9.0
	print	V9.0
	quads	V9.0
	refType	V9.0
	richDefaults	V9.0
	seqNum	V9.0
	soundIcon	V9.0
	style	V9.0
	subject	V9.0
	textFont	V9.0
	toggleNoView	V9.0
	vertices	V9.0
	width	V9.0
annotation method	destroy	V7.0
	getProps	V9.0
	setProps	V9.0
	getStateInModel	V9.0
app properties	activeDocs	V4.0
	calculate	V4.0
	formsVersion	V4.0
	fs	V4.0
	fullscreen	V4.0
	language	V4.2
	platform	V4.0
	runtimeHighlight	V4.0
	viewerType	V4.0
	viewerVariation	V4.0
	viewerVersion	V4.0

Object	Properties/Method Names	Minimum Supported SDK Version
	printerNames	V8.4
	runtimeHighlightColor	V8.4
	constants	V8.4
app methods	alert	V4.0
	beep	V4.0
	browseForDoc	V4.0
	clearInterval	V4.0
	clearTimeout	V4.0
	launchURL	V4.0
	mailMsg	V4.0
	response	V4.0
	setInterval	V4.0
	setTimeout	V4.0
	popupMenu	V4.0
	execDialog	V8.4
	execMenuItem	V8.4
	newDoc	V8.4
	openDoc	V8.4
	popupMenuEx	V8.4
	addMenuItem	V8.4
	addSubMenu	V8.4
	addToolButton	V8.4
	removeToolButton	V8.4
	listMenuItems	V8.4
	trustedFunction	V8.4
	beginPriv	V8.4
	endPriv	V8.4
color properties	black	V4.0
	blue	V4.0
	cyan	V4.0
	dkGray	V4.0
	gray	V4.0
	green	V4.0
	ltGray	V4.0
	magenta	V4.0
	red	V4.0
	transparent	V4.0
	white	V4.0
	yellow	V4.0

Object	Properties/Method Names	Minimum Supported SDK Version
color methods	convert	V4.0
	equal	V4.0
document properties	author	V4.0
	baseURL	V4.0
	bookmarkRoot	V7.0
	calculate	V4.0
	Collab	V4.0
	creationDate	V4.0
	creator	V4.0
	delay	V4.0
	dirty	V4.0
	documentFileName	V4.0
	external	V4.0
	filesize	V4.0
	icons	V4.0
	info	V4.0
	keywords	V4.0
	modDate	V4.0
	numFields	V4.0
	numPages	V4.0
	pageNum	V4.0
	path	V4.0
	producer	V4.0
	subject	V4.0
	title	V4.0
	URL	V8.4
	dataObjects	V8.4
	hostContainer	V8.4
	templates	V8.4
	media	V8.4
	dynamicXFAForm	V8.4
	mouseX	V8.4
	mouseY	V8.4
	pageWindowRect	V8.4
	securityHandler	V8.4
	zoom	V8.4
	zoomType	V8.4
	layout	V8.4
	xfa	V8.4

Object	Properties/Method Names	Minimum Supported SDK Version
document methods	addAnnot	V7.0
	addField	V4.0
	addIcon	V4.0
	calculateNow	V4.0
	deletePages	V4.0
	exportAsFDF	V4.0
	flattenPages	V7.1
	getAnnot	V7.0
	getAnnots	V7.0
	getField	V4.0
	getIcon	V4.0
	getNthFieldName	V4.0
	getOCGs	V4.0
	getPageBox	V4.0
	getPageNthWord	V4.0
	getPageNthWordQuads	V4.0
	getPageNumWords	V4.0
	getPageRotation	V7.0
	getPrintParams	V4.0
	getURL	V4.0
	importAnFDF	V4.0
	insertPages	V6.2
	mailForm	V4.0
	print	V4.0
	removeField	V4.0
	replacePages	V6.2
	resetForm	V4.0
	submitForm	V4.0
	mailDoc	V4.0
	addWatermarkFromFile	V8.4
	addWatermarkFromText	V8.4
	getPageLabel	V8.4
	setPageLabels	V8.4
	gotoNamedDest	V8.4
	saveAs	V8.4
	scroll	V8.4
	setPageTabOrder	V8.4
	selectPageNthWord	V8.4
	syncAnnotScan	V8.4

Object	Properties/Method Names	Minimum Supported SDK Version
	getAnnot3D	V8.4
	getAnnots3D	V8.4
	addLink	V8.4
	removeLinks	V8.4
	getLinks	V8.4
	importIcon	V8.4
	removeIcon	V8.4
	addWeblinks	V8.4
	removeWeblinks	V8.4
	closeDoc	V8.4
	exportDataObject	V8.4
	importDataObject	V8.4
	removeDataObject	V8.4
	getDataObject	V8.4
	embedDocAsDataObject	V8.4
	createTemplate	V8.4
	removeTemplate	V8.4
	getTemplate	V8.4
	exportAsText	V8.4
	importTextData	V8.4
	exportAsXFDF	V8.4
	importAnXFDF	V8.4
	exportAsXFDFStr	V8.4
	extractPages	V8.4
	movePage	V8.4
	newPage	V8.4
	getOCGOrder	V8.4
	setOCGOrder	V8.4
	setPageBoxes	V8.4
	setPageRotations	V8.4
	setPageTransitions	V9.1
	getPageTransition	V9.1
event properties	change	V4.0
	changeEx	V4.0
	commitKey	V4.0
	fieldFull	V4.0
	keyDown	V4.0
	modifier	V4.0
	name	V4.0

