



开发者指南

Foxit PDF SDK

For .NET Core

Microsoft® Partner
Gold Independent Software Vendor (ISV)

TABLE OF CONTENTS

1	Foxit PDF SDK 简介	1
1.1	为什么选择 Foxit PDF SDK	1
1.2	Foxit PDF SDK for .NET Core	1
1.3	评估	2
1.4	授权	2
1.5	关于此文档	2
2	入门指南	2
2.1	系统要求	2
2.2	Windows	3
2.2.1	包结构说明	3
2.2.2	运行 demo	3
2.2.3	创建一个简单的工程	4
2.3	Linux	11
2.3.1	包结构说明	11
2.3.2	运行 demo	12
2.3.3	创建一个简单的工程	13
2.4	Mac	16
2.4.1	包结构说明	16
2.4.2	运行 demo	16
2.4.3	创建一个简单的工程	17
2.5	创建一个可以根据平台切换库文件的简单工程	20
3	使用 SDK API	26
3.1	初始化库	26
3.1.1	如何初始化 Foxit PDF SDK	26
3.2	文档 (Document)	26
3.2.1	如何从 0 开始创建一个 PDF 文档	26
3.2.2	如何通过文件路径加载一个现有的 PDF 文档	27

3.2.3	如何通过内存缓冲区加载一个现有的 PDF 文档.....	27
3.2.4	如何通过自定义实现的 ReaderCallback 对象加载一个现有的 PDF 文档.....	27
3.2.5	如何加载 PDF 文档以及获取文档的首页	28
3.2.6	如何将 PDF 文档另存为一个新的文档.....	28
3.2.7	如何通过 FileWriterCallback 将 PDF 文档保存到内存缓冲区.....	29
3.3	页面 (Page).....	30
3.3.1	如何获取页面的大小.....	30
3.3.2	如何计算页面内容的边界框	30
3.3.3	如何创建一个 PDF 页面以及设置其页面大小.....	30
3.3.4	如何删除一个 PDF 页面	30
3.3.5	如何扁平化一个 PDF 页面.....	31
3.3.6	如何获取和设置 PDF 文档中的页面缩略图	31
3.4	渲染 (Render).....	32
3.4.1	如何将 PDF 页面渲染到 bitmap	32
3.4.2	如何渲染页面和注释.....	33
3.5	附件 (Attachment).....	33
3.5.1	如何向 PDF 中插入附件文件	33
3.5.2	如何删除 PDF 中指定的附件文件	34
3.6	文本页面 (Text Page).....	34
3.6.1	如何从 PDF 页面中提取文本.....	34
3.6.2	如何在 PDF 文档中获取矩形区域中的文本	34
3.7	文本搜索 (Text Search).....	35
3.7.1	如何在 PDF 文档中搜索指定的文本.....	35
3.8	文本链接 (Text Link)	36
3.8.1	如何检索 PDF 页面中的超链接	36
3.9	书签 (Bookmark)	36
3.9.1	如何遍历 PDF 文档中所有的书签	36
3.9.2	如何向 PDF 文档中插入一个新的书签	37
3.10	表单 (AcroForm)	38
3.10.1	如何加载 PDF 中的表单	38
3.10.2	如何获取表单域个数以及设置其属性	38

3.10.3 如何将 PDF 中的表单数据导出到 XML 文件	39
3.10.4 如何通过 XML 文件导入表单数据到 PDF	39
3.10.5 如何获取和设置表单域的属性.....	40
3.11 XFA 表单	40
3.11.1 如何加载 XFADoc 并且显示 XFA 交互式表单.....	40
3.11.2 如何导出和导入 XFA 表单数据.....	41
3.12 表单设计 (Form Design).....	42
3.12.1 如何向 PDF 添加一个文本表单域	42
3.12.2 如何从 PDF 中移除一个文本表单域.....	42
3.13 注释 (Annotations)	42
3.13.1 常规注释	42
3.13.2 从 FDF 文件导入注释或者将注释导出到 FDF 文件	48
3.14 图片转换 (Image Conversion)	48
3.14.1 如何将 PDF 页面转换为位图文件	48
3.14.2 如何将图片转换为 PDF 文件	49
3.15 水印 (Watermark).....	49
3.15.1 如何创建一个文本水印，并将其插入到 PDF 文档的第一页	50
3.15.2 如何创建一个图片水印，并将其插入到 PDF 文档的第一页	50
3.15.3 如何从 PDF 页面中删除所有的水印.....	51
3.16 条形码 (Barcode)	51
3.16.1 如何从字符串生成条形码位图.....	51
3.17 安全 (Security).....	52
3.17.1 如何使用用户密码 "123" 和所有者密码 "456" 加密 PDF 文档	52
3.17.2 如何使用证书加密 PDF 文件	53
3.17.3 如何使用 Foxit DRM 加密 PDF 文件.....	53
3.18 页面重排 (Reflow)	54
3.18.1 如何创建一个 Reflow 页面，并将其渲染为位图文件	54
3.19 异步加载 PDF (Asynchronous PDF).....	55
3.20 压感笔迹 (Pressure Sensitive Ink)	55
3.20.1 如何创建 PSI 并设置相关属性.....	55

3.21	Wrapper	56
3.21.1	如何打开包含 wrapper 数据的 PDF 文档	56
3.22	PDF 对象 (PDF Objects)	57
3.22.1	如何从目录字典中删除指定的属性	57
3.23	页面对象 (Page Object).....	57
3.23.1	如何在 PDF 页面中创建一个文本对象	57
3.23.2	如何向 PDF 页面中插入一个图片 logo	58
3.24	标记内容 (Marked content).....	58
3.24.1	如何获取页面中的标记内容以及 tag 名称.....	59
3.25	PDF 图层 (PDF Layer)	59
3.25.1	如何创建一个 PDF 图层	59
3.25.2	如何设置所有图层节点的信息.....	60
3.25.3	如何编辑图层树	61
3.26	签名 (Signature).....	61
3.26.1	如何对 PDF 文档进行签名.....	61
3.27	长期签名验证(LTV, Long term validation).....	63
3.27.1	如何使用 SDK 默认的 sub filter "ETSI.RFC3161" 签名回调及默认的 RevocationCallback 进行长期 签名验证	64
3.28	PAdES	65
3.29	PDF 行为 (PDF Action).....	66
3.29.1	如何创建一个 URI 行为并将其插入到 link 注释	66
3.29.2	如何创建一个 GoTo 行为并将其插入到 link 注释	66
3.30	JavaScript	67
3.30.1	如何添加文档级的 JavaScript 动作	68
3.30.2	如何添加注释级的 JavaScript 动作	68
3.30.3	如何添加表单级的 JavaScript 动作	68
3.30.4	如何使用 JavaScript 向 PDF 页面添加一个新的注释	69
3.30.5	如何使用 JavaScript 获取/设置注释的属性 (strokeColor, fillColor, readOnly, rect, type 等)	70
3.30.6	如何使用 JavaScript 销毁注释	70
3.31	密文 (Redaction)	71

3.31.1 如何将 PDF 文档第一页中的文本 "PDF" 设置为密文.....	71
3.32 对比 (Comparison)	73
3.32.1 如何对比两个 PDF 文档，并将差异保存到一个 PDF 文件中.....	73
3.33 Compliance.....	74
3.33.1 系统需求	75
3.33.2 Compliance 资源文件	75
3.33.3 如何运行 compliance demo	76
3.34 优化 (Optimization).....	79
3.34.1 如何通过压缩 PDF 文件中的彩色、灰度和黑白图像来减少 PDF 文件的大小	79
3.35 HTML 转 PDF.....	80
3.35.1 系统需求	80
3.35.2 HTML 转 PDF 引擎资源	81
3.35.3 如何运行 html2pdf demo	81
3.36 Office 转 PDF	83
3.36.1 系统需求	83
3.36.2 如何将 Word 文档转换为 PDF 文档	84
3.36.3 如何将 Excel 文件转换为 PDF 文档.....	84
3.36.4 如何将 PowerPoint 文件转换为 PDF 文档.....	84
3.37 输出预览 (Output Preview).....	85
3.37.1 系统需求	85
3.37.2 如何运行 output preview demo	85
3.37.3 如何使用 Foxit PDF SDK 进行输出预览	85
3.38 合并 (Combination).....	86
3.38.1 如何将多个 PDF 文件合并成一个 PDF 文件	86
3.39 PDF Portfolio	87
3.39.1 系统需求	87
3.39.2 如何创建一个新的空白的 PDF Portfolio 文档.....	87
3.39.3 如何从一个 PDF portfolio 文档创建一个 Portfolio 对象.....	87
3.39.4 如何获取 portfolio nodes.....	88
3.39.5 如何添加 file node 或者 folder node	89
3.39.6 如何移除 node.....	90

FAQ	91
附录	93
Foxit PDF SDK 支持的 JavaScript 列表.....	93
引用	99
技术支持	100

1 FOXIT PDF SDK 简介

您是否曾经想要构建一个可以对 PDF 文档进行任何操作的应用程序？如果您的答案是 "Yes"，那么恭喜您！您找到了业界中可以构建稳定、安全、高效且功能齐全的 PDF 应用的优选解决方案。

Foxit PDF SDK 提供高性能的开发库，帮助软件开发人员使用最流行的开发语言和环境在不同平台（包括 Windows、Mac、Linux、Web、Android、iOS 和 UWP）的企业版、移动版和云应用程序中添加强大的 PDF 功能。

1.1 为什么选择 Foxit PDF SDK

Foxit 是领先的 PDF 软件解决方案供应商，专注于 PDF 显示、编辑、创建、管理以及安全方面。Foxit PDF SDK 开发库已在当今许多知名的应用程序中使用，并且经过长期的测试证明 Foxit PDF SDK 的质量、性能和功能正是业界大部分应用程序所需要的。选择 Foxit PDF SDK 产品的几大理由：

- **易于集成**

开发人员可以将 SDK 无缝集成到他们自己的应用程序中。

- **轻量级**

部署简单快速，占用系统资源少。

- **支持跨平台**

支持当前主流的平台，比如 Windows、Mac、Linux、Web、Android、iOS 和 UWP。

- **基于福昕高保真的 PDF 渲染引擎**

Foxit PDF SDK 的核心技术是基于世界众多知名企业所信赖的福昕 PDF 引擎。福昕强大的 PDF 引擎可快速解析和渲染文档，不受设备环境的约束。

- **优秀的技术支持**

福昕对自己的开发产品提供了优秀的技术支持，当您在开发关键重要的产品时，可以提供高效的帮助和支持。福昕拥有一支 PDF 行业优秀的技术支持工程师团队，同时将定期地进行版本更新发布，通过添加新的功能和增强已有的功能来提升用户体验。

1.2 Foxit PDF SDK for .NET Core

使用 Foxit PDF SDK 的应用程序开发人员可以利用 Foxit 强大、符合标准的 PDF 技术来安全地显示、创建、编辑、注释、格式化、组织、打印、共享、保护、搜索文档，以及填写 PDF 表单。此外，Foxit PDF SDK (C++和.NET) 包含一个内置可嵌入的 PDF Viewer，使开发过程更容易和更快捷。有关更多详细信息，请访问网站 <https://developers.foxitsoftware.cn/pdf-sdk/>。

在本手册中，我们专注于介绍支持 Windows、Linux 和 Mac 跨平台的 Foxit PDF SDK for .NET Core。

.NET Core 是一个开源的，由微软和 Github 上的.NET 社群共同维护的通用开发平台。它是跨平台(支持 Windows、Linux 和 Mac)，可用于生成设备、云和物联网应用。

Foxit PDF SDK for .NET Core 提供简单易用的 API，帮助.NET 开发人员将强大的 PDF 技术无缝集成到他们自己的 Windows、Linux 和 Mac 平台项目中。并且提供了 PDF 文档相关的丰富功能，比如 PDF 浏览、书签导航、文本选择/复制/搜索、PDF 签名、PDF 表单、权限管理、PDF 注释以及全文搜索等。

1.3 评估

用户可申请下载 Foxit PDF SDK 的试用版本进行试用评估。试用版除了有试用期 10 天时间的限制以及生成的 PDF 页面上会有试用水印以外，其他都和标准认证版一样。当试用期到期后，用户需联系福昕销售团队并购买 licenses 以便继续使用 Foxit PDF SDK.

1.4 授权

程序开发人员需购买 licenses 授权才能在其解决方案中使用 Foxit PDF SDK。Licenses 授予用户发布基于 Foxit PDF SDK 开发的应用程序的权限。然而，在未经福昕软件公司授权下，用户不能将 Foxit PDF SDK 包中的任何文档、示例代码以及源代码分发给任何第三方机构。

1.5 关于此文档

此文档适用于需要将 Foxit PDF SDK for .NET Core 集成到自己的应用程序中的开发人员。它旨在介绍 SDK 包结构和 SDK 的用法。

2 入门指南

安装并集成 Foxit PDF SDK 非常简单。本手册将介绍如何在 Windows、Linux 和 Mac 平台集成 Foxit PDF SDK for .NET Core。本章的主要内容是介绍系统要求、SDK 包结构、以及如何运行 demo 和创建自己的项目。

2.1 系统要求

先决条件：在 Windows/Linux/macOS 平台上已安装.NET Core 2.1 或更高版本。

平台	系统要求	备注
Windows	Windows 10 (32-bit, 64-bit)	Visual Studio 2017 15.9 或更高版本
Linux	64-bit OS	所有的 Linux 示例都在 Ubuntu14.04 64 位系统上进行过测试。
Mac	Mac OS X 10.6 或更高版本 (64-bit)	Visual Studio for Mac 8.0 或更高版本

2.2 Windows

在本手册中，请知晓：图片上面高亮的矩形区域指的是 SDK 的版本号，当前 SDK 的版本是 8.1，则其代表 8_1。

2.2.1 包结构说明

下载 Foxit PDF SDK for .NET Core (Windows)包，解压到一个新的目录如 "foxitpdfsdk_8_1_win_dotnetcore"，如 Figure 2-1 所示。其中解压包中包括如下的内容：

doc: API 手册，开发者指南

examples: 示例工程和 demos

lib: SDK 库和授权文件

res: 输出预览 (output preview) demo 使用的默认 icc profile 文件

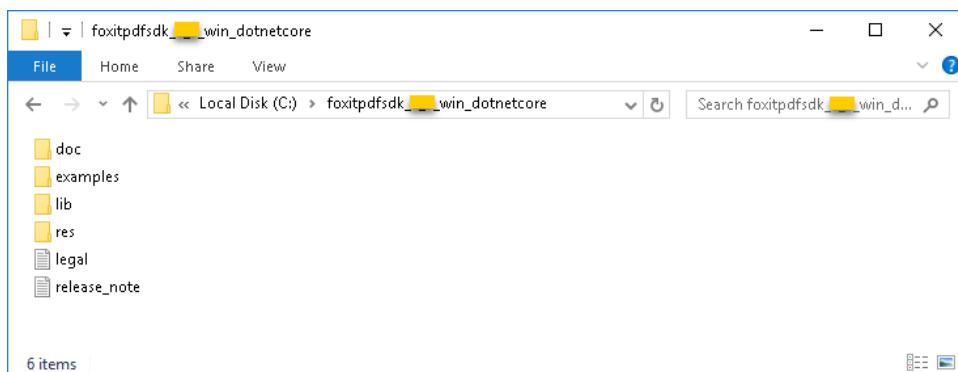


Figure 2-1

2.2.2 运行 demo

Foxit PDF SDK for .NET Core (Windows) 提供了一些示例 demo，在"\examples\simple_demo"目录下。所有的 demo (除了 security, compliance, html2pdf, office2pdf 和 output preview demo 以外)，都可以直接使用"\examples\simple_demo"目录下的 "RunDemo.bat" 文件在命令行中运行。

打开命令行窗口，导航到 "\examples\simple_demo"，然后运行如下的命令：

- 对于 32 位的操作系统，输入 "**RunDemo.bat all x86**" 运行所有的 demo，或者输入 "**RunDemo.bat demo_name x86**" 运行特定的单个 demo，比如，"**RunDemo.bat bookmark x86**" 只运行 bookmark demo。

- 对于 64 位的操作系统，输入 "**RunDemo.bat all**" 运行所有的 demo，或者输入 "**RunDemo.bat demo_name**" 运行特定的单个 demo，比如，"**RunDemo.bat bookmark**" 只运行 bookmark demo。

"\examples\simple_demo\input_files" 文件夹下包含了所有 demo 使用的输入文件。对于会生成输出文件 (pdf, 文本或者图片文件) 的 demo，会在 "\examples\simple_demo\output_files\" 文件夹下生成以该 demo 名称命名的文件夹，并且输出文件将会在该文件夹下生成。

在 "\examples\simple_demo\output_files\security" 文件夹下，如果您想要打开 "certificate_encrypt.pdf" 文档，您需要首先安装 "\examples\simple_demo\input_files" 文件夹下的 "foxit.cer" 和 "foxit_all.pfx" 证书。请按照如下的步骤：

- 安装 "foxit.cer"，双击其启动证书导入向导。然后选择 "Install certificate... > Next > Next > Finish"。
- 安装 "foxit_all.pfx"，双击其启动证书导入向导。然后选择 "Next > Next > (在文本框中输入私钥的密码)，然后点击 Next > Next > Finish"。

Compliance demo

对于 **compliance** demo，您需要首先构建一个资源目录，请联系 Foxit 支持团队或者销售团队获取相应的资源包。关于如何运行该 demo 的更详细的信息，请参考 3.33 "[Compliance](#)".

HTML to PDF demo

对于 **html2pdf** demo，您需要首先联系 Foxit 支持团队或者销售团队获取 HTML 转 PDF 的引擎包。关于如何运行该 demo 的更详细的信息，请参考 3.35 小节 "[HTML 转 PDF](#)".

Office to PDF demo

对于 **office2pdf demo**，您需要确保您 Windows 系统上已安装 Microsoft Office 2007 或以上版本，并且设置了默认打印机 (虚拟打印机也可以)。

Output Preview demo

对于 **output preview demo**，您需要设置包含默认 icc profile 文件的文件夹路径。关于如何运行该 demo 的更详细信息，请参考 3.37 小节 "[输出预览 \(Output Preview\)](#)"。

2.2.3 创建一个简单的工程

本节主要介绍如何使用 Foxit PDF SDK for .NET Core 创建一个简单的工程，该工程将 PDF 文档的首頁渲染成 bitmap，然后将其另存为 JPG 图片。该工程将以.NET Core 2.1 和 64 位操作系统为例。创建该工程，请按照如下的步骤操作：

- 1) 打开 Visual Studio 2017，创建一个名为 "test_dotnetcore" 的 C#语言的 .NET Core 控制台应用程序，如 Figure 2-2 所示。

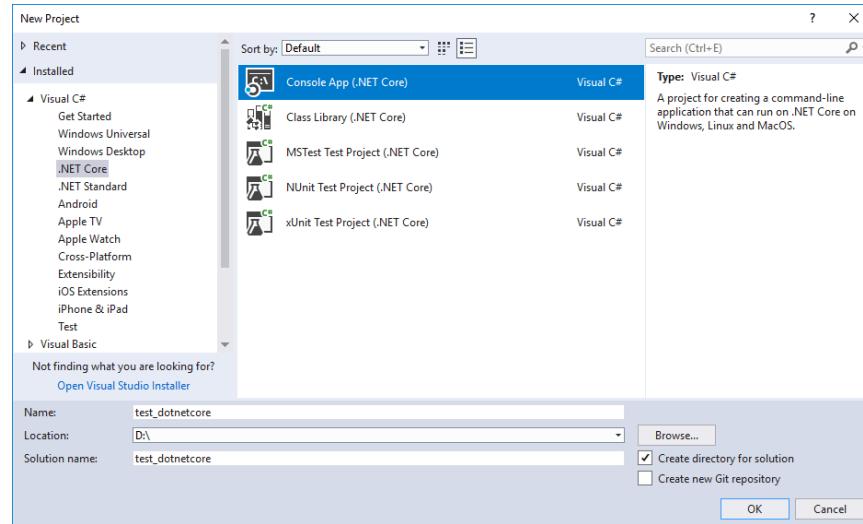


Figure 2-2

- 2) 将 "foxitpdfsdk_8_1_win_dotnetcore" 文件夹下的 "lib" 文件夹拷贝到 "test_dotnetcore" 工程目录下。
- 3) 添加 Foxit PDF SDK for .NET Core 动态库到工程的 **Dependencies**。为了在工程中使用 Foxit PDF SDK APIs，您必须首先引用 SDK 库。
 - i. 在 Solution Explorer 中，右击工程的 **Dependencies** 节点，然后点击 **Add Reference...**
 - ii. 在 **Reference Manager** 对话框，点击 **Browse** 标签，根据操作系统导航到 "test_dotnetcore\lib\x64_vc15" 或者 "test_dotnetcore\lib\x86_vc15" 文件夹。这里，我们使用 64 位的操作系统，因此导航到 "test_dotnetcore\lib\x64_vc15" 文件夹，选择 **fsdk_dotnetcore.dll** 动态库，然后点击 **OK**。(见 Figure 2-3 和 Figure 2-4)

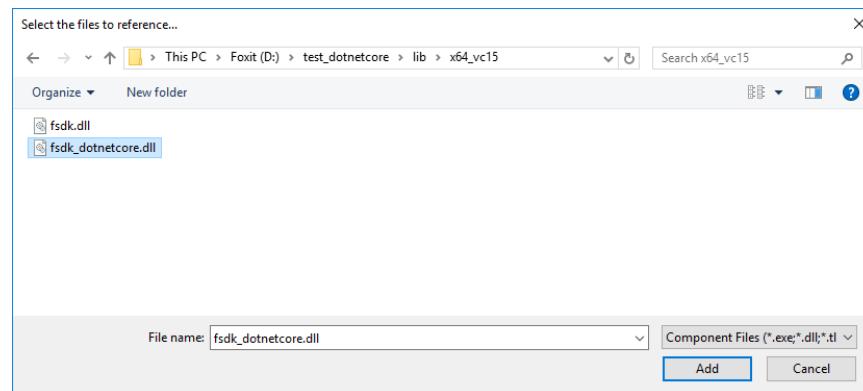


Figure 2-3

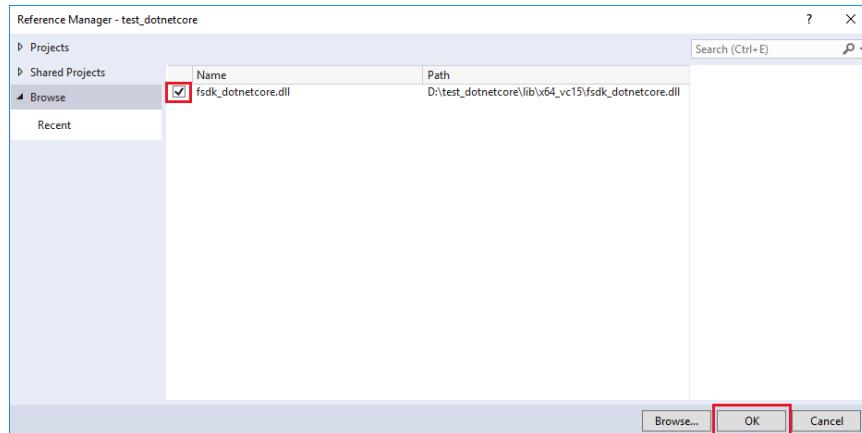


Figure 2-4

- 4) 添加 "System.Drawing.Common" NuGet 包到工程的 **Dependencies**。
- 右击工程的 **Dependencies** 节点，然后点击 **Manage NuGet Packages...**
 - 然后选择 **Browse** 选项卡，搜索 **System.Drawing.Common**，并安装。(见 Figure 2-5)

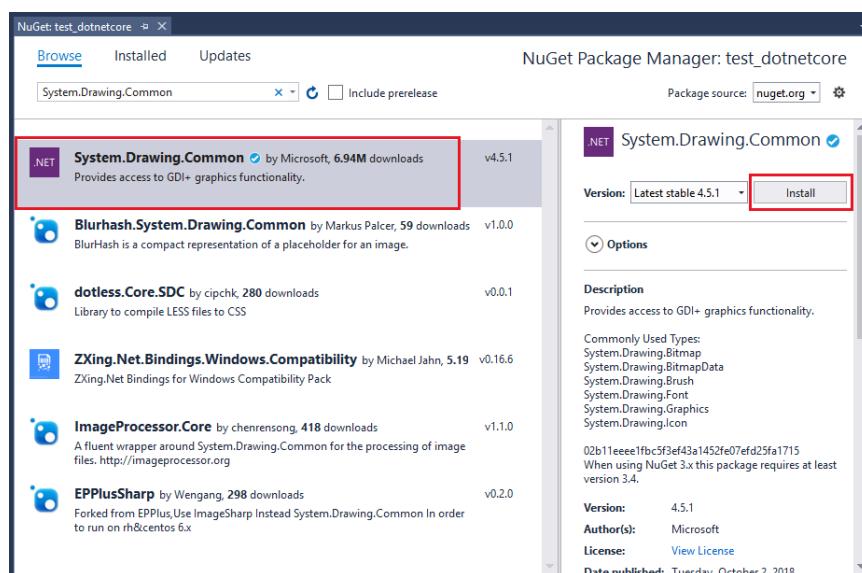


Figure 2-5

- 5) 将 "fsdk.dll" 添加到工程中。
- 右击 "test_dotnetcore" 工程，点击 **Add > Existing Item...**，导航到 "test_dotnetcore\lib\x64_vc15" 文件夹，选择 **fsdk.dll** 动态库，然后点击 **Add**。(见 Figure 2-6)

备注：请确保将 "fsdk.dll" 的 "Copy to Output Directory" 属性设置为 "Copy if newer"。否则，在运行工程之前，您需要手动将 "fsdk.dll" 动态库拷贝到可执行文件所在的目录。

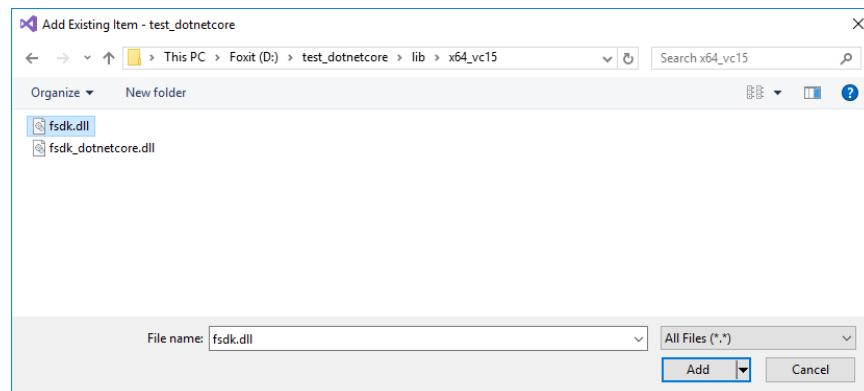


Figure 2-6

- 6) 配置工程的 build architecture。

点击 **Build -> Configuration Manager...**, 设置 "Active solution platform" 为 **x64**, 如 Figure 2-7 所示。

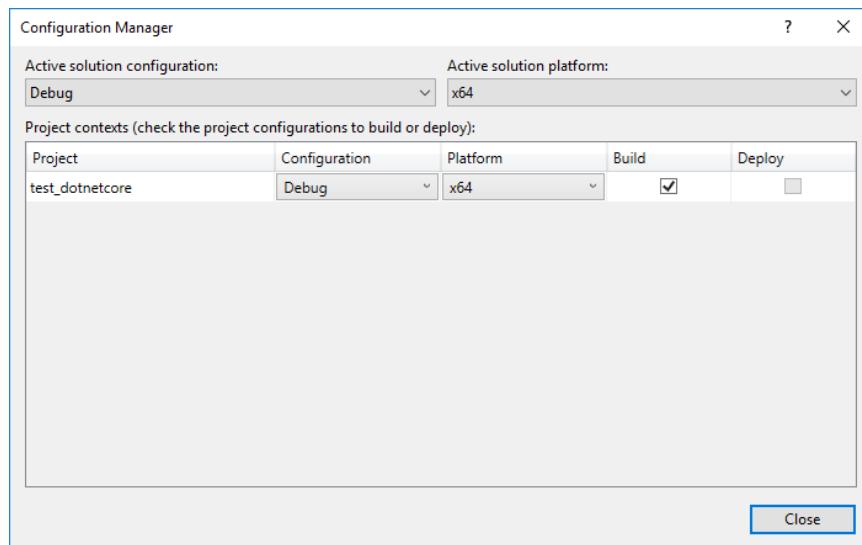


Figure 2-7

备注: 需要根据操作系统设置对应正确的 build architecture。

- 7) 拷贝一个 PDF 文件 (比如, "Sample.pdf") 到 "test_dotnetcore\test_dotnetcore" 目录下, 用于测试该工程。

备注: 请确保将 "**Sample.pdf**" 的 "Copy to Output Directory" 属性设置为 "Copy if newer"。否则, 在运行工程之前, 您需要手动将 "**Sample.pdf**" 拷贝到可执行文件所在的目录。

然后, *test_dotnetcore* 工程将如 Figure 2-8 所示。

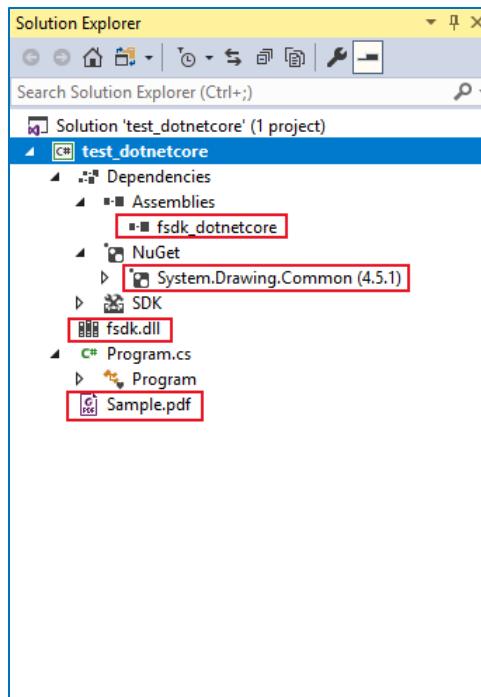


Figure 2-8

备注: 完成步骤3 到步骤7 后, 右击 "test_dotnetcore" 工程, 点击 **Edit test_dotnetcore.csproj**, 然后您将看到 **test_dotnetcore.csproj** 文件内容如下所示:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>netcoreapp2.1</TargetFramework>
  <Platforms>AnyCPU;x64</Platforms>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="System.Drawing.Common" Version="4.5.1" />
</ItemGroup>

<ItemGroup>
  <Reference Include="fsdk_dotnetcore">
    <HintPath>..\lib\x64_vc15\fsdk_dotnetcore.dll</HintPath>
  </Reference>
</ItemGroup>

<ItemGroup>
  <None Update="fsdk.dll">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
  <None Update="Sample.pdf">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

```
</None>
<ItemGroup>

</Project>
```

当创建工程时，您可以不需要操作步骤 3 到步骤 7，直接手动编辑 ".csproj" 文件，根据您的工程配置 fsdk_dotnetcore.dll 和 fsdk.dll 的路径。

- 8) 在 "Program.cs" 文件的开头加入 using 命名空间声明。

```
using System.Drawing;

using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
```

- 9) 初始化 Foxit PDF SDK 库。在调用任何 APIs 之前，应用程序必须使用 license 授权码初始化 Foxit PDF SDK 库。试用 license 文件在 "lib" 文件夹下。

```
string sn = " ";
string key = " ";
ErrorCode error_code = Library.Initialize(sn, key);
if (error_code != ErrorCode.e_ErrSuccess)
{
    return;
}
```

备注：参数 "sn" 的值在 "**gsdk_sn.txt**" 中 ("SN="后面的字符串)， "key" 的值在 "**gsdk_key.txt**" 中 ("Sign="后面的字符串)。

- 10) 加载一个 PDF 文档 (Sample.pdf)，然后解析该文档的首页。

```
PDFDoc doc = new PDFDoc("Sample.pdf");
error_code = doc.LoadW("");
if (error_code != ErrorCode.e_ErrSuccess)
{
    return;
}

// Get the first page of the document.
PDFPage page = doc.GetPage(0);

// Parse page.
page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);
```

- 11) 将页面渲染成 bitmap，然后将其另存为 JPG 图片。

```
int width = (int)(page.GetWidth());
int height = (int)(page.GetHeight());
Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());
```

```
// Prepare a bitmap for rendering.  
foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height,  
foxit.common.Bitmap.DIBFormat.e_DIBRgb32);  
System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();  
Graphics draw = Graphics.FromImage(sbitmap);  
draw.Clear(Color.White);  
  
// Render page  
Renderer render = new Renderer(bitmap, false);  
render.StartRender(page, matrix, null);  
  
// Add the bitmap to image and save the image.  
foxit.common.Image image = new foxit.common.Image();  
image.AddFrame(bitmap);  
image.SaveAs("testpage.jpg");
```

12) 点击 "Build > Build Solution" 编译该工程。

13) 点击 "Debug > Start Without Debugging" 运行工程，然后在输出目录下 ("test_dotnetcore\test_dotnetcore\bin\x64\Debug\netcoreapp2.1") 会生成 "testpage.jpg"。

备注：请检查 "fsdk.dll" 和 "fsdk_dotnetcore.dll" 是否拷贝至输出目录。如果没有，您需要手动将动态库拷贝至该目录下。

"Program.cs" 的完整内容如下：

```
using System;  
  
using System.Drawing;  
using foxit.common;  
using foxit.common.fxcr;  
using foxit.pdf;  
  
namespace test_dotnetcore  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            // 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=").  
            string sn = " ";  
            string key = " ";  
            ErrorCode error_code = Library.Initialize(sn, key);  
            if (error_code != ErrorCode.e_ErrSuccess)  
            {  
                return;  
            }
```

```
using (PDFDoc doc = new PDFDoc("Sample.pdf"))
{
    error_code = doc.LoadW("");
    if (error_code != ErrorCode.e_ErrSuccess)
    {
        Library.Release();
        return;
    }

    using (PDFPage page = doc.GetPage(0))
    {
        // Parse page.
        page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);

        int width = (int)(page.GetWidth());
        int height = (int)(page.GetHeight());
        Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

        // Prepare a bitmap for rendering.
        foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height,
foxit.common.Bitmap.DIBFormat.e_DIBRgb32);
        System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();
        Graphics draw = Graphics.FromImage(sbitmap);
        draw.Clear(Color.White);

        // Render page
        Renderer render = new Renderer(bitmap, false);
        render.StartRender(page, matrix, null);

        // Add the bitmap to image and save the image.
        foxit.common.Image image = new foxit.common.Image();
        image.AddFrame(bitmap);
        image.SaveAs("testpage.jpg");
    }
}
Library.Release();
}
```

2.3 Linux

在本手册中，请知晓：图片上面高亮的矩形区域指的是 SDK 的版本号，当前 SDK 的版本是 8.1，则其代表 8_1。

2.3.1 包结构说明

下载 Foxit PDF SDK for .NET Core (Linux 64 位) 包，解压到一个新的目录如 "foxitpdfsdk_8_1_linux64_dotnetcore"，如 Figure 2-9 所示。其中解压包中包括如下的内容：

- docs:** API 手册, 开发者指南
- examples:** 示例工程和 demos
- lib:** SDK 库和授权文件
- res:** 输出预览 (output preview) demo 使用的默认 icc profile 文件

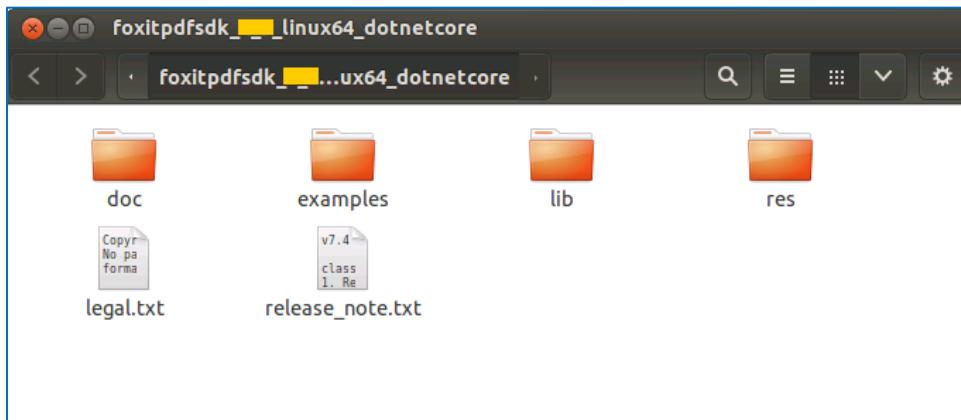


Figure 2-9

2.3.2 运行 demo

在运行 demo 之前, 请确保您已经在 64 位的 Linux 上正确配置了.NET Core 环境。

备注: Foxit PDF SDK for .NET Core demo 和下节中创建的工程都引用了 "System.Drawing.Bitmap", 因此请使用 "apt-get install libgdiplus" 命令安装 "libgdiplus" 库。

Foxit PDF SDK for .NET Core (Linux 64 位) 提供了一些示例 demo, 在 "\examples\simple_demo" 目录下。所有的 demo (除了 compliance 和 output preview demo 以外) 都可以直接使用 "\examples\simple_demo" 目录下的 "RunDemo.sh" 文件在终端窗口中运行。

打开终端窗口, 导航到 "\examples\simple_demo", 然后运行如下的命令:

- 输入 ".**R**un**D**emo.**sh** all" 运行所有的 demo。
- 输入 ".**R**un**D**emo.**sh** demo_name" 运行特定的单个 demo, 比如, ".**R**un**D**emo.**sh** bookmark" 只运行 bookmark demo。

"\examples\simple_demo\input_files" 文件夹下包含了所有 demo 使用的输入文件。对于会生成输出文件 (pdf, 文本或者图片文件) 的 demo, 会在 "\examples\simple_demo\output_files\" 文件夹下生成以该 demo 名称命名的文件夹, 并且输出文件将会在该文件夹下生成。

Compliance demo

对于 **compliance** demo, 您需要首先构建一个资源目录, 请联系 Foxit 支持团队或者销售团队获取相应的资源包。关于如何运行该 demo 的更详细的信息, 请参考 3.33 "[Compliance](#)"。

Output Preview demo

对于 **output preview demo**，您需要设置包含默认 icc profile 文件的文件夹路径。关于如何运行该 demo 的更详细信息，请参考 3.37 小节 "[输出预览 \(Output Preview\)](#)"。

2.3.3 创建一个简单的工程

本节主要介绍如何使用 Foxit PDF SDK for .NET Core 创建一个简单的工程，该工程将 PDF 文档的首页渲染成 bitmap，然后将其另存为 JPG 图片。该工程将以.NET Core 2.2 为例。创建该工程，请按照如下的步骤操作：

- 1) 创建一个名为 "test_dotnetcore" 的 C#语言的 .NET Core 控制台应用程序。打开一个终端窗口，导航到您要创建工程的目录，运行如下的命令：

```
dotnet new console -lang C# -o test_dotnetcore
```

然后，*test_dotnetcore* 工程将如 Figure 2-10 所示。

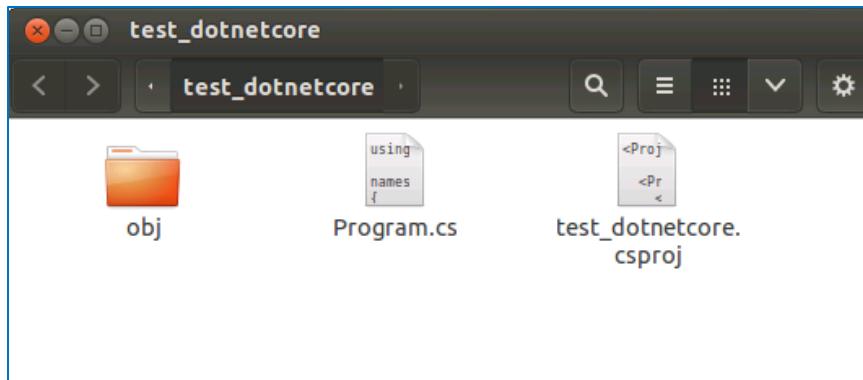


Figure 2-10

- 2) 将 "foxitpdfsdk_8_1_linux64_dotnetcore" 文件夹下的 "lib" 文件夹拷贝到 "test_dotnetcore" 工程目录下。
- 3) 拷贝一个 PDF 文件 (比如，"Sample.pdf") 到 "test_dotnetcore" 目录下，用于测试该工程。
- 4) 添加引用 "fsdk_dotnetcore.dll", "System.Drawing.Common" NuGet 包，以及添加 "libfsdk.so"。

编辑 **test_dotnetcore.csproj** 文件，添加如下的代码：(为了更好的阅读，我们将代码拷贝到 Visual Studio 中以便使用不同的颜色显示代码)

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>netcoreapp2.2</TargetFramework>
</PropertyGroup>
```

```
<ItemGroup>
  <PackageReference Include="System.Drawing.Common" Version="4.5.1" />
</ItemGroup>

<ItemGroup>
  <Reference Include="fsdk_dotnetcore">
    <HintPath>lib\fsdk_dotnetcore.dll</HintPath>
  </Reference>
</ItemGroup>

<ItemGroup>
  <FSdkLibSourceFiles Include="lib\libfsdk.so" />
</ItemGroup>

<Target Name="PreBuild" BeforeTargets="PreBuildEvent">
  <Copy SourceFiles="@{FSdkLibSourceFiles}" DestinationFolder="$(OutputPath)"
SkipUnchangedFiles="True" />
</Target>

<ItemGroup>
  <None Update="Sample.pdf">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>

</Project>
```

- 5) 在任意文本编辑器中打开 **Program.cs**, 添加如下的代码: (为了更好的阅读, 我们将代码拷贝到 Visual Studio 中以便使用不同的颜色显示代码)

```
using System;

using System.Drawing;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;

namespace test_dotnetcore
{
  class Program
  {
    static void Main(string[] args)
    {
      // 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=").
      string sn = " ";
      string key = " ";
      ErrorCode error_code = Library.Initialize(sn, key);
      if (error_code != ErrorCode.e_ErrSuccess)
      {
        return;
      }
    }
}
```

```

    }

    using (PDFDoc doc = new PDFDoc("Sample.pdf"))
    {
        error_code = doc.LoadW("");
        if (error_code != ErrorCode.e_ErrSuccess)
        {
            Library.Release();
            return;
        }

        using (PDFPage page = doc.GetPage(0))
        {
            // Parse page.
            page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);

            int width = (int)(page.GetWidth());
            int height = (int)(page.GetHeight());
            Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

            // Prepare a bitmap for rendering.
            foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height,
foxit.common.Bitmap.DIBFormat.e_DIBRgb32);
            System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();
            Graphics draw = Graphics.FromImage(sbitmap);
            draw.Clear(Color.White);

            // Render page
            Renderer render = new Renderer(bitmap, false);
            render.StartRender(page, matrix, null);

            // Add the bitmap to image and save the image.
            foxit.common.Image image = new foxit.common.Image();
            image.AddFrame(bitmap);
            image.SaveAs("testpage.jpg");
        }
    }
}

```

- 6) 运行工程。在终端窗口中，导航到 `test_dotnetcore` 目录，运行如下的命令：