Object	Properties/Method Names	Minimum Supported SDK Version
	rc	V4.0
	selEnd	V4.0
	selStart	V4.0
	shift	V4.0
	source	V4.0
	target	V4.0
	targetName	V4.0
	type	V4.0
	value	V4.0
	willCommit	V4.0
event methods	add	V9.0
field properties	alignment	V4.0
	borderStyle	V4.0
	buttonAlignX	V4.0
	buttonAlignY	V4.0
	buttonFitBounds	V4.0
	buttonPosition	V4.0
	buttonScaleHow	V4.0
	buttonScaleWhen	V4.0
	calcOrderIndex	V4.0
	charLimit	V4.0
	comb	V4.0
	commitOnSelChange	V4.0
	currentValueIndices	V4.0
	defaultValue	V4.0
	doNotScroll	V4.0
	doNotSpellCheck	V4.0
	delay	V4.0
	display	V4.0
	doc	V4.0
	editable	V4.0
	exportValues	V4.0
	hidden	V4.0
	fileSelect	V4.0
	fillColor	V4.0
	lineWidth	V4.0
	highlight	V4.0
	multiline	V4.0
	multipleSelection	V4.0

Object	Properties/Method Names	Minimum Supported SDK Version
	name	V4.0
	numItems	V4.0
	page	V4.0
	password	V4.0
	print	V4.0
	radiosInUnison	V4.0
	readonly	V4.0
	rect	V4.0
	required	V4.0
	richText	V4.0
	rotation	V4.0
	strokeColor	V4.0
	style	V4.0
	textColor	V4.0
	textFont	V4.0
	textSize	V4.0
	type	V4.0
	userName	V4.0
	value	V4.0
	valueAsString	V4.0
	richValue	V9.0
	submitName	V9.0
field methods	browseForFileToSubmit	V4.0
	buttonGetCaption	V4.0
	buttonGetIcon	V4.0
	buttonSetCaption	V4.0
	buttonSetIcon	V4.0
	checkThisBox	V4.0
	clearItems	V4.0
	defaultIsChecked	V4.0
	deleteItemAt	V4.0
	getArray	V4.0
	getItemAt	V4.0
	insertItemAt	V4.0
	isBoxChecked	V4.0
	isDefaultChecked	V4.0
	setAction	V4.0
	setFocus	V4.0
	setItems	V4.0

Object	Properties/Method Names	Minimum Supported SDK Version
	buttonImportIcon	V9.0
	getLock	V9.0
	setLock	V9.0
	signatureGetModifications	V9.0
	signatureGetSeedValue	V9.0
	signatureInfo	V9.0
	signatureSetSeedValue	V9.0
	signatureSign	V9.0
	signatureValidate	V9.0
global methods	setPersistent	V4.0
Icon properties	name	V4.0
util methods	printf	V4.0
	printf	V4.0
	printx	V4.0
	scand	V4.0
	iconStreamFromIcon	V9.0
identity properties	loginName	V4.2
	name	V4.2
	corporation	V4.2
	email	V4.2
collab properties	user	V6.2
ocg properties	name	V6.2
ocg methods	setAction	V6.2
bookmark properties	color	V8.4
	open	V8.4
	name	V8.4
	parent	V8.4
	children	V8.4
	language	V8.4
	style	V8.4
	platform	V8.4
bookmark methods	createChild	V8.4
	insertChild	V8.4
	execute	V8.4
	setAction	V8.4
	remove	V8.4
certificate properties	binary	V8.4
	issuerDN	V8.4
	keyUsage	V8.4

Object	Properties/Method Names	Minimum Supported SDK Version
	MD5Hash	V8.4
	privateKeyValidityEnd	V8.4
	privateKeyValidityStart	V8.4
	SHA1Hash	V8.4
	serialNumber	V8.4
	subjectCN	V8.4
	subjectDN	V8.4
	validityEnd	V8.4
	validityStart	V8.4
RDN properties	c	V8.4
	cn	V8.4
	e	V8.4
	l	V8.4
	o	V8.4
	ou	V8.4
	st	V8.4
security properties	handlers	V9.0
security methods	getHandler	V9.0
	importFromFile	V9.0
securityHandler properties	appearances	V9.0
	isLoggedIn	V9.0
	loginName	V9.0
	loginPath	V9.0
	name	V9.0
	uiName	V9.0
securityHandler methods	login	V9.0
	logout	V9.0
	newUser	V9.0
signatureInfo properties	objValidity	V9.0
	idValidity	V9.0
	idPrivValidity	V9.0
	docValidity	V9.0
	byteRange	V9.0
	verifyHandlerUIName	V9.0
	verifyHandlerName	V9.0
	verifyDate	V9.0
	subFilter	V9.0
	statusText	V9.0
	status	V9.0