```
dotnet run
```

然后，在"test_dotnetcore" 文件夹下将会生成 "testpage.jpg"。

备注：请检查 "`libfsdk.so`" 和 "`fsdk_dotnetcore.dll`" 是否拷贝至输出目录 ("`test_dotnetcore\bin\Debug\netcoreapp2.2`")。如果没有，您需要手动将动态库拷贝至该目录下。

2.4 Mac

在本手册中，请知晓：图片上面高亮的矩形区域指的是 SDK 的版本号，当前 SDK 的版本是 8.1，则其代表 8_1。

2.4.1 包结构说明

下载 Foxit PDF SDK for .NET Core (Mac for x86_64)包，解压到一个新的目录如 "foxitpdfsdk_8_1_mac_dotnetcore"，如 Figure 2-11 所示。其中解压包中包括如下的内容：

docs: API 手册，开发者指南

examples: 示例工程和 demos

lib: SDK 库和授权文件

res: 输出预览 (output preview) demo 使用的默认 icc profile 文件

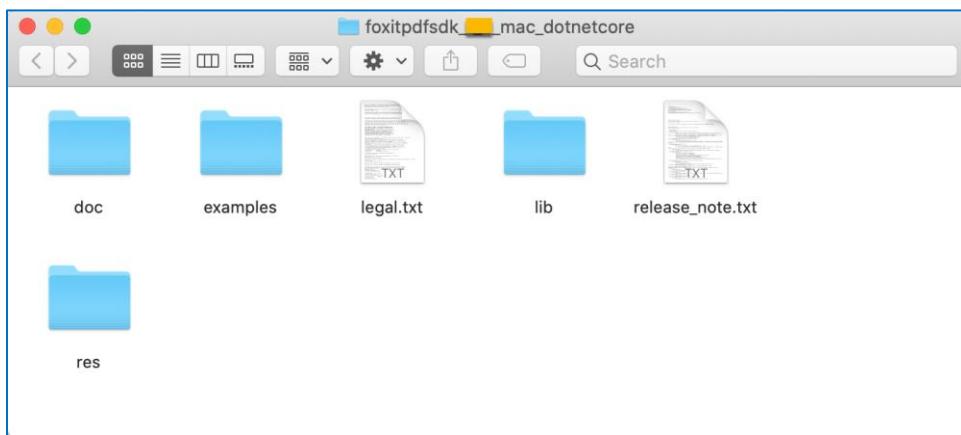


Figure 2-11

2.4.2 运行 demo

在运行 demo 之前，请确保您已经在 x86_64 位的 Mac 上正确配置了.NET Core 环境。

备注：Foxit PDF SDK for .NET Core demo 和下节中创建的工程都引用了 "System.Drawing.Bitmap"，因此请使用如下命令安装 "libgdiplus" 库：

```
brew search libgdiplus  
brew install mono-libgdiplus
```

Foxit PDF SDK for .NET Core (Mac for x86_64) 提供了一些示例 demo，在 "\examples\simple_demo" 目录下。所有的 demo (除了 compliance, html2pdf 和 output preview demo 以外) 都可以直接使用 "\examples\simple_demo" 目录下的 "RunDemo.sh" 文件在终端窗口中运行。

打开终端窗口，导航到 "\examples\simple_demo"，然后运行如下的命令：

- 输入 "**./RunDemo.sh all**" 运行所有的 demo。
- 输入 "**./RunDemo.sh demo_name**" 运行特定的单个 demo, 比如, "**./RunDemo.sh bookmark**" 只运行 bookmark demo。

"\examples\simple_demo\input_files" 文件夹下包含了所有 demo 使用的输入文件。对于会生成输出文件 (pdf, 文本或者图片文件) 的 demo, 会在 "\examples\simple_demo\output_files\" 文件夹下生成以该 demo 名称命名的文件夹, 并且输出文件将会在该文件夹下生成。

Compliance demo

对于 **compliance** demo, 您需要首先构建一个资源目录, 请联系 Foxit 支持团队或者销售团队获取相应的资源包。关于如何运行该 demo 的更详细的信息, 请参考 3.33 "[Compliance](#)"。

HTML to PDF demo

对于 **html2pdf** demo, 您需要首先联系 Foxit 支持团队或者销售团队获取 HTML 转 PDF 的引擎包。关于如何运行该 demo 的更详细的信息, 请参考 3.35 小节 "[HTML 转 PDF](#)".

Output Preview demo

对于 **output preview** demo, 您需要设置包含默认 icc profile 文件的文件夹路径。关于如何运行该 demo 的更详细信息, 请参考 3.37 小节 "[输出预览 \(Output Preview\)](#)"。

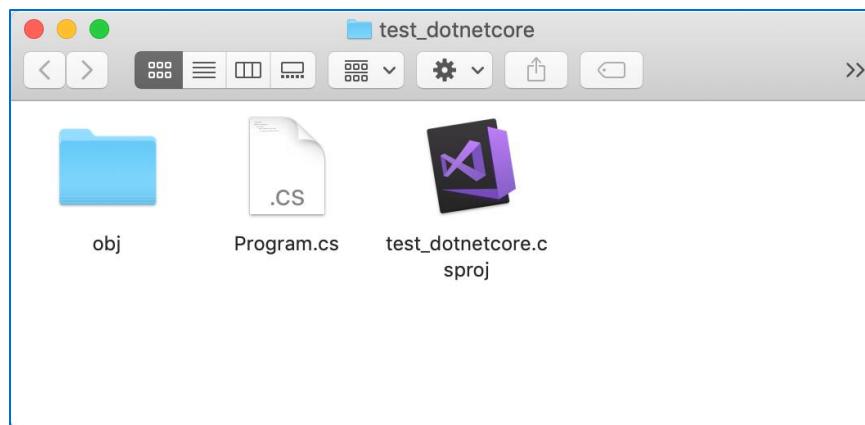
2.4.3 创建一个简单的工程

本节主要介绍如何使用 Foxit PDF SDK for .NET Core 创建一个简单的工程, 该工程将 PDF 文档的首页渲染成 bitmap, 然后将其另存为 JPG 图片。该工程将以.NET Core 2.2 为例。创建该工程, 请按照如下的步骤操作:

- 1) 创建一个名为 "test_dotnetcore" 的 C#语言的 .NET Core 控制台应用程序。打开一个终端窗口, 导航到您要创建工程的目录, 运行如下的命令:

```
dotnet new console -lang C# -o test_dotnetcore
```

然后, *test_dotnetcore* 工程将如 Figure 2-12 所示。

**Figure 2-12**

- 2) 将 "foxitpdfsdk_8_1_mac_dotnetcore" 文件夹下的 "lib" 文件夹拷贝到 "test_dotnetcore" 工程目录下。
- 3) 拷贝一个 PDF 文件 (比如, "Sample.pdf") 到 "test_dotnetcore" 目录下, 用于测试该工程。
- 4) 添加引用 "fsdk_dotnetcore.dll", "System.Drawing.Common" NuGet 包, 以及添加 "libfsdk.dylib"。

编辑 ***test_dotnetcore.csproj*** 文件, 添加如下的代码:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.2</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="System.Drawing.Common" Version="4.5.1" />
  </ItemGroup>

  <ItemGroup>
    <Reference Include="fsdk_dotnetcore">
      <HintPath>lib\fsdk_dotnetcore.dll</HintPath>
    </Reference>
  </ItemGroup>

  <ItemGroup>
    <FSdkLibSourceFiles Include="lib\libfsdk.dylib" />
  </ItemGroup>

  <Target Name="PreBuild" BeforeTargets="PreBuildEvent">
    <Copy SourceFiles="@{FSdkLibSourceFiles}" DestinationFolder="$(OutputPath)"
      SkipUnchangedFiles="True" />
  </Target>
```

```
<ItemGroup>
<None Update="Sample.pdf">
<CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
</None>
</ItemGroup>

</Project>
```

- 5) 打开 **Program.cs**, 添加如下的代码:

```
using System;

using System.Drawing;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;

namespace test_dotnetcore
{
    class Program
    {
        static void Main(string[] args)
        {

            // 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=").
            string sn = " ";
            string key = " ";
            ErrorCode error_code = Library.Initialize(sn, key);
            if (error_code != ErrorCode.e_ErrSuccess)
            {
                return;
            }

            using (PDFDoc doc = new PDFDoc("Sample.pdf"))
            {
                error_code = doc.LoadW("");
                if (error_code != ErrorCode.e_ErrSuccess)
                {
                    Library.Release();
                    return;
                }

                using (PDFPage page = doc.GetPage(0))
                {
                    // Parse page.
                    page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);

                    int width = (int)(page.GetWidth());
                    int height = (int)(page.GetHeight());
                    Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());
                }
            }
        }
    }
}
```

```
// Prepare a bitmap for rendering.  
foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height, foxit.common.Bitmap.DIBFormat.e_DIBRgb32);  
System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();  
Graphics draw = Graphics.FromImage(sbitmap);  
draw.Clear(Color.White);  
  
// Render page  
Renderer render = new Renderer(bitmap, false);  
render.StartRender(page, matrix, null);  
  
// Add the bitmap to image and save the image.  
foxit.common.Image image = new foxit.common.Image();  
image.AddFrame(bitmap);  
image.SaveAs("testpage.jpg");  
}  
}  
Library.Release();  
}  
}  
}
```

- 6) 运行工程。在终端窗口中，导航到 `test_dotnetcore` 目录，运行如下的命令：

dotnet run

然后，在"test_dotnetcore" 文件夹下将会生成 "testpage.jpg"。

备注：请检查 "libfsdk.dylib" 和 "fsdk_dotnetcore.dll" 是否拷贝至输出目录 ("test_dotnetcore\bin\Debug\netcoreapp2.2")。如果没有，您需要手动将动态库拷贝至该目录下。

2.5 创建一个可以根据平台切换库文件的简单工程

本节主要介绍如何创建一个可以根据平台切换库文件的跨平台的.NET Core 工程。和前面章节创建的工程一样，我们创建一个将 PDF 文档的首页渲染成 bitmap，然后将其另存为 JPG 图片的简单工程。该工程将以.NET Core 2.2 为例。创建该工程，请按照如下的步骤操作：

- 1) 创建一个名为 "test_autodotnetcore" 的 C# 语言的 .NET Core 控制台应用程序。打开一个命令行或者终端窗口，导航到您要创建工程的目录，运行如下的命令：

```
dotnet new console -lang C# -o test autodotnetcore
```

- 2) 在工程 "test_autodotnetcore" 文件夹下，创建一个 "lib" 文件夹，然后在 "lib" 文件夹下创建三个文件夹(比如，分别命名为 "win", "linux" 和 "osx")，用来存放指定平台的库文件。

 - 将 "foxitpdfsdk_8_1_win_dotnetcore" 包中 "lib" 目录下的文件夹拷贝到 "test_autodotnetcore/lib/win" 文件夹下。

- 将 "foxitpdfsdk_8_1_linux64_dotnetcore" 包中 "lib" 目录下的库文件拷贝到 "test_autodotnetcore/lib/linux" 文件夹下。
- 将 "foxitpdfsdk_8_1_mac_dotnetcore" 包中 "lib" 目录下的库文件拷贝到 "test_autodotnetcore/lib/osx" 文件夹下。
- 将任意平台包中的 "lib" 目录下的试用 license key 文件 "gsdk_key.txt" 和 "gsdk_sn.txt" 拷贝到 "test_autodotnetcore/lib" 文件夹下。

操作完成后, "test_autodotnetcore" 工程的目录结构将如 Figure 2-13 所示:

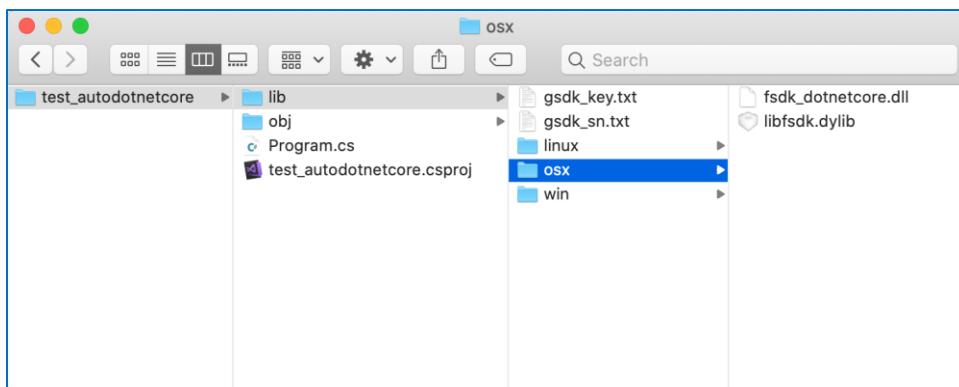


Figure 2-13

- 3) 拷贝一个 PDF 文件 (比如, "Sample.pdf") 到 "test_autodotnetcore" 目录下, 用于测试该工程。
- 4) 添加引用 "fsdk_dotnetcore.dll", "System.Drawing.Common" NuGet 包, 以及根据当前系统平台添加平台库文件, 比如 "libfsdk.dylib"。

编辑 **test_autodotnetcore.csproj** 文件, 添加如下的代码:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>netcoreapp2.2</TargetFramework>
  <AppendTargetFrameworkToOutputPath>False</AppendTargetFrameworkToOutputPath>
  <RuntimeIdentifiers>win-x86;win-x64;linux-x64;osx-x64</RuntimeIdentifiers>
</PropertyGroup>

<PropertyGroup Condition="$(Configuration)|$(Platform)=='Debug|AnyCPU'">
  <OutputPath>bin\</OutputPath>
</PropertyGroup>

<PropertyGroup Condition="$(Configuration)|$(Platform)=='Release|AnyCPU'">
  <OutputPath>bin\</OutputPath>
</PropertyGroup>
```

```
<!-- Include the platform library based on the platform -->
<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(Windows)\"">
  <Content Include="$(OutputPath)fsdk.dll" Link="fsdk.dll">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    <Pack>True</Pack>
    <PackagePath>lib\netstandard2.2</PackagePath>
  </Content>
</ItemGroup>

<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(Linux)\"">
  <Content Include="$(OutputPath)libfsdk.so" Link="libfsdk.so">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    <Pack>True</Pack>
    <PackagePath>lib\netstandard2.2</PackagePath>
  </Content>
</ItemGroup>

<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(OSX)\"">
  <Content Include="$(OutputPath)libfsdk.dylib" Link="libfsdk.dylib">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    <Pack>True</Pack>
    <PackagePath>lib\netstandard2.2</PackagePath>
  </Content>
</ItemGroup>

<!-- Reference the "System.Drawing.Common" Nuget package -->
<ItemGroup>
  <PackageReference Include="System.Drawing.Common" Version="4.5.1" />
</ItemGroup>

<!-- Remove the "lib" directory from the project, because -->
<!-- it is in the same directory with the "test_autodotnetcore.csproj" file -->
<ItemGroup>
  <None Remove="lib\**" />
</ItemGroup>

<!-- Reference the "fsdk_dotnetcore.dll" based on the platform -->
<ItemGroup>
  <Reference Include="fsdk_dotnetcore">
    <HintPath>$(OutputPath)fsdk_dotnetcore.dll</HintPath>
  </Reference>
</ItemGroup>

<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(Windows) And '$(Platform)'=='x86'\"">
  <FSdkLibSourceFiles Include="lib\win\x86_vc15\*fsdk*.*" />
</ItemGroup>

<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(Windows) And ('$(Platform)'=='AnyCPU' Or
'$Platform'=='x64')\"">
  <FSdkLibSourceFiles Include="lib\win\x64_vc15\*fsdk*.*" />
</ItemGroup>
```

```
<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(Linux)\">
  <FSdkLibSourceFiles Include="lib\linux\*fsdk*.*" />
</ItemGroup>

<ItemGroup Condition="\"$(MSBuild)::IsOsPlatform(OSX)\">
  <FSdkLibSourceFiles Include="lib\osx\*fsdk*.*" />
</ItemGroup>

<Target Name="PreBuild" BeforeTargets="PreBuildEvent">
  <Copy SourceFiles="@(&FSdkLibSourceFiles)" DestinationFolder="$(OutputPath)"
SkipUnchangedFiles="True" />
</Target>


<ItemGroup>
  <None Update="Sample.pdf">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>

</Project>
```

备注：如果 "lib" 文件夹和 "test_dotnetcore.csproj" 文件在同一个目录下，您需要添加如下的代码排除 "lib" 文件夹下的库文件。

```
<ItemGroup>
  <None Remove="lib\**" />
</ItemGroup>
```

5) 打开 **Program.cs**，添加如下的代码：

```
using System;

using System.Drawing;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;

namespace test_dotnetcore
{
  class Program
  {
    static void Main(string[] args)
    {

      // 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=").
      string sn = " ";
      string key = " ";
      ErrorCode error_code = Library.Initialize(sn, key);
      if (error_code != ErrorCode.e_ErrSuccess)
      {
```

```
        return;
    }

    using (PDFDoc doc = new PDFDoc("Sample.pdf"))
    {
        error_code = doc.LoadW("");
        if (error_code != ErrorCode.e_ErrSuccess)
        {
            Library.Release();
            return;
        }

        using (PDFPage page = doc.GetPage(0))
        {
            // Parse page.
            page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);

            int width = (int)(page.GetWidth());
            int height = (int)(page.GetHeight());
            Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

            // Prepare a bitmap for rendering.
            foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height, foxit.common.Bitmap.DIBFormat.e_DIBRgb32);
            System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();
            Graphics draw = Graphics.FromImage(sbitmap);
            draw.Clear(Color.White);

            // Render page
            Renderer render = new Renderer(bitmap, false);
            render.StartRender(page, matrix, null);

            // Add the bitmap to image and save the image.
            foxit.common.Image image = new foxit.common.Image();
            image.AddFrame(bitmap);
            image.SaveAs("testpage.jpg");
        }
        Library.Release();
    }
}
```

- 6) 运行工程。在命令行或者终端窗口，导航到 *test_autodotnetcore* 目录，输入如下的命令编译和运行工程：

编译 ***test_autodotnetcore.csproj***:

```
dotnet build test_autodotnetcore.csproj // 对于 64 位的 Windows, Mac, 和 Linux
```

备注：该工程默认使用 64 位的库。如果您需要编译 32 位的库，您可以使用命令行 "**dotnet build test_autodotnetcore.csproj -p:Platform=x86**"。

运行工程：

```
dotnet run
```

然后，在 "test_autodotnetcore" 或者 "test_autodotnetcore/bin" 文件夹下将会生成 "testpage.jpg"。

3 使用 SDK API

在本节中，我们将介绍 Foxit PDF SDK 的主要功能，并列举相关示例来展示如何将 Foxit PDF SDK for .NET Core 集成到 Windows、Linux 和 Mac 平台的工程中。您可以参阅 API reference^[2] 来获取示例中 APIs 更详细的使用说明。

3.1 初始化库

在调用任何 API 之前，都需要首先初始化 Foxit PDF SDK。`foxit.common.Library.Initialize` 用来初始化 Foxit PDF SDK，您需要购买正式的 license 来获取 license key 和序列号。当不再需要使用 Foxit PDF SDK 时，请调用 `foxit.common.Library.Release` 将其释放。

备注：参数 "sn" 的值在 "gsdk_sn.txt" 中 ("SN=" 后面的字符串)， "key" 的值在 "gsdk_key.txt" 中 ("Sign=" 后面的字符串)。

Example:

3.1.1 如何初始化 Foxit PDF SDK

```
using foxit.common;

string sn = "";
string key = "";
ErrorCode error_code = Library.Initialize(sn, key);
if (error_code != ErrorCode.e_ErrSuccess)
    return;
...
```

3.2 文档 (Document)

一个 PDF document 对象可以由一个已有的 PDF 文件从文件路径、内存缓冲区、自定义实现的 ReaderCallback 对象、输入文件流中构建。然后调用 `PDFDoc.Load` 或者 `PDFDoc.StartLoad` 加载文档内容。PDF document 对象用于文档级操作，比如打开和关闭 PDF 文档，获取页面、metadata 等。

Example:

3.2.1 如何从 0 开始创建一个 PDF 文档

```
using foxit.pdf;
...
PDFDoc doc = new PDFDoc();
```

备注：创建一个新的 PDF 文档，该文档没有任何页面。

3.2.2 如何通过文件路径加载一个现有的 PDF 文档

```
using foxit.pdf;
using foxit.common;
...
PDFDoc doc = new PDFDoc("Sample.pdf");
error_code = doc.Load(null);
if (error_code != ErrorCode.e_ErrSuccess) return;
...
```

3.2.3 如何通过内存缓冲区加载一个现有的 PDF 文档

```
using foxit.pdf;
using foxit.common;
...
byte[] byte_buffer = File.ReadAllBytes(input_file);
IntPtr buffer = System.Runtime.InteropServices.Marshal.AllocHGlobal(byte_buffer.Length);
try {
    System.Runtime.InteropServices.Marshal.Copy(byte_buffer, 0, buffer, byte_buffer.Length);
}
Finally {
    System.Runtime.InteropServices.Marshal.FreeHGlobal(buffer);
}
PDFDoc doc = new PDFDoc(buffer, (uint)byte_buffer.Length);
error_code = doc.Load(null);
if (error_code != ErrorCode.e_ErrSuccess) return;
...
```

3.2.4 如何通过自定义实现的 ReaderCallback 对象加载一个现有的 PDF 文档

```
using foxit.pdf;
using foxit.common;
...
class FileReader : FileReaderCallback
{
    private FileStream file_ = null;
    private long offset_ = 0;

    public FileReader(long offset)
    {
        this.offset_ = offset;
    }

    public Boolean LoadFile(String file_path)
    {
        file_ = new FileStream(file_path, FileMode.OpenOrCreate);
        return true;
    }
}
```

```
public override long GetSize()
{
    return this.offset_;
}

public override bool ReadBlock(IntPtr buffer, long offset, uint size)
{
    int read_size = 0;
    file_.Seek(offset, SeekOrigin.Begin);
    byte[] array = new byte[size + 1];
    read_size = file_.Read(array, 0, (int)size);
    Marshal.Copy(array, 0, buffer, (int)size);
    return read_size == size ? true : false;
}

public override void Release()
{
    this.file_.Close();
}
}

...

FileReader file_reader = new FileReader(offset);
file_reader.LoadFile(file_name);

PDFDoc doc_real = new PDFDoc(file_reader, false)

error_code = doc.Load(null);
if (error_code != ErrorCode.e_ErrSuccess) return;
...
```

3.2.5 如何加载 PDF 文档以及获取文档的首页

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

PDFDoc doc = new PDFDoc("Sample.pdf");
error_code = doc.Load(null);
if (error_code != ErrorCode.e_ErrSuccess) return;
...

// Get the first page of the document.
PDFPage page = doc.GetPage(0);
// Parse page.
page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);
...
```

3.2.6 如何将 PDF 文档另存为一个新的文档

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

```
PDFDoc doc = new PDFDoc("Sample.pdf");
error_code = doc.Load("");
if (error_code != ErrorCode.e_ErrSuccess) return;
...
// Do operations for the PDF document.
...
// Save the changes for the PDF document.
string newPdf = "the output path for the saved PDF";
doc.SaveAs(newPdf, (int)PDFDoc.SaveFlags.e_SaveFlagNoOriginal);
```

3.2.7 如何通过 **FileWriterCallback** 将 PDF 文档保存到内存缓冲区

```
using foxit.pdf;
using foxit.common;
using foxit.common.fxcr;
using System.IO;
using System.Runtime.InteropServices;
...

class FileWriter : FileWriterCallback
{
    private MemoryStream memoryfilestream = new MemoryStream();
    public FileWriter()
    {
    }

    public override long GetSize()
    {
        return this.memoryfilestream.Length;
    }

    public override bool WriteBlock(IntPtr pData, long offset, uint size)
    {
        byte[] ys = new byte[size];
        Marshal.Copy(pData, ys, 0, (int)size);
        memoryfilestream.Write(ys, 0, (int)size);
        return true;
    }

    public override bool Flush()
    {
        return true;
    }

    public override void Release()
    {
    }
}

FileWriter fileWriter = new FileWriter();
```

```
// Assuming PDFDoc doc has been loaded.  
...  
  
doc.StartSaveAs(fileWriter, (int) PDFDoc.SaveFlags.e_SaveFlagNoOriginal, null);  
...
```

3.3 页面 (Page)

PDF 页面是 PDF Document 基础和重要的组成部分。使用函数 `PDFDoc.GetPage` 从文档中获取 `PDFPage` 对象。页面级 API 提供了解析/渲染/编辑(包括创建、删除、扁平化等)页面、获取 PDF 注释、获取和设置页面属性等功能。对于大多数情况，在渲染和处理页面之前，需要先对页面进行解析。

Example:

3.3.1 如何获取页面的大小

```
using foxit.pdf;  
using foxit.common;  
...  
  
// Assuming PDFPage page has been loaded and parsed.  
...  
int width = (int)(page.GetWidth());  
int height = (int)(page.GetHeight());  
...
```

3.3.2 如何计算页面内容的边界框

```
using foxit.pdf;  
...  
  
// Assuming PDFPage page has been loaded and parsed.  
...  
  
RectF ret = page.CalcContentBBox(PDFPage.CalcMarginMode.e_CalcContentsBox);  
...
```

3.3.3 如何创建一个 PDF 页面以及设置其页面大小

```
using foxit.pdf;  
...  
  
// Assuming PDFDoc doc has been loaded.  
  
PDFPage page = doc.InsertPage(index, PageWidth, PageHeight);
```

3.3.4 如何删除一个 PDF 页面

```
using foxit.pdf;
```

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

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

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

3.3.5 如何扁平化一个 PDF 页面

```
using foxit.pdf;
...

// Assuming PDFPage page has been loaded and parsed.

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

// Flatten a PDF page without annotations.
page.Flatten(true, (int)PDFPage.FlattenOptions.e_FlattenNoAnnot);

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

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

3.3.6 如何获取和设置 PDF 文档中的页面缩略图

```
using foxit.pdf;
...

// Assuming PDFPage page has been loaded and parsed.

// Get page thumbnails.
page.LoadThumbnail();

// Set thumbnails to the page.
// Assuming Bitmap bitmap has been created.
page.SetThumbnail(bitmap);
...
```

3.4 渲染 (Render)

PDF 渲染是通过 Foxit 渲染引擎实现的，Foxit 渲染引擎是一个图形引擎，用于将页面渲染到位图或平台设备上下文。Foxit PDF SDK 提供了 APIs 用来设置渲染选项/flags，例如设置 flag 来决定是否渲染表单域和签名，是否绘制图像反锯齿 (anti-aliasing) 和路径反锯齿。可以使用以下 APIs 进行渲染：

- 渲染页面和注释时，首先使用 `Renderer.SetRenderContentFlags` 接口来决定是否同时渲染页面和注释，然后使用 `Renderer.StartRender` 接口进行渲染。`Renderer.StartQuickRender` 接口也可以用来渲染页面，但仅用于缩略图。
- 渲染单个 annotation 注释，使用 `Renderer.RenderAnnot` 接口。
- 在位图上渲染，使用 `Renderer.StartRenderBitmap` 接口。
- 渲染一个重排的页面，使用 `Renderer.StartRenderReflowPage` 接口。

在 Foxit PDF SDK 中，Widget 注释常与表单域和表单控件相关联。渲染 widget 注释，推荐使用如下的流程：

- 加载 PDF 页面后，首先渲染页面以及该页面上所有的注释 (包括 widget 注释)。
- 然后，如果使用 `pdf.interform.Filler` 对象来填表，则应使用 `pdf.interform.Filler.Render` 接口来渲染当前获取到焦点的表单控件，而不是使用 `Renderer.RenderAnnot` 接口。

Example:

3.4.1 如何将 PDF 页面渲染到 bitmap

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;

// Assuming PDFPage page has been loaded and parsed.

int width = (int)(page.GetWidth());
int height = (int)(page.GetHeight());
Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

// Prepare a bitmap for rendering.
foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height,
foxit.common.Bitmap.DIBFormat.e_DIBRgb32);
System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();
Graphics draw = Graphics.FromImage(sbitmap);
draw.Clear(Color.White);

// Render page.
Renderer render = new Renderer(bitmap, false);
render.StartRender(page, matrix, null);
...
```

3.4.2 如何渲染页面和注释

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;

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

int width = (int)(page.GetWidth());
int height = (int)(page.GetHeight());
Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

// Prepare a bitmap for rendering.
foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height,
foxit.common.Bitmap.DIBFormat.e_DIBRgb32);
System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();
Graphics draw = Graphics.FromImage(sbitmap);
draw.Clear(Color.White);

// Render page
Renderer render = new Renderer(bitmap, false);
render.SetRenderContentFlags((int)Renderer.ContentFlag.e_RenderAnnot |
(int)Renderer.ContentFlag.e_RenderPage);
render.StartRender(page, matrix, null);
...
```

3.5 附件 (Attachment)

在 Foxit PDF SDK 中，attachments 指的是文档附件而不是文件附件注释。它允许将整个文件封装在文档中，就像电子邮件附件一样。Foxit PDF SDK 提供应用程序 APIs 来访问附件，例如加载附件，获取附件，插入/删除附件，以及访问附件的属性。

Example:

3.5.1 如何向 PDF 中插入附件文件

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.objects;

// Assuming PDFDoc doc has been loaded.

string text_path = "The input path of the attached file you need to insert";
Attachments attachments = new Attachments(doc, new PDFNameTree());
attachment.AddFromFilePath("OriginalAttachmentsInfo", text_path);
...
```

3.5.2 如何删除 PDF 中指定的附件文件

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.objects;

// Assuming PDFDoc doc has been loaded.
...
Attachments attachment = new Attachments(doc, new PDFNameTree());
string strKey = attachment.GetKey(index);
attachment.RemoveEmbeddedFile(strKey);
...
```

3.6 文本页面 (Text Page)

Foxit PDF SDK 提供 APIs 来提取，选择，搜索和检索 PDF 文档中的文本。PDF 文本内容存储在与特定页面相关的 **TextPage** 对象中。**TextPage** 类可用于获取 PDF 页面中文本的信息，例如单个字符，单个单词，指定字符范围或矩形内的文本内容等。它还可用于构造其他文本相关类的对象，用来对文本内容执行更多操作或从文本内容访问指定信息：

- 在 PDF 页面的文本内容中搜索文本，使用 **TextPage** 对象来构建 **TextSearch** 对象。
- 访问类似超文本链接的文本，使用 **TextPage** 对象来构建 **PageTextLinks** 对象。

Example:

3.6.1 如何从 PDF 页面中提取文本

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

// Assuming PDFPage page has been loaded and parsed.

using (var text_page = new TextPage(page, (int)TextPage.TextParseFlags.e_ParseTextNormal))
{
    int count = text_page.GetCharCount();
    if (count > 0)
    {
        String chars = text_page.GetChars(0, count);
        writer.Write(chars);
    }
}
...
```

3.6.2 如何在 PDF 文档中获取矩形区域中的文本

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

```

using foxit.common.fxcr;
...
RectF rect = new RectF(100, 50, 220, 100);
TextPage text_page = new TextPage(page, (int)foxit.pdf.TextPage.TextParseFlags.e_ParseTextNormal);
String str_text = text_page.GetTextInRect(rect);
...

```

3.7 文本搜索 (Text Search)

Foxit PDF SDK 提供 APIs 来搜索 PDF 文档、XFA 文档、文本页面或者 PDF 注释中的文本。它提供了文本搜索和获取搜索结果的函数：

- 指定搜索模式和选项，使用 `TextSearch.SetPattern`、`TextSearch.SetStartPage` (仅对 PDF 文档中的文本搜索有用)、`TextSearch.SetEndPage` (仅对 PDF 文档中的文本搜索有用)、和 `TextSearch.SetSearchFlags` 接口。
- 进行搜索，使用 `TextSearch.FindNext` 和 `TextSearch.FindPrev` 接口。
- 获取搜索结果，使用 `TextSearch.GetMatchXXX()` 接口。

Example:

3.7.1 如何在 PDF 文档中搜索指定的文本

```

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

using (TextSearch search = new TextSearch(doc, null, (int)TextPage.TextParseFlags.e_ParseTextNormal))
{
    int start_index = 0;
    int end_index = doc.GetPageCount() - 1;
    search.SetStartPage(0);
    search.SetEndPage(doc.GetPageCount() - 1);

    String pattern = "Foxit";
    search.SetPattern(pattern);

    Int32 flags = (int)TextSearch.SearchFlags.e_SearchNormal;
    search.SetSearchFlags(flags);
    int match_count = 0;
    while (search.FindNext())
    {
        RectFArray rect_array = search.GetMatchRects();
        match_count++;
    }
}

```

3.8 文本链接 (Text Link)

在 PDF 页面中，指向网站、网络资源以及电子邮件地址的超链接文本和普通文本一样。在处理文本链接之前，用户应首先调用 [PageTextLinks.GetTextLink](#) 接口来获取一个 `textlink` 对象。

Example:

3.8.1 如何检索 PDF 页面中的超链接

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

// Get the text page object.
TextPage text_page = new TextPage(page, (int)foxit.pdf.TextPage.TextParseFlags.e_ParseTextNormal);
PageTextLinks page_textlinks = new PageTextLinks(text_page);
TextLink text_link = page_textlinks.GetTextLink(index); // specify an index.
string str_url = text_link.GetURI();
...
```

3.9 书签 (Bookmark)

Foxit PDF SDK 提供了名为书签的导航工具，允许用户在 PDF 文档中快速定位和链接他们感兴趣的的部分。PDF 书签也称为大纲 (outline)，每个书签包含一个目标位置或动作来描述它链接到的位置。它是一个树形的层次结构，因此在访问 `bookmark` 树之前，必须首先调用接口 [pdf.PDFDoc.GetRootBookmark](#) 以获取整个 `bookmark` 树的根节点。这里，“书签根节点”是一个抽象对象，它只有一些子节点，没有兄弟节点，也没有任何数据 (包括 `bookmark` 数据，目标位置数据和动作数据)。因为它没有任何数据，因此无法在应用程序界面上显示，能够调用的接口只有 [Bookmark.GetFirstChild](#)。

在获取书签根节点后，就可以调用以下的接口去访问其他的书签：

- 访问 parent bookmark，使用 [Bookmark.GetParent](#) 接口。
- 访问第一个 child bookmark，使用 [Bookmark.GetFirstChild](#) 接口。
- 访问 next sibling bookmark，使用 [Bookmark.GetNextSibling](#) 接口。
- 插入一个新的 bookmark，使用 [Bookmark.Insert](#) 接口。
- 移动一个 bookmark，使用 [Bookmark.MoveTo](#) 接口。

Example:

3.9.1 如何遍历 PDF 文档中所有的书签

```
using foxit;
using foxit.common;
```

```
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.actions;
...

//Assuming PDFDoc doc has been loaded.
...
Bookmark root = doc.GetRootBookmark();
Bookmark first_bookmark = root.GetFirstChild();
if (first_bookmark != null)
{
    TraverseBookmark(first_bookmark, 0);
}

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

3.9.2 如何向 PDF 文档中插入一个新的书签

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.actions;

// Assuming PDFDoc doc has been loaded.

Bookmark root = doc.GetRootBookmark();
if (root.IsEmpty())
{
    root = doc.CreateRootBookmark();
}
using (Destination dest = Destination.CreateFitPage(doc, 0))
{
    string ws_title = string.Format("A bookmark to a page (index: {0})", 0);

    Bookmark child;
    using (child = root.Insert(ws_title, Bookmark.Position.e_PosLastChild))
    {
        child.SetDestination(dest);
        child.SetColor(0xF68C21);
```

```
}
```

3.10 表单 (AcroForm)

PDF 目前支持两种类型的 form，用于以交互方式收集用户信息：AcroForms 和 XFA forms。

Acroforms 是基于 PDF 框架的原始的可填写表单。Foxit PDF SDK 提供了以编程方式查看和编辑表单域的 APIs。在 PDF 文档中，表单域通常用于收集数据。[Form](#) 类提供了 APIs 用来获取表单域或表单控件，导入/导出表单数据，以及其他功能，例如：

- 获取表单域，使用 [Form.GetFieldCount](#) 和 [Form.GetField](#) 接口。
- 获取 PDF 页面中的表单控件，使用 [Form.GetControlCount](#) 和 [Form.GetControl](#) 接口。
- 从 XML 文件导入表单数据，使用 [Form.ImportFromXML](#) 接口；导出表单数据到 XML 文件，使用 [Form.ExportToXML](#) 接口。
- 获取 form filler 对象，使用 [Form.GetFormFiller](#) 接口。

从FDF/XFDF文件中导入表单数据，或者导出数据到FDF/XFDF文件，请参考
[pdf.PDFDoc.ImportFromFDF](#) 和 [pdf.PDFDoc.ExportToFDF](#)接口。

Example:

3.10.1 如何加载 PDF 中的表单

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
...
// Assuming PDFDoc doc has been loaded.

Boolean hasForm = doc.HasForm();
If(hasForm)
    Form form = new Form(doc);
...
```

3.10.2 如何获取表单域个数以及设置其属性

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
```

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

Boolean hasForm = doc.HasForm();
If(hasForm)
    Form form = new Form(doc);
int count = form.GetFieldCount("");
for (int i = 0; i < count; i++)
{
    Field field = form.GetField(i, "");
    ...
}
```

3.10.3 如何将 PDF 中的表单数据导出到 XML 文件

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
...

// Assuming PDFDoc doc has been loaded.

Boolean hasForm = doc.HasForm();
If(hasForm)
    Form form = new Form(doc);
...
form.ExportToXML(XMLFilePath);
...
```

3.10.4 如何通过 XML 文件导入表单数据到 PDF

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
...

// Assuming PDFDoc doc has been loaded.

Boolean hasForm = doc.HasForm();
If(hasForm)
    Form form = new Form(doc);
...
form.ImportFromXML(XMLFilePath);
```

...

3.10.5 如何获取和设置表单域的属性

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
...
// Assuming PDFDoc doc has been loaded.

Boolean hasForm = doc.HasForm();
If(hasForm)
    Form form = new Form(doc);
Field field = form.GetField(0, "Text Field0");
field.GetAlignment();
field.GetAlternateName();
field.SetAlignment(Alignment.e_AlignmentLeft);
field.SetValue("3");
...
```

3.11 XFA 表单

XFA (XML Forms Architecture) forms 是基于 XML 的表单，封装在 PDF 内。XFA 提供了基于模板的语法和一系列处理规则，允许用户构建交互式表单。最简单的来说，基于模板的语法定义了用户数据的字段。

Foxit PDF SDK 提供了 APIs 用来渲染 XFA 表单、填表、导出和导入表单数据。

备注:

- Foxit PDF SDK 提供了两个回调类 `foxit.addon.xfa.AppProviderCallback` 和 `foxit.addon.xfa.DocProviderCallback`，分别将回调对象通过 `Library.RegisterXFAppProviderCallback` 以及 `XFADoc` 的构造函数注册到 SDK 中。这两个类中的所有函数都是纯虚函数，需要用户自己实现。
- 使用 XFA form 功能，请确保授权 key 文件中包含 'XFA' 的权限。

Example:

3.11.1 如何加载 XFADoc 并且显示 XFA 交互式表单

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
```

```
using foxit.addon;
using foxit.addon.xfa;
...
// Implement from AppProviderCallback
CFS_XFAAppHandler pXFAAppHandler = new CFS_XFAAppHandler();
Library.RegisterXFAAppProviderCallback(pXFAAppHandler);
string input_file = input_path + "xfa_dynamic.pdf";
using (PDFDoc doc = new PDFDoc(input_file))
{
    error_code = doc.Load(null);
    if(error_code != ErrorCode.e_ErrSuccess)
    {
        Console.WriteLine("The PDFDoc [{0}] Error: {1}\n", input_file, error_code);
        Library.Release();
        return;
    }

    // Implement from DocProviderCallback
    CFS_XFADocHandler pXFADocHandler = new CFS_XFADocHandler();
    using (XFADoc xfa_doc = new XFADoc(doc, pXFADocHandler))
    {
        ...
    }
}
```

3.11.2 如何导出和导入 XFA 表单数据

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.addon;
using foxit.addon.xfa;
...
// Assuming FSXFADoc xfa_doc has been loaded.

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

xfa_doc.ResetForm();
doc.SaveAs("xfa_dynamic_resetform.pdf", (int)foxit.pdf.PDFDoc.SaveFlags.e_SaveFlagNormal);

xfa_doc.ImportData(output_path + "xfa_form.xml");
doc.SaveAs("xfa_dynamic_importdata.pdf", (int)foxit.pdf.PDFDoc.SaveFlags.e_SaveFlagNormal);
...
```

3.12 表单设计 (Form Design)

可填写的 PDF 表单 (AcroForm) 特别适用于各种应用程序表单设计，比如税收和其他政府部门表单。表单设计提供了 APIs 用来向 PDF 文件中添加表单域或者从 PDF 文档中移除表单域。从零开始设计一个表单允许开发人员创建他们需要的内容和布局的表单。

Example:

3.12.1 如何向 PDF 添加一个文本表单域

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
...
// Assuming PDFDoc doc has been loaded.

Control control = form.AddControl(page, "Text Field0", Field.Type.e_TypeTextField, new RectF(50f, 600f, 90f,
640f))
...
```

3.12.2 如何从 PDF 中移除一个文本表单域

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.interform;
using foxit.pdf.annots;
using foxit.pdf.actions;
...
// Assuming PDFDoc doc has been loaded.

Field field = form.GetField(0, "Text Field0");
form.RemoveField(field);
.....
```

3.13 注释 (Annotations)

3.13.1 常规注释

一个 annotation 注释将对象（如注释，线条和高亮）与 PDF 文档页面上的位置相关联。其提供了一种通过鼠标和键盘与用户进行交互的方式。PDF 包括如 Table 3-1 中列出的各种标准注释类型。在这些注释类型中，许多被定义为标记注释，因为它们主要用于标记 PDF 文档。标记注释中作为其自身一

部分的文本，可以在其他符合标准的阅读器中以其他方式显示，例如在 Comments 面板。Table 3-1 中的“Markup”列用来说明是否为标记注释。

Foxit PDF SDK 支持 PDF Reference^[1] 中定义的大多数注释类型。Foxit PDF SDK 提供了注释创建，属性访问和修改，外观设置和绘制的 APIs。

Table 3-1

注释类型	描述	Markup	SDK 是否支持
Text(Note)	Text annotation	Yes	Yes
Link	Link Annotations	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

备注：

1. Widget 和 watermark 注释类型是比较特殊的。'Annotation' 模块不支持它们。Widget 类型仅在 'form filler' 模块中使用，watermark 类型仅在 'watermark' 模块中使用。
2. Foxit SDK 支持名为 PSI (pressure sensitive ink, 压感笔迹) 的自定义注释类型。在 PDF Reference [1] 中没有对该注释进行描述。通常，PSI 用于手写签名功能，Foxit SDK 将其视为 PSI 注释，以便其他 PDF 产品可以对其进行相关处理。

Example:

3.13.1.1 如何向 PDF 页面中添加 link 注释

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.annots;
using foxit.pdf.actions;
...
// Assuming PDFPage page has been loaded and parsed.

Annot annot = page.AddAnnot(Annot.Type.e_Link, new RectF(350, 350, 380, 400))
Link link = new Link(annot);
link.SetHighlightingMode(Annot.HighlightingMode.e_HighlightingToggle);

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

3.13.1.2 如何向 PDF 页面中添加 highlight 注释，并且设置相关属性

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.annots;
using foxit.pdf.actions;
...
// Assuming PDFPage page has been loaded and parsed.

// Add highlight annotation.
Annot annot = page.AddAnnot(Annot.Type.e_Highlight, new RectF(10, 450, 100, 550));
Highlight highlight = new Highlight(annot);
highlight.SetContent("Highlight");
QuadPoints quad_points = new QuadPoints();
quad_points.first = new foxit.common.fxcrt.PointF(10, 500);
quad_points.second = new foxit.common.fxcrt.PointF(90, 500);
quad_points.third = new foxit.common.fxcrt.PointF(10, 480);
quad_points.fourth = new foxit.common.fxcrt.PointF(90, 480);
```

```
QuadPointsArray quad_points_array = new QuadPointsArray();
quad_points_array.Add(quad_points);
highlight.SetQuadPoints(quad_points_array);
highlight.SetSubject("Highlight");
highlightSetTitle("Foxit SDK");
highlight.SetCreationDateTime(GetLocalDateTime());
highlight.SetModifiedDateTime(GetLocalDateTime());
highlight.SetUniqueID(RandomUID());

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

3.13.1.3 如何在创建 *markup* 注释时设置 *popup* 信息

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.annots;
using foxit.pdf.actions;
...

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

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

// Add popup to note annotation.
Annot annot_popup = page.AddAnnot(Annot.Type.e_Popup, new RectF(300, 450, 500, 550));
Popup popup = new Popup(annot);
popup.SetBorderColor(0x00FF00);
popup.SetOpenStatus(false);
popup.SetModifiedDateTime(GetLocalDateTime());
note.SetPopup(popup);

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

3.13.1.4 如何使用设备坐标获取 PDF 中特定的注释

```
using foxit;
using foxit.common;
```

```
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.annots;
...