Object	Properties/Method Names	Minimum Supported SDK Version
	reason	V9.0
	name	V9.0
	mdp	V9.0
	location	V9.0
	handlerUIName	V9.0
	handlerUserName	V9.0
	handlerName	V9.0
	dateTrusted	V9.0
	date	V9.0
search properties	attachments	V9.0
	bookmarks	V9.0
	docText	V9.0
	ignoreAccents	V9.0
	markup	V9.0
	matchCase	V9.0
	matchWholeWord	V9.0
	maxDocs	V9.0
	proximity	V9.0
	stem	V9.0
	wordMatching	V9.0
	ignoreAsianCharacterWidth	V9.0
search methods	query	V9.0
	addIndex	V9.0
	removeIndex	V9.0
link properties	borderColor	V8.4
	borderWidth	V8.4
	highlightMode	V8.4
	rect	V8.4
link methods	setAction	V8.4
app.media properties	align	V8.4
	canResize	V8.4
	ifOffScreen	V8.4
	over	V8.4
	windowType	V8.4
app.media methods	createPlayer	V8.4
	openPlayer	V8.4
doc.media methods	getOpenPlayers	V8.4
Playerargs properties	doc	V8.4
	annot	V8.4

Object	Properties/Method Names	Minimum Supported SDK Version
	rendition	V8.4
	URL	V8.4
	mimeType	V8.4
	settings	V8.4
	events	V8.4
MediaPlayer properties	isOpen	V8.4
	isPlaying	V8.4
	settings	V8.4
	visible	V8.4
MediaPlayer methods	close	V8.4
	play	V8.4
	seek	V8.4
	stop	V8.4
MediaSettings properties	autoPlay	V8.4
	baseURL	V8.4
	bgColor	V8.4
	bgOpacity	V8.4
	duration	V8.4
	floating	V8.4
	page	V8.4
	repeat	V8.4
	showUI	V8.4
	visible	V8.4
	volume	V8.4
	windowType	V8.4
floating properties	align	V8.4
	over	V8.4
	canResize	V8.4
	hasClose	V8.4
	hasTitle	V8.4
	title	V8.4
	ifOffScreen	V8.4
	rect	V8.4
eventListener methods	afterClose	V9.0
	afterPlay	V9.0
	afterReady	V9.0
	afterSeek	V9.0
	afterStop	V9.0
	onClose	V9.0

Object	Properties/Method Names	Minimum Supported SDK Version
	onPlay	V9.0
	onReady	V9.0
	onSeek	V9.0
	onStop	V9.0
Template properties	hidden	V9.1
	name	V9.1
Template method	spawn	V9.1
span properties	alignment	V9.1
	fontFamily	V9.1
	fontStretch	V9.1
	fontWeight	V9.1
	fontStyle	V9.1
	strikethrough	V9.1
	subscript	V9.1
	superscript	V9.1
	text	V9.1
	textColor	V9.1
	textSize	V9.1
	underline	V9.1
soap properties	wireDump	V9.1
Soap method	request	V9.1
	streamDigest	V9.1
	streamEncode	V9.1
	streamFromString	V9.1
	stringFromStream	V9.1
hostContainer method	postMessage	V9.2
Fullscreen properties	transitions	V9.2
	defaultTransition	V9.2
	loop	V9.2
	timeDelay	V9.2
	useTimer	V9.2
	isFullScreen	V9.2

Global methods

Method Names	Minimum Supported SDK Version
AFNumber_Format	V4.0
AFNumber_Keystroke	V4.0
AFPercent_Format	V4.0
AFPercent_Keystroke	V4.0

Method Names	Minimum Supported SDK Version
AFDate_FormatEx	V4.0
AFDate_KeystrokeEx	V4.0
AFDate_Format	V4.0
AFDate_Keystroke	V4.0
AFTime_FormatEx	V4.0
AFTime_KeystrokeEx	V4.0
AFTime_Format	V4.0
AFTime_Keystroke	V4.0
AFSpecial_Format	V4.0
AFSpecial_Keystroke	V4.0
AFSpecial_KeystrokeEx	V4.0
AFSimple	V4.0
AFMakeNumber	V4.0
AFSimple_Calculate	V4.0
AFRange_Validate	V4.0
AFMergeChange	V4.0
AFParseDateEx	V4.0
AFExtractNums	V4.0

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>

Note: sdk_folder is the directory of unzipped package.

SUPPORT

Foxit Support

In order to provide you with a more personalized support for a resolution, please log in to your [Foxit account](#) and submit a ticket so that we can collect details about your issue. We will work to get your problem solved as quickly as we can once your ticket is routed to our support team.

You can also check out our [Support Center](#), choose Foxit PDF SDK which also has a lot of helpful articles that may help with solving your issue.

Phone Support:

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