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

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

Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());

// Assuming PointF point has been got.

float tolerance = 3.0f;

Annot annot = page.GetAnnotAtDevicePoint(point, tolerance, matrix);
...
```

3.13.1.5 如何提取 *text markup annotation* 中的文本内容

```
using foxit.common;
using foxit.pdf;
using foxit.pdf.annots;
...

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

using (var page = doc.GetPage(0))
{
    // Parse the first page.
    page.StartParse((int) PDFPage.ParseFlags.e_ParsePageNormal, null, false);
    // Get a TextPage object.
    using (var text_page = new TextPage(page, (int) TextPage.TextParseFlags.e_ParseTextNormal))
    {
        int annot_count = page.GetAnnotCount();
        for (int i = 0; i < annot_count; i++)
        {
            Annot annot = page.GetAnnot(i);
            TextMarkup text_markup = new TextMarkup(annot);
            if (!text_markup.IsEmpty())
            {
                // Get the texts which intersect with a text markup annotation.
                string text = text_page.GetTextUnderAnnot(text_markup);
            }
        }
    }
}
```

3.13.1.6 如何为freetext 注释添加richtext

```
using foxit.common;
using foxit.pdf;
using foxit.pdf.annots;

// 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 freetext = null;
Annot annot = null;
using (annot = pdf_page.AddAnnot(Annot.Type.e_FreeText, new RectF(50, 50, 150, 100)))
using (freetext = new FreeText(annot))
{
    // Set annotation's properties.

    // Add/insert richtext string with style.
    using (RichTextStyle richtext_style = new RichTextStyle())
    {
        {
            String font_name = "Times New Roman";
            using (richtext_style.font = new foxit.common.Font(font_name, 0,
foxit.common.Font.Charset.e_CharsetANSI, 0))
            {
                richtext_style.text_color = 0xFF0000;
                richtext_style.text_size = 10;
                freetext.AddRichText("Textbox annotation ", richtext_style);

                richtext_style.text_color = 0x00FF00;
                richtext_style.is_underline = true;
                freetext.AddRichText("1-underline ", richtext_style);
            }
        }
        {
            String font_name = "Calibri";
            using (richtext_style.font = new foxit.common.Font(font_name, 0,
foxit.common.Font.Charset.e_CharsetANSI, 0))
            {
                richtext_style.text_color = 0x0000FF;
                richtext_style.is_underline = false;
                richtext_style.is_strikethrough = true;
                int richtext_count = freetext.GetRichTextCount();
                freetext.InsertRichText(richtext_count - 1, "2_strikethrough ", richtext_style);
            }
        }
    }

    // Appearance should be reset.
    freetext.resetAppearanceStream();
}
```

3.13.2 从 FDF 文件导入注释或者将注释导出到 FDF 文件

在 Foxit PDF SDK 中，可以使用来自应用程序或者 FDF 文件的数据来创建注释。同时，PDF SDK 支持将注释导出到 FDF 文件。

Example:

3.13.2.1 如何从 FDF 文件导入注释，并将其添加到 PDF 文档的首页

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;

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

string fdf_file = "The FDF file path";
foxit.fdf.FDFDoc fdf_doc = new foxit.fdf.FDFDoc(fdf_file);
pdf_doc.ImportFromFDF(fdf_doc, (int)foxit.pdf.PDFDoc.DataType.e_Annots, new Range());
...
```

3.14 图片转换 (Image Conversion)

Foxit PDF SDK 提供了 PDF 文件和图片之间进行转换的 APIs。应用程序可以轻松地实现图片创建和图片转换等功能，支持如下的图片格式：BMP、TIFF、PNG、JPX、JPEG 和 GIF。通过 Foxit PDF SDK，PDF 文件和支持的图片格式 (除了 GIF) 之间可以互相转换。Foxit PDF SDK 只支持将 GIF 图片转换为 PDF 文件。

Example:

3.14.1 如何将 PDF 页面转换为位图文件

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using System.Drawing;

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

Image image = new Image();

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

```
// Parse page.  
page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);  
  
int width = (int)(page.GetWidth());  
int height = (int)(page.GetHeight());  
Matrix2D matrix = page.GetDisplayMatrix(0, 0, width, height, page.GetRotation());  
  
// Prepare a bitmap for rendering.  
foxit.common.Bitmap bitmap = new foxit.common.Bitmap(width, height,  
foxit.common.Bitmap.DIBFormat.e_DIBRgb32);  
System.Drawing.Bitmap sbitmap = bitmap.GetSystemBitmap();  
Graphics draw = Graphics.FromImage(sbitmap);  
draw.Clear(Color.White);  
  
// Render page  
Renderer render = new Renderer(bitmap, false);  
render.StartRender(page, matrix, null);  
image.AddFrame(bitmap);  
}  
}  
...
```

3.14.2 如何将图片转换为 PDF 文件

```
using foxit;  
using foxit.common;  
using foxit.common.fxcr;  
using foxit.pdf;  
using System.Drawing;  
  
// Assuming PDFDoc doc has been loaded.  
...  
Image image = new Image(input_file);  
int count = image.GetFrameCount();  
  
for (int i = 0; i < count; i++) {  
    PDFPage page = doc.InsertPage(i, PDFPage.Size.e_SizeLetter);  
    page.StartParse((int)PDFPage.ParseFlags.e_ParsePageNormal, null, false);  
  
    // Add image to page  
    page.AddImage(image, i, new PointF(0, 0), page.GetWidth(), page.GetHeight(), true);  
}  
  
doc.SaveAs(output_file, (int)PDFDoc.SaveFlags.e_SaveFlagNoOriginal);  
...
```

3.15 水印 (Watermark)

水印是一种 PDF 注释，广泛用于 PDF 文档。水印是文档上嵌入的可见叠加层，包含文本、logo 或版权声明。水印的目的是对作者工作成果的保护，防止其未经授权而被他人使用。Foxit PDF SDK 提供了允许应用程序创建、插入和删除水印的 APIs。

Example:

3.15.1 如何创建一个文本水印，并将其插入到 PDF 文档的第一页

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

WatermarkSettings settings = new WatermarkSettings();
settings.offset_x = 0;
settings.offset_y = 0;
settings.opacity = 90;
settings.position = Position.e_PosTopRight;
settings.rotation = -45.0f;
settings.scale_x = 1.0f;
settings.scale_y = 1.0f;

WatermarkTextProperties text_properties = new WatermarkTextProperties();
text_properties.alignment = Alignment.e_AlignmentCenter;
text_properties.color = 0xF68C21;
text_properties.font_style = WatermarkTextProperties.FontStyle.e_FontStyleNormal;
text_properties.line_space = 1;
text_properties.font_size = 12.0f;
text_properties.font = font;

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

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

3.15.2 如何创建一个图片水印，并将其插入到 PDF 文档的第一页

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

WatermarkSettings settings = new WatermarkSettings();
settings.flags = (int)(WatermarkSettings.Flags.e_FlagASPageContents | WatermarkSettings.Flags.e_FlagOnTop);
settings.offset_x = 0.0f;
settings.offset_y = 0.0f;
settings.opacity = 20;
settings.position = Position.e_PosCenter;
settings.rotation = 0.0f;

Image image = new Image(image_file);
```

```
foxit.common.Bitmap bitmap = image.GetFrameBitmap(0);

settings.scale_x = page.GetWidth() * 0.618f / bitmap.GetWidth();
settings.scale_y = settings.scale_x;

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

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

3.15.3 如何从 PDF 页面中删除所有的水印

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

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

3.16 条形码 (Barcode)

条形码用于表示与某个对象相关联的数据，该数据可通过光学机器进行读取。最初的条形码系统是通过平行线间宽度和间距的不同来表示数据，可称为线性条形码或一维条形码(1D)。后来条形码逐渐演变成矩形、点、六边形等 2D 几何图案。虽然 2D 系统使用了一系列符号，但是它们通常也被称为条形码。条形码最初由特定的光学扫描器进行扫描，该光学扫描器被称为条形码读取器。后来扫描器和解释软件成功应用于桌面打印机和智能手机等设备。Foxit PDF SDK 提供了从给定字符串生成条形码位图的应用程序。Table 3-2 列出了 Foxit PDF SDK 支持的条形码类型。

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:

3.16.1 如何从字符串生成条形码位图

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
...

// Strings used as barcode content.
```

```

String sz_code_str = "TEST-SHEET";

// Barcode format types.
Barcode.Format sz_code_format = Barcode.Format.e_FormatCode39;

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

//Image names for the saved image files for QR code.
String sz_bmp_qr_name = "/QR_CODE_TestForBarcodeQrCode_L.bmp";

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

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

Barcode barcode;
Barcode barcode = new Barcode();

foxit.common.Bitmap bitmap = barcode.GenerateBitmap(sz_code_str, sz_code_format, unit_width, unit_height,
qr_level);
...

```

3.17 安全 (Security)

Foxit PDF SDK 提供了一系列加密和解密功能，以满足不同级别的文档安全保护。用户可以使用常规密码加密和证书驱动加密，或使用自己的安全处理机制来自定义安全实现。另外，Foxit PDF SDK 还提供了 APIs 用于集成第三方安全技术 (Microsoft RMS)，允许开发人员使用 Microsoft RMS SDK 加密和解密 PDF 文档。

备注：有关RMS加密和解密更详细的信息，请参考SDK包中 "*\examples\simple_demo*" 文件夹下的 "**security**" demo。

Example:

3.17.1 如何使用用户密码 "123" 和所有者密码 "456" 加密 PDF 文档

```

using foxit.common;
using foxit.pdf;
...
using (var handler = new StdSecurityHandler())
{
    using (var encrypt_data = new StdEncryptData(true, -4, SecurityHandler.CipherType.e_CipherAES, 16))
    {
        handler.Initialize(encrypt_data, "123", "456");
        doc.SetSecurityHandler(handler);

        doc.SaveAs(output_file, (int)PDFDoc.SaveFlags.e_SaveFlagNormal);
    }
}

```

```
}
```

```
...
```

3.17.2 如何使用证书加密 PDF 文件

```
using foxit.common;
using foxit.pdf;
...
PDFDoc doc = new PDFDoc(input_file);
ErrorCode error_code = doc.Load(null);
if (error_code != ErrorCode.e_ErrSuccess){
    return;
}

// Do encryption.
string cert_file_path = input_path + "foxit.cer";

var cms = new EnvelopedCms(new ContentInfo(seed));
cms.ContentEncryptionAlgorithm.Oid.Value = "1.2.840.113549.3.4";
cms.Encrypt(new CmsRecipient(new X509Certificate2(cert_file_path)));
byte[] bytes = cms.Encode();

byte[] data = new byte[20 + bytes.Length];
Array.Copy(seed, 0, data, 0, 20);
Array.Copy(bytes, 0, data, 20, bytes.Length);
SHA1 sha1 = new SHA1CryptoServiceProvider();
byte[] initial_key = new byte[16];
Array.Copy(sha1.ComputeHash(data), initial_key, initial_key.Length);

StringArray string_array = new StringArray();
string_array.AddBytes(bytes);
using (var handler = new CertificateSecurityHandler())
{
    using (var encrypt_data = new CertificateEncryptData())
    {
        encrypt_data.Set(true, SecurityHandler.CipherType.e_CipherAES,string_array);
        handler.Initialize(encrypt_data, initial_key);

        doc.SetSecurityHandler(handler);
        doc.SaveAs(output_file, (int)PDFDoc.SaveFlags.e_SaveFlagNormal);
    }
}
...
...
```

3.17.3 如何使用 Foxit DRM 加密 PDF 文件

```
using foxit.common;
using foxit.pdf;
...
PDFDoc doc = new PDFDoc(input_file);
ErrorCode error_code = doc.Load(null);
if (error_code != ErrorCode.e_ErrSuccess){
```

```

    return;
}

// Do encryption.
var handler = new DRMSecurityHandler();
var encrypt_data = new DRMEncryptData(true,"Simple-DRM-filter", SecurityHandler.CipherType.e_CipherAES,
16, true, -4);
string file_id = "Simple-DRM-file-ID";
string initialize_key = "Simple-DRM-initialize-key";
handler.Initialize(encrypt_data, file_id, initialize_key);
doc.SetSecurityHandler(handler);
doc.SaveAs(output_file, (int)PDFDoc.SaveFlags.e_SaveFlagNormal);
...

```

3.18 页面重排 (Reflow)

页面重排功能是在页面大小发生变化时自动重排页面内容。该功能对那些需要在不同尺寸的输出设备上显示 PDF 文档的应用程序具有很大的利用价值。页面重排让应用程序无需考虑设备的不同尺寸。Foxit PDF SDK 提供了 APIs 用来创建、渲染、释放 Reflow 页面，以及访问 Reflow 页面的属性。

Example:

3.18.1 如何创建一个 Reflow 页面，并将其渲染为位图文件

```

using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
...

// Assuming PDFDoc doc has been loaded.

PDFPage page = doc.GetPage(0);

// Parse PDF page.
page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);

ReflowPage reflow_page = new ReflowPage(page);

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

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

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

```

```
float content_height = reflow_page.GetContentHeight();

// Assuming Bitmap bitmap has been created.

// Render reflow page.
Renderer renderer = new Renderer(bitmap, false);
Matrix2D matrix = reflow_page.GetDisplayMatrix(0, 0);
renderer.StartRenderReflowPage(reflow_page, matrix, null);
...
```

3.19 异步加载 PDF (Asynchronous PDF)

异步加载 PDF 技术是一种在文档加载需要花费很长时间时，可以不用加载整个文档就可以对 PDF 页面进行访问的方法。该方法专为访问互联网上的 PDF 文件而设计。使用异步 PDF 技术，应用程序无需等待下载整个 PDF 文件就可以对其进行访问，可以打开任何已经下载加载的 PDF 页面。该技术为 Web 阅读类的应用程序提供了一种方便和有效的方式。关于如何使用异步模式打开和解析 PDF 页面，请参考 SDK 包中 "\examples\simple_demo" 文件夹下的 "**async_load**" demo。

3.20 压感笔迹 (Pressure Sensitive Ink)

压感笔迹 (PSI) 是一种获取变化电力输出以响应作用于压力感应设备元件上的各种变化压力或受力的技术。在 PDF 中，PSI 通常被用于手写签名，通过捕捉手指或触控笔的压力变化来收集 PSI 数据。PSI 数据包含操作区域的坐标和画布，并用其来绘制 PSI 的外观。Foxit PDF SDK 允许应用程序创建 PSI、访问其属性、操作 ink 笔迹和画布、以及释放 PSI。

Example:

3.20.1 如何创建 PSI 并设置相关属性

```
using foxit;
using foxit.common;
using foxit.common.fxcrf;
using foxit.pdf;
using foxit.pdf.annots;
...

PSI psi = new PSI(480, 180, true);

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

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

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

// Add points to pressure sensitive ink.
```

```
float x = 121.3043f;
float y = 326.6846f;
float pressure = 0.0966f;
Path.PointType type = Path.PointType.e_TypeMoveTo;
foxit.common.fxcrt.PointF pt = new foxit.common.fxcrt.PointF(x, y);
psi.AddPoint(pt, type, pressure);
...
```

3.21 Wrapper

Wrapper 为用户提供了一种保存与 PDF 文档相关的数据的方法。例如，在打开一个加密未授权的 PDF 文档，用户会看到错误信息提示其没有权限访问该文档。在这种情况下，使用 wrapper，用户即使无法访问 PDF 中的内容，但仍然可以访问该文档的 wrapper 数据。Wrapper 数据可用来提供信息给用户，比如文档的解密方法。

Example:

3.21.1 如何打开包含 wrapper 数据的 PDF 文档

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.objects;

// Assuming PDFDoc doc has been loaded.

// file_name is PDF document which includes wrapper data.
PDFDoc doc = new PDFDoc(file_name);
ErrorCode code = doc.Load(null);
if (code != ErrorCode.e_ErrSuccess)
{
    return;
}
if (!doc.IsWrapper())
{
    return;
}

long offset = doc.GetWrapperOffset();
FileReader file_reader = new FileReader(offset);
file_reader.LoadFile(file_name);
...
```

3.22 PDF 对象 (PDF Objects)

PDF 中有八种类型的对象：布尔对象、数字对象、字符串对象、名称对象、数组对象、字典对象、流对象和空对象。PDF 对象是文档级文档，与页面对象（见 3.23）不同，每个页面对象都与特定的页面相关联。Foxit PDF SDK 提供了 APIs 用来创建、修改、检索和删除文档中的这些对象。

Example:

3.22.1 如何从目录字典中删除指定的属性

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.objects;
...
// Assuming PDFDoc doc has been loaded.

PDFDictionary doc_catalog_dict = doc.GetCatalog();
String[] key_strings = { "Type", "Boolean", "Name", "String", "Array", "Dict" };
int count = key_strings.Length;
for (int i = 0; i < count; i++)
{
    if (catalog.HasKey(key_strings[i]))
    {
        catalog.RemoveAt(key_strings[i]);
    }
}
...
...
```

3.23 页面对象 (Page Object)

页面对象可以帮助对 PDF 对象 (关于 PDF 对象更详细的介绍，见 3.22) 知识了解有限的开发人员能够处理 PDF 文档中的文本、路径、图像和画布等对象。Foxit PDF SDK 提供 APIs 用以在页面中添加和删除 PDF 对象并设置特定属性。使用页面对象，用户可以从对象内容创建 PDF 页面。页面对象的其他可能用法包括向 PDF 文档添加页眉和页脚，向每个页面添加图片 logo，或者根据需要生成 PDF 模板。

Example:

3.23.1 如何在 PDF 页面中创建一个文本对象

```
using foxit.common;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.common.fxcr;
```

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

long position = page.GetLastGraphicsObjectPosition(GraphicsObject.Type.e_TypeText);
TextObject text_object = TextObject.Create();
text_object.SetFillColor(0xFFFF7F00);

// Prepare text state.
TextState state = new TextState();
Font font = new Font("Simsun", (int)FontStyles.e_StylesSmallCap, Font.Charset.e_CharsetGB2312, 0);
state.font_size = 80.0f;
state.font = font;
state.textmode = TextState.Mode.e_ModeFill;
text_object.SetTextState(page, state, false, 750);

// Set text.
text_object.SetText("Foxit Software");
long last_position = page.InsertGraphicsObject(position, text_object);
...
```

3.23.2 如何向 PDF 页面中插入一个图片 logo

```
using foxit.common;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.common.fx crt;
...

// Assuming PDFPage page has been loaded and parsed.

long position = page.GetLastGraphicsObjectPosition(GraphicsObject.Type.e_TypeImage);
Image image = new Image(image_file);
ImageObject image_object = ImageObject.Create(page.GetDocument());
image_object.SetImage(image, 0);

float width = image.GetWidth();
float height = image.GetHeight();
float page_width = page.GetWidth();
float page_height = page.GetHeight();

image_object.SetMatrix(new Matrix2D(width, 0, 0, height, (page_width - width) / 2.0f, (page_height - height) / 2.0f));
page.InsertGraphicsObject(position, image_object);
page.GenerateContent();
...
```

3.24 标记内容 (Marked content)

在 PDF 文档中，可以将一部分内容标记为标记内容元素。标记内容功能有助于管理 PDF 文档的逻辑结构信息并且可以用于生成加标记的 PDF (tagged PDF)。加标记的 PDF 具有标准的结构类型和属

性，有助于提取和再利用页面内容。有关标记内容的更多详细信息，请参阅 PDF reference 1.7 [1] 的第 10.5 章。

Example:

3.24.1 如何获取页面中的标记内容以及 tag 名称

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.pdf.objects;

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

long position = page.GetFirstGraphicsObjectPosition(GraphicsObject.Type.e_TypeText);
GraphicsObject text_obj = page.GetGraphicsObject(position);
MarkedContent content = text_obj.GetMarkedContent();

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

3.25 PDF 图层 (PDF Layer)

Foxit PDF SDK 支持 PDF 图层，也称为可选内容组 (Optional Content Groups, OCG)。用户可以选择性地查看或隐藏多图层 PDF 文档的不同层中的内容。多图层广泛用于许多应用领域，如 CAD 制图、地图、分层艺术品以及多语言文档等。

在 Foxit PDF SDK 中，PDF 图层与图层节点相关联。要获取图层节点，用户应首先构建 PDF `LayerTree` 对象，然后调用函数 `LayerTree.GetRootNode` 以获取整个图层树的根图层节点。另外，您可以从根图层节点开始枚举图层树中的所有节点。Foxit PDF SDK 提供 APIs 用来获取/设置图层数据，查看或隐藏不同图层中的内容，设置图层名称，添加或删除图层，以及编辑图层。

Example:

3.25.1 如何创建一个 PDF 图层

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
```

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

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

3.25.2 如何设置所有图层节点的信息

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

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

LayerTree layertree = new LayerTree(doc);
LayerNode root = layertree.GetRootNode();
if (root.IsEmpty())
{
    return;
}
SetAllLayerNodesInformation(root);
...
```

3.25.3 如何编辑图层树

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

LayerTree layertree = new LayerTree(doc);
LayerNode root = layertree.GetRootNode();
int children_count = root.GetChildrenCount();
root.RemoveChild(children_count - 1);
LayerNode child = root.GetChild(children_count - 2);
LayerNode child0 = root.GetChild(0);
child.MoveTo(child0, 0);
child.AddChild(0, "AddedLayerNode", true);
child.AddChild(0, "AddedNode", false);
```

3.26 签名 (Signature)

PDF 签名可用于创建和签署 PDF 文档的数字签名，从而保护文档内容的安全性并避免文档被恶意篡改。它可以让接收者确保其收到的文档是由签名者发送的，并且文档内容是完整和未被篡改的。Foxit PDF SDK 提供 APIs 用来创建数字签名，验证签名的有效性，删除现有的数字签名，获取和设置数字签名的属性，显示签名和自定义签名表单域的外观。

备注： Foxit PDF SDK 提供了默认签名回调函数，支持如下两种类型的 *signature filter* 和 *subfilter*:

- (1) *filter*: *Adobe.PPKLite* *subfilter*: *adbe.pkcs7.detached*
- (2) *filter*: *Adobe.PPKLite* *subfilter*: *adbe.pkcs7.sha1*

如果您使用以上任意一种的 *signature filter* 和 *subfilter*，您可以直接签名 PDF 文档和验证签名的有效性，而不需要注册自定义回调函数。

Example:

3.26.1 如何对 PDF 文档进行签名

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using foxit.common;
using foxit.pdf;
using foxit;
using foxit.pdf.annots;
using foxit.common.fxcr;
using System.Runtime.InteropServices;
```

```
using foxit.pdf.interform;

static foxit.common.DateTime GetLocalDateTime()
{
    System.DateTimeOffset rime = System.DateTimeOffset.Now;
    foxit.common.DateTime datetime = new foxit.common.DateTime();
    datetime.year = (UInt16)rime.Year;
    datetime.month = (UInt16)rime.Month;
    datetime.day = (ushort)rime.Day;
    datetime.hour = (UInt16)rime.Hour;
    datetime.minute = (UInt16)rime.Minute;
    datetime.second = (UInt16)rime.Second;
    datetime.utc_hour_offset = (short)rime.Offset.Hours;
    datetime.utc_minute_offset = (ushort)rime.Offset.Minutes;
    return datetime;
}

static Signature AddSignature(PDFPage pdf_page, string sub_filter) {
    float page_height = pdf_page.GetHeight();
    float page_width = pdf_page.GetWidth();
    RectF new_sig_rect = new RectF(0, (float)(page_height*0.9), (float)(page_width*0.4), page_height);
    // Add a new signature to page.
    Signature new_sig = pdf_page.AddSignature(new_sig_rect);
    if (new_sig.IsEmpty()) return null;
    // Set values for the new signature.
    new_sig.SetValue(Signature.KeyName.e_KeyNameSigner, "Foxit PDF SDK");
    String new_value = String.Format("As a sample for subfilter \"\{0\}\\"", sub_filter);
    new_sig.SetValue(Signature.KeyName.e_KeyNameReason, new_value);
    new_sig.SetValue(Signature.KeyName.e_KeyNameContactInfo, "support@foxitsoftware.com");
    new_sig.SetValue(Signature.KeyName.e_KeyNameDN, "CN=CN,MAIL=MAIL@MAIL.COM");
    new_sig.SetValue(Signature.KeyName.e_KeyNameLocation, "Fuzhou, China");
    new_value = String.Format("As a sample for subfilter \"\{0\}\\"", sub_filter);
    new_sig.SetValue(Signature.KeyName.e_KeyNameText, new_value);
    foxit.common.DateTime sign_time = GetLocalDateTime();
    new_sig.SetSignTime(sign_time);
    String image_file_path = input_path + "FoxitLogo.jpg";
    using (Image image = new Image(image_file_path))
    {
        new_sig.SetImage(image, 0);
        // Set appearance flags to decide which content would be used in appearance.
        int ap_flags = Convert.ToInt32(Signature.APFlags.e_APFlagLabel | Signature.APFlags.e_APFlagSigner |
            Signature.APFlags.e_APFlagReason | Signature.APFlags.e_APFlagDN |
            Signature.APFlags.e_APFlagLocation | Signature.APFlags.e_APFlagText |
            Signature.APFlags.e_APFlagSigningTime | Signature.APFlags.e_APFlagBitmap);
        new_sig.SetAppearanceFlags(ap_flags);
    }
    return new_sig;
}

static void AdobePPKLiteSignature(PDFDoc pdf_doc) {
    string filter = "Adobe.PPKLite";
```

```

string sub_filter = "adbe.pkcs7.detached";

using (PDFPage pdf_page = pdf_doc.GetPage(0))
{
    // Add a new signature to first page.
    using (Signature new_signature = AddSignature(pdf_page, sub_filter))
    {
        // Set filter and subfilter for the new signature.
        new_signature.SetFilter(filter);
        new_signature.SetSubFilter(sub_filter);

        // Sign the new signature.
        String signed_pdf_path = output_directory + "signed_newsignature.pdf";

        String cert_file_path = input_path + "foxit_all.pfx";
        byte[] cert_file_password = Encoding.ASCII.GetBytes("123456");
        new_signature.StartSign(cert_file_path, cert_file_password,
            Signature.DigestAlgorithm.e_DigestSHA1, signed_pdf_path, IntPtr.Zero, null);
        Console.WriteLine("[Sign] Finished!");
    }
}

static void Main(String[] args)
{
    ...
    AdobePPKLiteSignature(pdf_doc);
    ...
}

```

3.27 长期签名验证(LTV, Long term validation)

从 7.0 版本开始，Foxit PDF SDK 提供了 API 接口进行长期签名验证，主要用于解决已经过期的签名的验证问题。LTV 需要 DSS(Document Security Store)，其包含了签名的验证信息，以及需要文档时间戳签名 DTS (Document Timestamp Signature)，其是 time stamp 类型的 signature。

为了支持 LTV，Foxit PDF SDK 提供了：

- 支持添加 time stamp 类型的 signature，以及提供了 sub filter "ETSI.RFC3161" 的默认签名回调。
- TimeStampServerMgr 和 TimeStampServer 类，用于 time stamp 的 server 设置和管理等。sub filter "ETSI.RFC3161" 的默认签名回调将会使用默认的 time stamp server。
- LTVerifier 类，其提供了验证签名和向文档中添加 DSS 信息的功能。同时，也提供了 LTVerifier 所需的一个基本默认的回调函数 RevocationCallback。

以下仅以使用 SDK 默认的 sub filter "ETSI.RFC3161" 签名回调及默认的 RevocationCallback 为例来说明如何进行长期签名验证。有关更详细的信息，请参阅下载包中 "\examples\simple_demo" 目录下的 "**Itv**" demo。

Example:

3.27.1 如何使用 SDK 默认的 sub filter "ETSI.RFC3161" 签名回调及默认的 RevocationCallback 进行长期签名验证

```
using System;

using foxit.common;
using foxit.pdf;
using foxit;
using foxit.common.fxcr;

// Initialize time stamp server manager, add and set a default time stamp server, which will be used by default
signature callback for time stamp signature.
TimeStampServerMgr.Initialize();
using (TimeStampServer timestamp_server = TimeStampServerMgr.AddServer(server_name, server_url,
server_username, server_password))
{
    TimeStampServerMgr.SetDefaultServer(timestamp_server);
    // Assume that "signed_pdf_path" represents a signed PDF document which contains signed signature.
    using (PDFDoc pdf_doc = new PDFDoc(signed_pdf_path))
    {
        pdf_doc.StartLoad(null, true, null);
        // Use LTVVerifier to verify and add DSS.
        using (LTVVerifier ltv_verifier = new LTVVerifier(pdf_doc, true, true, false,
LTVVerifier.TimeType.e_SignatureCreationTime))
        {
            // Set verifying mode which is necessary.
            ltv_verifier.SetVerifyMode(LTVVerifier.VerifyMode.e_VerifyModeAcrobat);
            SignatureVerifyResultArray sig_verify_result_array = ltv_verifier.Verify();
            for (uint i = 0; i < sig_verify_result_array.GetSize(); i++)
            {
                // Itv state would be e_LTVStateNotEnable here.
                SignatureVerifyResult.LTVState ltv_state = sig_verify_result_array.GetAt(i).GetLTVState();
                if ((sig_verify_result_array.GetAt(i).GetSignatureState() &
Convert.ToInt32(Signature.States.e_StateVerifyValid)) > 0)
                    ltv_verifier.AddDSS(sig_verify_result_array.GetAt(i));
            }
            // Add a time stamp signature as DTS and sign it. "saved_ltv_pdf_path" represents the newly saved signed
PDF file.
            using (PDFPage pdf_page = pdf_doc.GetPage(0))
            {
                // The new time stamp signature will have default filter name "Adobe.PPKLite" and default subfilter name
"ETSI.RFC3161".
                Signature timestamp_signature = pdf_page.AddSignature(new RectF(), "", 
Signature.SignatureType.e_SignatureTypeTimeStamp, true);
                byte[] empty_byte = Encoding.ASCII.GetBytes("");
            }
        }
    }
}
```

```

Progressive sign_progressive = timestamp_signature.StartSign("", empty_byte,
Signature.DigestAlgorithm.e_DigestSHA256,
saved_ltv_pdf_path, IntPtr.Zero, null);
if (sign_progressive.GetRateOfProgress() != 100)
sign_progressive.Continue();

// Then use LTVVerifier to verify the new signed PDF file.
using (PDFDoc check_pdf_doc = new PDFDoc(saved_ltv_pdf_path))
{
check_pdf_doc.StartLoad(null, true, null);
// Use LTVVerifier to verify.
using (LTVVerifier ltv_verifier = new LTVVerifier(pdf_doc, true, true, false,
LTVVerifier.TimeType.e_SignatureCreationTime))
{
// Set verifying mode which is necessary.
ltv_verifier.SetVerifyMode(LTVVerifier.VerifyMode.e_VerifyModeAcrobat);
SignatureVerifyResultArray sig_verify_result_array = ltv_verifier.Verify();
for (uint i = 0; i < sig_verify_result_array.GetSize(); i++)
{
// ltv state would be e_LTVStateEnable here.
SignatureVerifyResult.LTVState ltv_state = sig_verify_result_array.GetAt(i).GetLTVState();
... // User can get other information from SignatureVerifyResult.
}
}
}
}
}
}

// Release time stamp server manager when everything is done.
TimeStampServerMgr.Release();

```

3.28 PAdES

从 7.0 版本开始，Foxit PDF SDK 支持 PAdES (PDF Advanced Electronic Signature)，其是 CAdES 签名在 PDF 中的应用。CAdES 是高级数字签名的一种新标准，其默认 subfilter 是 "ETSI.CAdES.detached"。PAdES 签名分为四个等级：B-B, B-T, B-LT, 和 B-LTA。

- B-B: 包含基本的必须出现的属性。
- B-T: 在 B-B 的基础上，包含文档时间戳或者签名时间戳，来为存在的签名提供可信的时间。
- B-LT: 在 B-T 的基础上，包含 DSS/VRI，来提供证书和吊销信息。
- B-LTA: 在 B-LT 的基础上，为存在的吊销信息提供可信时间 DTS。

Foxit PDF SDK 提供了 subfilter 为 "ETSI.CAdES.detached" 的默认签名回调，可用来签名和验证 subfilter 为 "ETSI.CAdES.detached" 的签名。还提供了 TimeStampServerMgr 和 TimeStampServer 类，用于设置和管理 time stamp server。subfilter "ETSI.CAdES.detached" 的默认签名回调将会使用默认的 time stamp server。

Foxit PDF SDK 提供了从签名中获取 PAdES 不同等级的方法，应用层面也可以根据各个等级的要求来判定所属等级。

3.29 PDF 行为 (PDF Action)

PDF Action 代表 PDF 操作类的基类。Foxit PDF SDK 提供了 APIs 用来创建一系列行为，并获取行为句柄，比如 embedded goto action, JavaScript action, named action 和 launch action 等。

Example:

3.29.1 如何创建一个 URI 行为并将其插入到 link 注释

```
using foxit.common;
using foxit.pdf;
using foxit;
using foxit.pdf.annots;
using foxit.common.fxcrt;
using foxit.pdf.annots;
using foxit.pdf.actions;

// Add link annotation
Link link = null;
Annot annot = null;
using (annot = page.AddAnnot(Annot.Type.e_Link, new RectF(350, 350, 380, 400)))
using (link = new Link(annot))
using (PDFDoc doc = page.GetDocument())
{
    foxit.pdf.actions.Action action = null;
    link.SetHighlightingMode(Annot.HighlightingMode.e_HighlightingToggle);

    // Add action for link annotation
    UriAction uriaction = null;
    using (action = foxit.pdf.actions.Action.Create(doc, foxit.pdf.actions.Action.Type.e_TypeGoto))
    using (uriaction = new UriAction(action))
    {
        uriaction.SetTrackPositionFlag(true);
        uriaction.SetURI("www.foxitsoftware.com");
        link.SetAction(uriaction);

        // Appearance should be reset.
        link.ResetAppearanceStream();
    }
}
```

3.29.2 如何创建一个 GoTo 行为并将其插入到 link 注释

```
using foxit.common;
using foxit.pdf;
using foxit;
using foxit.pdf.annots;
using foxit.common.fxcrt;
```

```
using foxit.pdf.annots;
using foxit.pdf.actions;

using foxit.common;
using foxit.pdf;
using foxit;
using foxit.pdf.annots;
using foxit.common.fxcr;
using foxit.pdf.annots;
using foxit.pdf.actions;

Link link = null;
Annot annot = null;
using (annot = page.AddAnnot(Annot.Type.e_Link, new RectF(350, 350, 380, 400)))
using (link = new Link(annot))
using (PDFDoc doc = page.GetDocument())
{
    foxit.pdf.actions.Action action = null;
    link.SetHighlightingMode(Annot.HighlightingMode.e_HighlightingToggle);

    // Add action for link annotation
    GotoAction gotoaction = null;
    using (action = foxit.pdf.actions.Action.Create(doc, foxit.pdf.actions.Action.Type.e_TypeGoto))
    using (gotoaction = new GotoAction(action))
    {
        Destination dest = Destination.CreateXYZ(doc, 0, 0, 0, 0);
        gotoaction.SetDestination(dest);
        link.SetAction(gotoaction);
        // Appearance should be reset.
        link.ResetAppearanceStream();
    }
}
```

3.30 JavaScript

创建 JavaScript 是为了将 Web 页面的相关处理从服务器转移到基于 Web 的应用程序的客户端上。Foxit PDF SDK JavaScript 以 JavaScript 语言的形式实现新对象及其附带方法和属性的扩展。其使开发人员能够管理文档安全性，与数据库通信，处理文件附件以及操作 PDF 文件，因此其表现为交互式、web 表单等。

JavaScript action 是一种由 JavaScript 解释器编译和执行脚本的动作。类 `foxit.pdf.actions.JavaScriptAction` 派生自 `Action`，并提供接口用来获取/设置 JavaScript action 数据。

在[附录](#)中可以查看 Foxit PDF SDK 支持的 JavaScript 方法和属性列表。

Example:

3.30.1 如何添加文档级的 JavaScript 动作

```
using foxit.pdf.annots;
using foxit.pdf.actions;

// Load Document doc.
...
using (foxit.pdf.actions.Action action = foxit.pdf.actions.Action.Create(doc,
foxit.pdf.actions.Action.Type.e_TypeJavaScript))
using (JavaScriptAction javascript_action = new JavaScriptAction(action))
{
    javascript_action.SetScript("app.alert(\"Hello Foxit \");");
    using (AdditionalAction aa = new AdditionalAction(doc))
    {
        aa.setAction(AdditionalAction.TriggerEvent.e_TriggerDocWillClose, javascript_action);
        aa.DoJSAction(AdditionalAction.TriggerEvent.e_TriggerDocWillClose);
    }
}
...
...
```

3.30.2 如何添加注释级的 JavaScript 动作

```
using foxit.pdf.annots;
using foxit.pdf.actions;

...
// Load Document and get a widget annotation.
...
using (foxit.pdf.actions.Action action = foxit.pdf.actions.Action.Create(page.GetDocument(),
foxit.pdf.actions.Action.Type.e_TypeJavaScript))
using (JavaScriptAction javascript_action = new JavaScriptAction(action))
{
    javascript_action.SetScript("app.alert(\"Hello Foxit \");");
    using (AdditionalAction aa = new AdditionalAction(annot))
    {
        aa.setAction(AdditionalAction.TriggerEvent.e_TriggerAnnotMouseButtonPressed, javascript_action);
        aa.DoJSAction(AdditionalAction.TriggerEvent.e_TriggerAnnotMouseButtonPressed);
    }
}
...
...
```

3.30.3 如何添加表单级的 JavaScript 动作

```
using foxit.pdf.interform;
using foxit.pdf.actions;

...
// Load Document and get a form field.
...
// Add text field.
```

```
using (Control control = form.AddControl(page, "Text Field0", Field.Type.e_TypeTextField, new RectF(50f, 600f, 90f, 640f)))
using (Control control1 = form.AddControl(page, "Text Field1", Field.Type.e_TypeTextField, new RectF(100f, 600f, 140f, 640f)))
using (Control control2 = form.AddControl(page, "Text Field2", Field.Type.e_TypeTextField, new RectF(150f, 600f, 190f, 640f)))
using (Field field = control.GetField())
{
    field.SetValue("3");
    // Update text field's appearance.
    using (Widget widget = control.GetWidget())
        widget.ResetAppearanceStream();
    using (Field field1 = control1.GetField())
    {
        field1.SetValue("23");
        // Update text field's appearance.
        using (Widget widget1 = control1.GetWidget())
            widget1.ResetAppearanceStream();

        using (Field field2 = control2.GetField())
        using (PDFDoc doc = form.GetDocument())
        using (foxit.pdf.actions.Action action = foxit.pdf.actions.Action.Create(form.GetDocument(),
foxit.pdf.actions.Action.Type.e_TypeJavaScript))
            using (foxit.pdf.actions.JavaScriptAction javascript_action = new foxit.pdf.actions.JavaScriptAction(action))
            {
                javascript_action.SetScript("AFSimple_Calculate(\"SUM\", new Array (\"Text Field0\", \"Text Field1\");)");
                using (AdditionalAction aa = new AdditionalAction(field2))
                {
                    aa.SetAction(AdditionalAction.TriggerEvent.e_TriggerFieldRecalculateValue, javascript_action);
                    // Update text field's appearance.
                    using (Widget widget2 = control2.GetWidget())
                        widget2.ResetAppearanceStream();
                }
            }
        }
    }
}
...
...
```

3.30.4 如何使用 JavaScript 向 PDF 页面添加一个新的注释

```
using foxit.pdf.annots;
using foxit.pdf.actions;
...

// Load Document and get form field, construct a Form object and a Filler object.
...

using (foxit.pdf.actions.Action action = foxit.pdf.actions.Action.Create(form.GetDocument(),
foxit.pdf.actions.Action.Type.e_TypeJavaScript))
using (foxit.pdf.actions.JavaScriptAction javascript_action = new foxit.pdf.actions.JavaScriptAction(action))
{
```

```
javascript_action.SetScript("var annot = this.addAnnot({ page : 0, type : \"Square\", rect : [ 0, 0, 100, 100 ],  
name : \"UniqueID\", author : \"A. C. Robat\", contents : \"This section needs revision.\" });");  
using (AdditionalAction aa = new AdditionalAction(field))  
{  
    aa.setAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter, javascript_action);  
    aa.DoJSAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter);  
}  
}  
...
```

3.30.5 如何使用 JavaScript 获取/设置注释的属性 (strokeColor, fillColor, readOnly, rect, type 等)

```
using foxit.pdf.annots;  
using foxit.pdf.actions;  
...  
  
// Load Document and get form field, construct a Form object and a Filler object.  
...  
  
using (foxit.pdf.actions.Action action = foxit.pdf.actions.Action.Create(form.GetDocument(),  
foxit.pdf.actions.Action.Type.e_TypeJavaScript))  
  
// Get properties of annotations.  
using (foxit.pdf.actions.JavaScriptAction javascript_action = new foxit.pdf.actions.JavaScriptAction(action))  
{  
    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);}");  
    using (AdditionalAction aa = new AdditionalAction(field))  
    {  
        aa.setAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter, javascript_action);  
        aa.DoJSAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter);  
    }  
}  
  
// Set properties of annotations (only take strokeColor as an example).  
using (foxit.pdf.actions.JavaScriptAction javascript_action1 = new foxit.pdf.actions.JavaScriptAction(action))  
{  
    javascript_action1.SetScript("var ann = this.getAnnot(0, \"UniqueID\");if (ann != null) { ann.strokeColor =  
color.blue; }");  
    using (AdditionalAction aa1 = new AdditionalAction(field1))  
    {  
        aa1.setAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter, javascript_action1);  
        aa1.DoJSAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter);  
    }  
}  
...
```

3.30.6 如何使用 JavaScript 销毁注释

```
using foxit.pdf.annots;
```

```

using foxit.pdf.actions;
...
// Load Document and get form field, construct a Form object and a Filler object.
...
using (foxit.pdf.actions.Action action = foxit.pdf.actions.Action.Create(form.GetDocument(),
foxit.pdf.actions.Action.Type.e_TypeJavaScript))
using (foxit.pdf.actions.JavaScriptAction javascript_action = new foxit.pdf.actions.JavaScriptAction(action))
{
    javascript_action.SetScript("var ann = this.getAnnot(0, \" UniqueID \"); if (ann != null) { ann.destroy(); } ");
    using (AdditionalAction aa = new AdditionalAction(field))
    {
        aa.setAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter, javascript_action);
        aa.DoJSAction(AdditionalAction.TriggerEvent.e_TriggerAnnotCursorEnter);
    }
}
...

```

3.31 密文 (Redaction)

密文是一种在保持文档布局的同时删除文档中敏感信息的功能。它可以帮助用户永久删除 PDF 文档中的可见文本和图片，以保护一些保密信息，如社会安全号码、信用卡信息、产品发布日期等等。

密文是一种标记注释，用于标记 PDF 文件的某些内容，标记的内容在注释被应用后会被删除。

执行密文，您可以使用如下的 APIs:

- 调用 `foxit.addon.Redaction.Redaction` 创建一个 redaction 模块。如果在函数 `common.Library.Initialize` 中使用的 license 授权信息没有定义"Redaction"，则表示用户没有权限使用 redaction 相关的函数，并且构造函数会抛出 `foxit.common.ErrorCode.e_ErrInvalidLicense` 异常。
- 然后调用 `foxit.addon.Redaction.MarkRedactAnnot` 创建一个 redaction 对象，对需要进行 redaction 的页面内容 (文本对象、图片对象和路径对象) 进行标记。
- 最后调用 `foxit.addon.Redaction.Apply` 在标记区域应用 redaction: 永久删除标记区域的文本和图形。

备注: 要使用 redaction 功能，请确保授权 key 文件中包含 'Redaction' 模块。

Example:

3.31.1 如何将 PDF 文档第一页中的文本 "PDF" 设置为密文

```

using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.addon;

```

```

using foxit.pdf.annots;

...
using (Redaction redaction = new Redaction(doc))
{
    using (PDFPage page = doc.GetPage(0))
    {
        // Parse PDF page.
        page.StartParse((int)foxit.pdf.PDFPage.ParseFlags.e_ParsePageNormal, null, false);
        TextPage text_page = new TextPage(page, (int)foxit.pdf.TextPage.TextParseFlags.e_ParseTextNormal);
        TextSearch text_search = new TextSearch(text_page);
        text_search.SetPattern("PDF");
        RectFArray rect_array = new RectFArray();
        while (text_search.FindNext())
        {
            RectFArray itemArray = text_search.GetMatchRects();
            rect_array.InsertAt(rect_array.GetSize(), itemArray);
        }
        if (rect_array.GetSize() > 0)
        {
            Redact redact = redaction.MarkRedactAnnot(page, rect_array);
            redact.ResetAppearanceStream();
            doc.SaveAs(output_path + "AboutFoxit_redeacted_default.pdf",
(int)foxit.pdf.PDFDoc.SaveFlags.e_SaveFlagNormal);

            // 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_path + "AboutFoxit_redeacted_setColor.pdf",
(int)foxit.pdf.PDFDoc.SaveFlags.e_SaveFlagNormal);

            redact.SetOpacity((float)0.5);
            redact.ResetAppearanceStream();
            doc.SaveAs(output_path + "AboutFoxit_redeacted_setOpacity.pdf",
(int)foxit.pdf.PDFDoc.SaveFlags.e_SaveFlagNormal);

            if (redaction.Apply())
                Console.WriteLine("Redact page(0) succeed.");
            else
                Console.WriteLine("Redact page(0) failed.");
            redact.Dispose();
        }
    }
}

```

3.32 对比 (Comparison)

对比功能可以帮助用户查看两个版本的 PDF 文档之间的差异。Foxit PDF SDK 提供 APIs 用以逐页比较两个 PDF 文档，并返回文档间的差异。

差异可以定义为三种类型：删除、插入和替换。您可以将这些差异保存为 PDF 文件并标记为注释。

备注：要使用对比功能，请确保授权 key 文件中包含 'Comparison' 模块。

Example:

3.32.1 如何对比两个 PDF 文档，并将差异保存到一个 PDF 文件中

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;
using foxit.pdf.annots;
using foxit.addon;

...

using (PDFDoc base_doc = new PDFDoc("input_base_file"))
{
    error_code = base_doc.Load(null);
    if (error_code != ErrorCode.e_ErrSuccess)
    {
        Library.Release();
        return;
    }

    using (PDFDoc compared_doc = new PDFDoc("input_compared_file"))
    {
        error_code = compared_doc.Load(null);
        if (error_code != ErrorCode.e_ErrSuccess)
        {
            Library.Release();
            return;
        }

        using (Comparison comparison = new Comparison(base_doc, compared_doc))
        {
            // Start comparing.
            CompareResults result = comparison.DoCompare(0, 0,
(int)Comparison.CompareType.e_CompareTypeText);
            CompareResultInfoArray oldInfo = result.results_base_doc;
            CompareResultInfoArray newInfo = result.results_compared_doc;
            uint oldInfoSize = oldInfo.GetSize();
            uint newInfoSize = newInfo.GetSize();

            using (PDFPage page = compared_doc.GetPage(0))
            {

```

```

for (uint i = 0; i < newInfoSize; i++)
{
    CompareResultInfo item = newInfo.GetAt(i);
    CompareResultInfo.CompareResultType type = item.type;
    if (type == CompareResultInfo.CompareResultType.e_CompareResultTypeDeleteText)
    {
        String res_string = String.Format("\'{0}\'", item.diff_contents);

        // Add stamp to mark the "delete" type differences between the two documents.
        CreateDeleteTextStamp(page, item.rect_array, 0xff0000, res_string, "Compare : Delete", "Text");
    }
    else if (type == CompareResultInfo.CompareResultType.e_CompareResultTypeInsertText)
    {
        String res_string = String.Format("\'{0}\'", item.diff_contents);

        // Highlight the "insert" type differences between the two documents.
        CreateHighlightRect(page, item.rect_array, 0x0000ff, res_string, "Compare : Insert", "Text");
    }
    else if (type == CompareResultInfo.CompareResultType.e_CompareResultTypeReplaceText)
    {
        String res_string = String.Format("[Old]: \'{0}\'\r\n[New]: \'{1}\'", oldInfo.GetAt(i).diff_contents,
item.diff_contents);

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

// Save the comparison result to a PDF file.
compared_doc.SaveAs(output_path + "result.pdf", (int)PDFDoc.SaveFlags.e_SaveFlagNormal);
}
}
}

```

备注：对于 *CreateDeleteTextStamp* 和 *CreateHighlightRect* 函数，请参考 SDK 包中 "*\examples\simple_demo*" 文件夹下的 "**pdfcompare**" demo。

3.33 Compliance

PDF Compliance

Foxit PDF SDK 支持 PDF 版本标准化转换，当前支持的版本有 PDF 1.3, PDF 1.4, PDF 1.5, PDF 1.6 和 PDF 1.7。当转换到 PDF 1.3 版本时，如果源文档含有透明度的数据，则其会被转换到 PDF 1.4 版本而不是 PDF 1.3 版本 (PDF 1.3 版本不支持透明度)；如果源文档不含有任何透明度的数据，则会按预期转换到 PDF 1.3 版本。

PDF/A Compliance

PDF/A 是一种 ISO 标准的 PDF 文件格式版本，用于电子文档的存档和长期保存。PDF/A 与 PDF 的不同之处在于 PDF/A 禁用了 PDF 中不适合长期存档的特性，比如字体链接（与嵌入字体相对）、加密、JavaScript、音频和视频等。

Foxit PDF SDK 提供 APIs 用以验证 PDF 是否符合 PDF/A 标准，或将 PDF 转换为符合 PDF/A 标准的文档。支持的 PDF/A 标准包括 PDF/A-1a、PDF/A-1b、PDF/A-2a、PDF/A-2b、PDF/A-2u、PDF/A-3a、PDF/A-3b、PDF/A-3u (ISO 19005- 1, 19005 -2 和 19005-3)。

本节将介绍如何设置相关环境以运行 'compliance' demo。

3.33.1 系统需求

平台: Windows, Linux, Mac

开发语言: C, C++, Java, C#, Objective-C

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

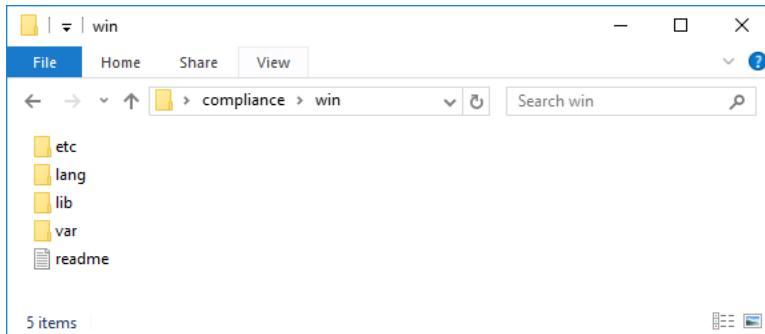
SDK 版本: Foxit PDF SDK 7.0 或更高版本 (对于 PDF Compliance，则需要 Foxit PDF SDK 7.1 或更高版本)

3.33.2 Compliance 资源文件

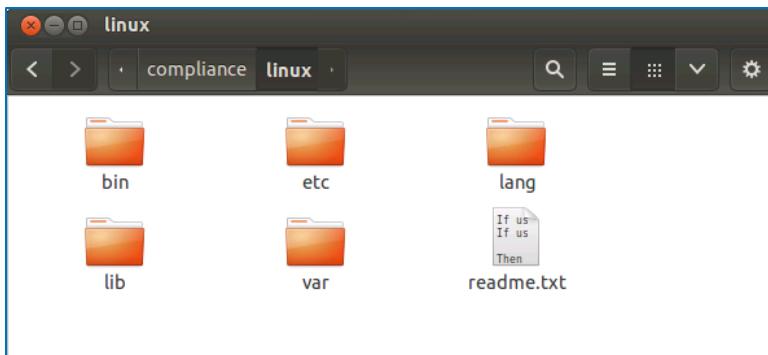
请联系 Foxit 支持团队或者销售团队以获取 Compliance 资源文件包。

获取到资源文件包后，将其解压到所需目录（比如，Windows 解压到 "**compliance/win**"，Linux 解压到 "**compliance/linux**"，Mac 解压到 "**compliance/mac**"），然后您将看到 Compliance 的资源文件如下：

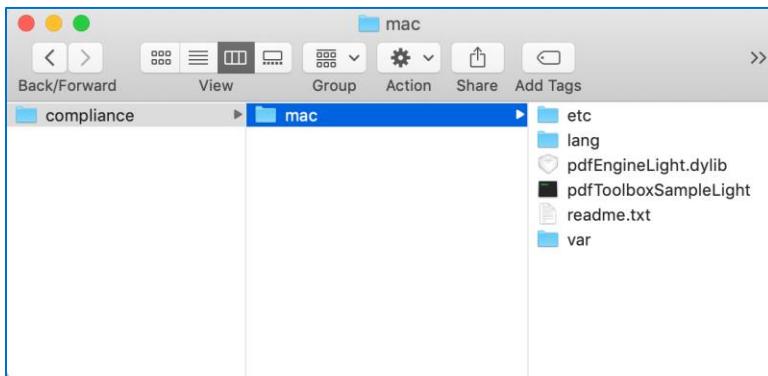
Windows 平台:



Linux 平台:



Mac 平台:



3.33.3 如何运行 compliance demo

Foxit PDF SDK 提供了一个 **compliance** demo 用来展示如何使用 Foxit PDF SDK 验证 PDF 文档是否符合 PDF/A 标准，如何将 PDF 转换为符合 PDF/A 标准的文档，以及如何进行 PDF 版本标准化转换。该 demo 位于 "\examples\simple_demo\compliance" 文件夹下。

3.33.3.1 构建一个 compliance 资源目录

在运行 **compliance** demo 之前，您需要首先构建一个 compliance 资源目录，然后将该目录的路径传给 **ComplianceEngine.Initialize** 接口用来初始化 compliance 引擎。

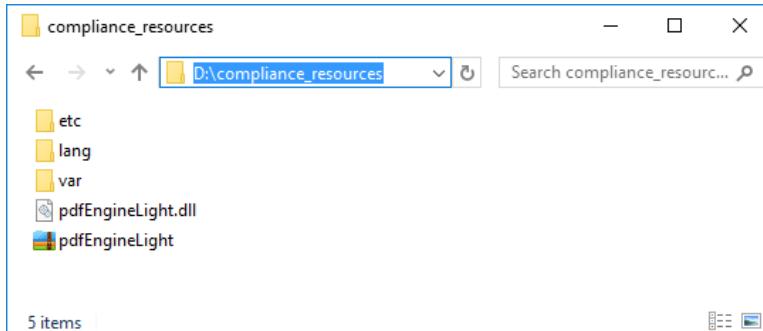
Windows

在 Windows 平台构建一个 compliance 资源目录，请按照如下的步骤：

- 1) 新建一个文件夹作为 compliance 的资源目录。比如，"D:/compliance_resources"。
- 2) 将 "compliance/win" 目录下的 "**etc**"、"**lang**"、"**var**" 文件夹拷贝到 "D:/compliance_resources"。
- 3) 根据要编译的平台架构，选择相应的库资源。
 - 如果使用 **win32**，则将 "compliance/win/lib/x86" 文件夹下的所有文件拷贝到 "D:/compliance_resources"。

- 如果使用 **win64**, 则将 "compliance/win/lib/x64" 文件夹下的所有文件拷贝到 "D:/compliance_resources"。

例如, 使用 **win64** 平台架构, 则 compliance 资源目录如下所示:

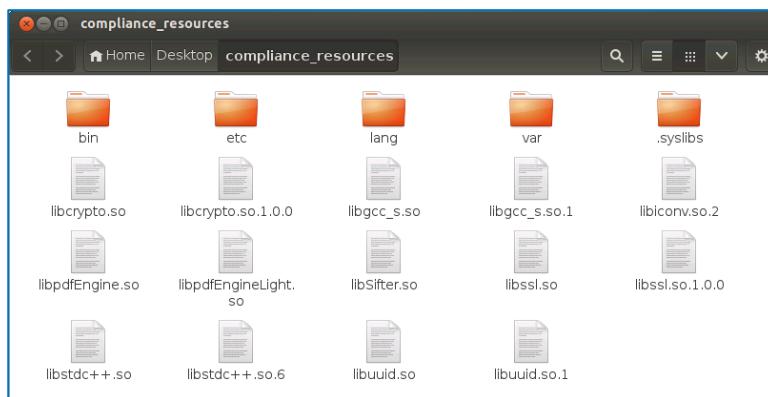


Linux (64-bit)

在 Linux 平台构建一个 compliance 资源目录, 请按照如下的步骤:

- 1) 新建一个文件夹作为 compliance 的资源目录。比如, "/root/Desktop/compliance_resources"。
- 2) 将 "compliance/linux" 目录下的 "bin"、"etc"、"lang"、"var" 文件夹拷贝到 "D:/compliance_resources"。
- 3) 将 "compliance/linux/lib/x64" 文件夹下的所有文件拷贝到 "/root/Desktop/compliance_resources"。

则 compliance 资源目录将如下所示:



备注: 对于 Linux 平台, 在运行 demo 之前, 您需要将 compliance 资源目录加入到系统共享库目录的查找路径中, 否则 **ComplianceEngine.Initialize** 会执行失败。

例如, 在运行 demo 之前, 您可以通过命令 (`export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}: /root/Desktop/compliance_resources`) 将 compliance 资源目录临时加入到 LD_LIBRARY_PATH。

Mac (64-bit)

对于 Mac 平台，您可以直接使用 "compliance/mac" 资源文件夹作为 compliance 的资源目录。

3.33.3.2 配置 demo

构建 compliance 资源目录后，在 "\examples\simple_demo\compliance\compliance.cs" 文件中配置 demo。

本节以 Windows 平台为例，展示如何在 "compliance.cs" 文件中进行 demo 配置。对于 Linux 和 Mac 平台，其配置操作与 Windows 平台相同。

指定 compliance 资源目录

如下所示，添加 compliance 资源目录，用以初始化 compliance 引擎。

```

177     try
178     {
179         string input_file = input_path + "AboutFoxit.pdf";
180         // "compliance_resource_folder_path" is the path of compliance resource folder. Please refer to Developer Guide for more details.
181         string compliance_resource_folder_path = "D:/compliance_resources";
182         // If you use an authorization key for Foxit PDF SDK, please set a valid unlock code string to compliance_engine_unlockcode for ComplianceEngine.
183         // If you use a trial key for Foxit PDF SDK, just keep compliance_engine_unlockcode as an empty string.
184         string compliance_engine_unlockcode = "";
185
186         if (compliance_resource_folder_path.Length < 1)
187             Console.WriteLine("compliance_resource_folder_path is still empty. Please set it with a valid path to compliance resource folder path.");
188         return ;
189     }
190     // Initialize compliance engine.
191     error_code = ComplianceEngine.Initialize(compliance_resource_folder_path, compliance_engine_unlockcode);

```

备注：如果您已经购买了授权的 license key (包含 'Compliance' 模块的权限)，Foxit 销售团队会给您额外发送一个 unlock code，用以初始化 compliance 引擎。

(可选) 为 compliance engine 设置语言

ComplianceEngine.SetLanguage 函数用来为 compliance 引擎设置语言。默认的语言是英语，所有支持的语言如下所示：

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

例如，将语言设置为 "Chinese-Simplified"。

```
// Set language. The default language is "English".
ComplianceEngine.SetLanguage("Chinese-Simplified");
```

(可选) 为 compliance 引擎设置临时文件夹

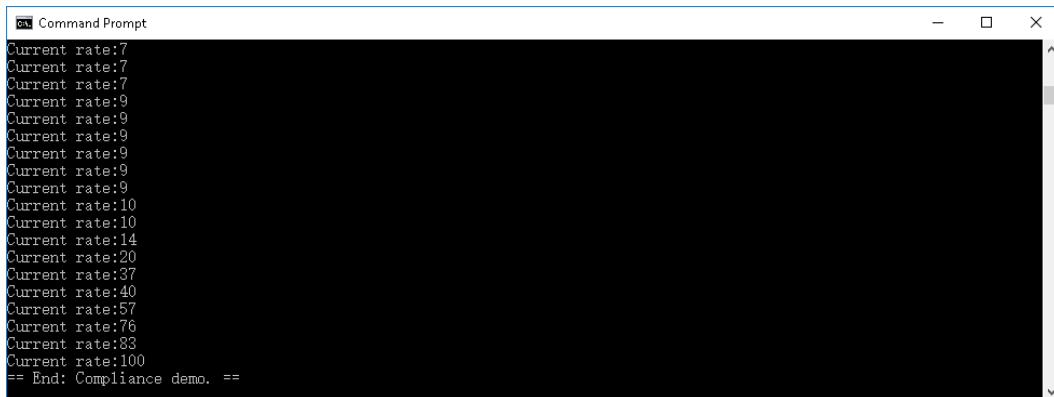
ComplianceEngine.SetTempFolderPath 函数用于设置一个临时文件夹以存储处理过程 (如验证或转换) 中生成的文件。如果此函数未设置自定义的临时文件夹，则将使用系统中默认的临时文件夹。

例如，将路径设置为 "D:/compliance_temp" (必须为一个有效的路径)。

```
// Set custom temp folder path for ComplianceEngine.
ComplianceEngine.SetTempFolderPath(L"D:/compliance_temp");
```

3.33.3.3 运行 demo

打开命令行窗口，导航到"\examples\simple_demo"，对于 64 位 Windows 平台，运行 "RunDemo.bat compliance"。成功运行 demo 后，控制台将默认打印以下内容：



```
Current rate:7
Current rate:7
Current rate:7
Current rate:9
Current rate:10
Current rate:10
Current rate:14
Current rate:20
Current rate:37
Current rate:40
Current rate:57
Current rate:76
Current rate:83
Current rate:100
== End: Compliance demo. ==
```

该 demo

- 验证 PDF ("\\examples\\simple_demo\\input_files\\AboutFoxit.pdf") 是否符合 PDF/A-1a 标准，并将此文档转换为符合 PDF/A-1a 标准的文档。
- 将 PDF ("\\examples\\simple_demo\\input_files\\AF_ImageXObject_FormXObject.pdf") 分别转换为 PDF-1.4 和 PDF-1.7 版本。

输出文档位于"\examples\simple_demo\output_files\compliance" 文件夹下。

3.34 优化 (Optimization)

优化功能可以通过压缩 PDF 文件中的图片、删除冗余数据，以及丢弃无用的用户数据等方式有效地减少 PDF 文件的大小，从而节省磁盘空间以及便于 PDF 文件的传输和存储。从 7.1 版本开始，优化模块提供了压缩 PDF 文件中彩色、灰度和黑白图像的方法，用于减少 PDF 文件的大小。

备注：要使用优化功能，请确保授权 key 文件中包含 'Optimization' 模块。

Example:

3.34.1 如何通过压缩 PDF 文件中的彩色、灰度和黑白图像来减少 PDF 文件的大小

```
using System;

using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;
using foxit.addon;
using foxit.addon.optimization;
```

```

using (PDFDoc doc = new PDFDoc("input_pdf_file"))
{
    error_code = doc.Load(null);
    if (error_code != ErrorCode.e_ErrSuccess)
    {
        Console.WriteLine("Document Load Error: {0}\n", error_code);
        return;
    }
    using (Optimization_Pause pause = new Optimization_Pause(0, true))
    using (OptimizerSettings settings = new OptimizerSettings())
    {
        Console.WriteLine("Optimized Start.\n");
        Progressive progressive = Optimizer.Optimize(doc, settings, pause);
        Progressive.State progress_state = Progressive.State.e_ToBeContinued;
        while (Progressive.State.e_ToBeContinued == progress_state)
        {
            progress_state = progressive.Continue();
            int percent = progressive.GetRateOfProgress();
            Console.WriteLine("Optimize progress percent: {0}%\n", percent);
        }
        if (Progressive.State.e_Finished == progress_state)
        {
            doc.SaveAs("ImageCompression_Optimized.pdf",
(int)PDFDoc.SaveFlags.e_SaveFlagRemoveRedundantObjects);
        }
        Console.WriteLine("Optimized Finish.\n");
    }
}

```

3.35 HTML 转 PDF

对于一些内容比较多的 HTML 大文件或者网页，直接进行打印或者存档不太容易。Foxit PDF SDK 提供 API 接口将在线网页或者本地 HTML 文件转换为 PDF 文件，比如发票，报告等，使其更容易打印或者存档。在 HTML 转 PDF 的过程中，Foxit PDF SDK 支持在基于 HTML 的组织结构的基础上创建和添加 PDF Tag。

从 8.1 版本开始，Foxit PDF SDK 支持以文件流的形式提供 HTML2PDF 转换后生成的文件，如果您需要使用该功能，请联系 Foxit 支持团队或者销售团队以获取最新的 HTML 转 PDF 引擎文件包。

从 7.6 版本开始，Foxit PDF SDK 支持在 Linux 平台将 HTML 转化为 PDF。但是对于 Linux 平台的 HTML 转 PDF 引擎，libnss 必须是 3.22 版本的。

本节主要介绍如何配置运行 'html2pdf' demo 所需的环境。

3.35.1 系统需求

平台: Windows, Linux, Mac

开发语言: C, C++, Java, C#, Objective-C

License Key: license key 中包含 'Conversion' 模块的权限

SDK 版本: Foxit PDF SDK 7.0 或更高版本

3.35.2 HTML 转 PDF 引擎资源

请联系 Foxit 支持团队或者销售团队以获取 HTML 转 PDF 引擎文件包。

当获取到引擎文件包后，将其解压到所需目录(比如，Windows 解压到 "**D:/htmktopdf/win**"，Linux 解压到 "**htmktopdf/linux**"，Mac 解压到 "**htmktopdf/mac**")。

3.35.3 如何运行 html2pdf demo

Foxit PDF SDK 提供了一个 **html2pdf** demo 用来展示如何使用 Foxit PDF SDK 将 html 文件转换为 PDF 文件。该 demo 位于 "\examples\simple_demo\html2pdf" 文件夹下。

3.35.3.1 配置 demo

对于 html2pdf demo，您可以在"\examples\simple_demo\html2pdf\html2pdf.cs"文件中配置 demo，或者您可以在命令行或者终端窗口中直接使用参数来对 demo 进行配置。以下将以 Windows 平台为例，在 "html2pdf.cs" 文件中配置 demo。对于 Linux 和 Mac 平台，其配置操作与 Windows 相同。

指定 html2pdf 引擎资源目录

在 "html2pdf.cs" 文件中，如下所示，添加 "fxhtml2pdf.exe" 引擎文件的路径，用以将 html 文件转换为 PDF 文件。

```
// "engine_path" is the path of the engine file "fxhtml2pdf" which is used to converting html to pdf. Please refer to Developer Guide for more details.  
static String engine_path = "D:/htmktopdf/win/fxhtml2pdf.exe"; // or engine_path = "D:/htmktopdf/win/fxhtml2pdf";
```

(可选) 指定 cookies 文件路径

添加 cookies 文件路径，该文件是从您需要转换的 web 页面中导出的。比如，

```
// "cookies_path" is the path of the cookies file exported from the web pages that you want to convert. Please refer to Developer Guide for more details.  
static String cookies_path = "D:/cookies.txt";
```

3.35.3.2 运行 demo

运行 demo (不带参数)

打开命令行窗口，导航到"\examples\simple_demo"，对于 64 位 Windows 平台，运行 "**RunDemo.bat html2pdf**"。成功运行 demo 后，控制台将打印 "Convert HTML to PDF successfully."。

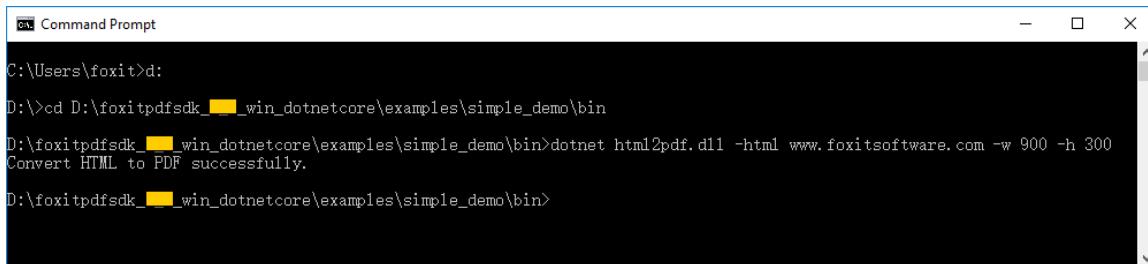
运行 demo (带参数)

打开命令行窗口，导航到 "\examples\simple_demo"，对于 64 位 Windows 平台，首先运行 "**RunDemo.bat html2pdf**"。

然后，在命令行窗口，导航到 "\examples\simple_demo\bin"，输入 "**dotnet html2pdf.dll --help**" 命令查看如何使用参数来运行 demo。

例如，将 "www.foxitsoftware.com" 网页转换为 PDF 文件，并且设置转换后的 PDF 页面的宽度为 900 points，高度为 300 points，请运行如下的命令：

```
dotnet html2pdf.dll -html www.foxitsoftware.com -w 900 -h 300
```



```
C:\ Command Prompt
C:\Users\foxit>d:
D:\>cd D:\foxitpdfsdk_2020_win_dotnetcore\examples\simple_demo\bin
D:\foxitpdfsdk_2020_win_dotnetcore\examples\simple_demo\bin>dotnet html2pdf.dll -html www.foxitsoftware.com -w 900 -h 300
Convert HTML to PDF successfully.
D:\foxitpdfsdk_2020_win_dotnetcore\examples\simple_demo\bin>
```

输出文档位于"\examples\simple_demo\output_files\html2pdf" 文件夹下。

参数描述

基本语法：

```
html2pdf <-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 rotate degree>] [-mode <page mode>]
[-scale <whether scale page>] [-link <whether convert link>] [-tag <whether generate tag>]
[-cookies <cookies file path>] [-timeout <timeout>]
```

html2pdf --help

备注：

- <> 必选
- [] 可选

参数	描述
--help	参数用法的帮助信息。
-html	URL 或者 html 文件的路径。比如'-html www.foxitsoftware.com'。
-o	输出 PDF 文件的路径。
-engine	"fxhtml2pdf.exe" 引擎文件的路径。
-w	输出 PDF 文件页面的宽度，单位是 points。
-h	输出 PDF 文件页面的高度，单位是 points。
-r	输出 PDF 文件页面的旋转度。

参数	描述
	<ul style="list-style-type: none">• 0 : 0 .• 1 : 90 度.• 2 : 180 度.• 3 : 270 度.
-ml	输出 PDF 文件页面的左边距。
-mr	输出 PDF 文件页面的右边距。
-mt	输出 PDF 文件页面的上边距。
-mb	输出 PDF 文件页面的下边距。
-mode	输出 PDF 文件的页面模式。 <ul style="list-style-type: none">• 0 : 单页模式。• 1 : 多页模式。
-scale	是否缩放页面。 <ul style="list-style-type: none">• 'yes' : 缩放页面。• 'no' : 不需要缩放页面。
-link	是否转换链接。 <ul style="list-style-type: none">• 'yes' : 转换链接。• 'no' : 不需要转换链接。
-tag	是否生成 tag. <ul style="list-style-type: none">• 'yes' : 生成 tag。• 'no' : 不需要生成 tag。
-cookies	cookies 文件路径，该文件是从您需要转换的 web 页面中导出的。
-timeout	加载 web 页面的超时时间。

3.36 Office 转 PDF

从 7.3 版本开始，Foxit PDF SDK 提供了 API 接口，用于在 Windows 平台上将 Microsoft Office 文档 (Word 和 Excel) 转换为专业质量的 PDF 文档。

从 7.4 版本开始，Foxit PDF SDK 支持在 Windows 平台上将 PowerPoint 文档转换为 PDF 文档。

对于使用此功能，请注意：

- 确保 Windows 系统已安装 Microsoft Office 2007 或更高版本。
- 在将 Excel 转换为 PDF 之前，请确保在 Windows 系统上已设置了默认打印机 (虚拟打印机也可以)。

3.36.1 系统需求

平台: Windows

开发语言: C, C++, Java, C#

License Key: license key 中包含 'Conversion' 模块的权限

SDK 版本: Word 和 Excel (Foxit PDF SDK 7.3 或更高版本) , PowerPoint (Foxit PDF SDK 7.4 或更高版本)

Example:

3.36.2 如何将 Word 文档转换为 PDF 文档

```
using foxit;
...
// Make sure that SDK has already been initialized successfully.

string word_file_path = "test.doc";
string saved_pdf_path = "saved.pdf";

// Use default Word2PDFSettingData values.
using (foxit.addon.conversion.Word2PDFSettingData word_convert_setting_data = new
foxit.addon.conversion.Word2PDFSettingData())
{
    foxit.addon.conversion.Convert.FromWord(word_file_path, "", saved_pdf_path,
word_convert_setting_data);
}
```

3.36.3 如何将 Excel 文件转换为 PDF 文档

```
using foxit;
...
// Make sure that SDK has already been initialized successfully.

string excel_file_path= "test.xls";
string saved_pdf_path = "saved.pdf";
// Use default Excel2PDFSettingData values.
using (foxit.addon.conversion.Excel2PDFSettingData excel_convert_setting_data = new
foxit.addon.conversion.Excel2PDFSettingData())
{
    foxit.addon.conversion.Convert.FromExcel(excel_file_path, "", saved_pdf_path,
excel_convert_setting_data);
}
```

3.36.4 如何将 PowerPoint 文件转换为 PDF 文档

```
using foxit;
...
// Make sure that SDK has already been initialized successfully.

string ppt_file_path = "test.ppt";
string saved_pdf_path = "saved.pdf";

// Use default PowerPoint2PDFSettingData values.
using (foxit.addon.conversion.PowerPoint2PDFSettingData ppt_convert_setting_data = new
foxit.addon.conversion.PowerPoint2PDFSettingData())
{
```

```
    foxit.addon.conversion.Convert.FromPowerPoint(ppt_file_path, "", saved_pdf_path,
    ppt_convert_setting_data);
}
```

3.37 输出预览 (Output Preview)

从版本 7.4 开始，Foxit PDF SDK 支持输出阅览功能，可以预览分色和测试不同的颜色配置文件。

3.37.1 系统需求

平台: Windows, Linux, Mac

开发语言: C, C++, Java, C#, Objective-C

License Key: 有效的 license key

SDK 版本: Foxit PDF SDK 7.4 或更高版本

3.37.2 如何运行 output preview demo

在运行 "\examples\simple_demo\output_preview" 文件夹下的 output preview demo 之前，您需要首先将变量 **default_icc_folder_path** 设置为 SDK 包下 "\res\icc_profile" 文件夹的路径。例如：

```
// "default_icc_folder_path" is the path of the folder which contains default icc profile files. Please refer to
Developer Guide for more details.
string default_icc_folder_path = "E:\foxitpdfsdk_X_X_win_dotnetcore/res/icc_profile";
```

然后，按照其他 demo 运行的步骤运行该 demo。

3.37.3 如何使用 Foxit PDF SDK 进行输出预览

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

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

// Set folder path which contains default icc profile files.
Library.SetDefaultICCProfilesPath(default_icc_folder_path);

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

using (OutputPreview output_preview = new OutputPreview(pdf_doc))
{
    string simulation_icc_file_path = "icc_profile.icc";
    output_preview.SetSimulationProfile(simulation_icc_file_path);
    output_preview.SetShowType(OutputPreview.ShowType.e_ShowAll);
    StringArray process_plates = output_preview.GetPlates(OutputPreview.ColorantType.e_ColorantTypeProcess);
    StringArray spot_plates = output_preview.GetPlates(OutputPreview.ColorantType.e_ColorantTypeSpot);

    // Set check status of process plate to be true, if there's any process plate.
    for (uint i = 0; i < process_plates.GetSize(); i++)
```

```
{  
    output_preview.SetCheckStatus(process_plates.GetAt(i), true);  
}  
  
// Set check status of spot plate to be true, if there's any spot plate.  
for (uint i = 0; i<spot_plates.GetSize(); i++)  
{  
    output_preview.SetCheckStatus(spot_plates.GetAt(i), true);  
}  
  
using (System.Drawing.Bitmap preview_bitmap = output_preview.GeneratePreviewBitmap(pdf_page,  
display_matrix, renderer))  
{  
    ...// Use preview bitmap or save it.  
}  
}
```

3.38 合并 (Combination)

合并功能用来将多个 PDF 文件合并成一个 PDF 文件。

3.38.1 如何将多个 PDF 文件合并成一个 PDF 文件

```
using foxit.pdf;  
  
// Make sure that SDK has already been initialized successfully.  
  
CombineDocumentInfoArray info_array = new CombineDocumentInfoArray();  
info_array.Add(new CombineDocumentInfo(input_path + "AboutFoxit.pdf", ""));  
info_array.Add(new CombineDocumentInfo(input_path + "Annot_all.pdf", ""));  
info_array.Add(new CombineDocumentInfo(input_path + "SamplePDF.pdf", ""));  
  
String savepath = output_path + "Test_Combined.pdf";  
int option = (int)(Combination.CombineDocsOptions.e_CombineDocsOptionBookmark |  
Combination.CombineDocsOptions.e_CombineDocsOptionAcroformRename |  
Combination.CombineDocsOptions.e_CombineDocsOptionStructreeTree |  
Combination.CombineDocsOptions.e_CombineDocsOptionOutputIntents |  
Combination.CombineDocsOptions.e_CombineDocsOptionOCProperties |  
Combination.CombineDocsOptions.e_CombineDocsOptionMarkInfos |  
Combination.CombineDocsOptions.e_CombineDocsOptionPageLabels |  
Combination.CombineDocsOptions.e_CombineDocsOptionNames |  
Combination.CombineDocsOptions.e_CombineDocsOptionObjectStream |  
Combination.CombineDocsOptions.e_CombineDocsOptionDuplicateStream);  
  
Progressive progress = Combination.StartCombineDocuments(savepath, info_array, option, null);  
Progressive.State progress_state = Progressive.State.e_ToBeContinued;  
while (Progressive.State.e_ToBeContinued == progress_state)  
{  
    progress_state = progress.Continue();  
}
```

3.39 PDF Portfolio

PDF portfolios 是具有不同格式的文件的组合。Portfolio 文件本身是一个 PDF 文档，不同格式的文件可以嵌入到这种 PDF 文档中。

3.39.1 系统需求

平台: Windows, Linux, Mac

开发语言: C, C++, Java, C#, Objective-C

License Key: 有效的 license key

SDK 版本: Foxit PDF SDK 7.6 或更高版本

Example:

3.39.2 如何创建一个新的空白的 PDF Portfolio 文档

```
using foxit;
using foxit.common;
using foxit.common.fxcrf;
using foxit.pdf;

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

using (Portfolio new_portfolio = Portfolio.CreatePortfolio())
{
    // Set properties, add file/folder node to the new portfolio.
    ...

    // Get portfolio PDF document object.
    using (PDFDoc portfolio_pdf_doc = new_portfolio.GetPortfolioPDFDoc())
    {
        ...
    }
}
```

3.39.3 如何从一个 PDF portfolio 文档创建一个 Portfolio 对象

```
using foxit;
using foxit.common;
using foxit.common.fxcrf;
using foxit.pdf;

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

using (PDFDoc portfolio_pdf_doc = new PDFDoc("portfolio.pdf"))
{
    ErrorCode error_code = portfolio_pdf_doc.Load(null);
    if (ErrorCode.e_Success == error_code)
    {
```

```
if (portfolio_pdf_doc.IsPortfolio())
{
    using (Portfolio existed_portfolio = Portfolio.CreatePortfolio(portfolio_pdf_doc))
    {
        ...
    }
}
}
```

3.39.4 如何获取 portfolio nodes

```
using foxit;
using foxit.common;
using foxit.common.fxcr;
using foxit.pdf;

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

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

using (PortfolioNode root_node = portfolio.GetRootNode())
{
    using (PortfolioFolderNode root_folder = new PortfolioFolderNode(root_node))
    {
        using (PortfolioNodeArray sub_nodes = root_folder.GetSortedSubNodes())
        {
            for (uintindex = 0; index < sub_nodes.GetSize(); index++)
            {
                using (PortfolioNode node = sub_nodes.GetAt(index))
                {
                    switch (node.GetNodeType())
                    {
                        case PortfolioNode::e_TypeFolder:
                            {
                                using (PortfolioFolderNode folder_node = new PortfolioFolderNode(node))
                                {
                                    // Use PortfolioFolderNode's getting method to get some properties.
                                    ...

                                    PortfolioNodeArray sub_nodes_2 = folder_node.GetSortedSubNodes();
                                    ...
                                    break;
                                }
                            }
                        case PortfolioNode::e_TypeFile:
                            {
                                using (PortfolioFileNode file_node = new PortfolioFileNode(node))
                                {
                                    // Get file specification object from this file node, and then get/set information from/to this
file specification object.
                                }
                            }
                    }
                }
            }
        }
    }
}
```

```
        using (FileSpec file_spec = file_node.GetFileSpec())
    {
        ...
        break;
    }
}
}
}
}
}
```

3.39.5 如何添加 file node 或者 folder node

```
using foxit;
using foxit.common;
using foxit.common.fxcrt;
using foxit.pdf;

// 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.
string path_to_a_file = "directory/Sample.txt";
using (PortfolioFileNode 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.
MyStreamCallback my_stream_callback = new MyStreamCallback();
using (PortfolioFileNode 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".
...

using (PortfolioFileNode 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.
```

```
using (PortfolioFolderNode new_sub_foldernode = root_folder.AddSubFolder("Sub Folder-1") {
```

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

```
    ...
```

```
}
```

3.39.6 如何移除 node

```
using foxit;
```

```
using foxit.common;
```

```
using foxit.common.fxcrt;
```

```
using foxit.pdf;
```

```
// 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);
```

FAQ

1. 如何获取 PDF 文件中指定位置的文本对象，以及更改文本对象的内容？

使用 Foxit PDF SDK 获取 PDF 文件中指定位置的文本对象以及修改文本对象的内容，请按照如下的步骤：

- 1) 打开一个 PDF 文件。
- 2) 加载 PDF 页面并获取该页面中的页面对象。
- 3) 使用 **PDFPage.GetGraphicsObjectAtPoint** 获取指定位置的文本对象。注意：使用页面对象获取矩形来查看文本对象的位置。
- 4) 更改文本对象的内容并保存 PDF 文档。

以下是示例代码：

```
using foxit.common;
using foxit.pdf;
using foxit.pdf.graphics;
using foxit.common.fxcr;

...
namespace graphics_objectsCS
{
    class graphics_objects
    {
        ...
        static bool ChangeTextObjectContent()
        {

            string input_file = input_path + "AboutFoxit.pdf";
            using (var doc = new PDFDoc(input_file))
            {
                ErrorCode error_code = doc.Load(null);
                if (error_code != ErrorCode.e_ErrSuccess)
                {
                    Console.WriteLine("The Doc {0} Error: {1}", input_file, error_code);
                    return false;
                }
                try
                {
                    // Get original shading objects from the first PDF page.
                    using (var original_page = doc.GetPage(0))
                    {
                        using (var progressive =
original_page.StartParse((int)PDFPage.ParseFlags.e_ParsePageNormal, null, false))
                        {
                            PointF pointf = new PointF(92, 762);
                            using (var arr = original_page.GetGraphicsObjectsAtPoint(pointf, 10,
GraphicsObject.Type.e_TypeText))
```

```
{  
    for (int i = 0; i < arr.GetSize(); i++)  
    {  
        using (var graphobj = arr.GetAt(i))  
        {  
            using (var textobj = graphobj->GetTextObject())  
            {  
                textobj.SetText("Foxit Test");  
            }  
        }  
    }  
    original_page.GenerateContent();  
}  
}  
string output_directory = output_path + "graphics_objects/";  
string output_file = output_directory + "After_revise.pdf";  
doc.SaveAs(output_file, (int)PDFDoc.SaveFlags.e_SaveFlagNormal);  
}  
catch (System.Exception e)  
{  
    Console.WriteLine(e.Message);  
}  
return true;  
}  
}  
...  
}
```

2. 是否可以改变嵌入 TIFF 图像的 DPI?

无法改变。PDF 中图像的 DPI 是静态的，如果图像已经存在，Foxit PDF SDK 没有更改图像 DPI 的功能。

解决办法是您可以使用第三方库来更改图像的 DPI，然后将其添加到 PDF 中。

备注: Foxit PDF SDK 提供了一个函数 "Image.SetDPIs"，可以用来设置图片对象的 DPI 属性，但是它仅支持使用 Foxit PDF SDK 创建或者使用 "Image.AddFrame" 函数创建的图像，不支持 JPX, GIF 和 TIF 格式。

附录

Foxit PDF SDK 支持的 JavaScript 列表

对象的属性或者方法

对象	属性/方法名称	支持的最低 SDK 版本
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
annotation method	destroy	V7.0
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
app methods	viewerVersion	V4.0
	alert	V4.0
	beep	V4.0
	browseForDoc	V4.0
	clearInterval	V4.0
	clearTimeOut	V4.0
	launchURL	V4.0

对象	属性/方法名称	支持的最低 SDK 版本
	mailMsg	V4.0
	response	V4.0
	setInterval	V4.0
	setTimeOut	V4.0
	popUpMenu	V4.0
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
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

对象	属性/方法名称	支持的最低 SDK 版本
	subject	V4.0
	title	V4.0
document methods	addAnnot	V7.0
	addField	V4.0
	addIcon	V4.0
	calculateNow	V4.0
	createDataObject	V6.2
	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
event properties	removeField	V4.0
	replacePages	V6.2
	resetForm	V4.0
	submitForm	V4.0
	mailDoc	V4.0
	change	V4.0
	changeEx	V4.0
	commitKey	V4.0

对象	属性/方法名称	支持的最低 SDK 版本
event methods	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	-	-
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
	name	V4.0
	numItems	V4.0

对象	属性/方法名称	支持的最低 SDK 版本
	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
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
global methods	setPersistent	V4.0
Icon properties	name	V4.0
util methods	printd	V4.0
	printf	V4.0
	printx	V4.0

对象	属性/方法名称	支持的最低 SDK 版本
	scand	V4.0
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

全局方法

方法名称	支持的最低 SDK 版本
AFNumber_Format	V4.0
AFNumber_Keystroke	V4.0
AFPercent_Format	V4.0
AFPercent_Keystroke	V4.0
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

引用

[1] PDF reference 1.7

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[2] PDF reference 2.0

<https://www.iso.org/standard/63534.html>

[3] Foxit PDF SDK API reference

sdk_folder/doc/Foxit PDF SDK DotnetCore API Reference.html

备注: sdk_folder 是 SDK 包解压后的目录。

技术支持

您可以直接联系 **Foxit**, 请使用以下的联系方式:

线上支持:

- <https://www.foxitsoftware.cn/support>

联系销售:

- 电话: 1-866-680-3668
- 邮箱: sales@foxitsoftware.com

联系技术支持团队:

- 电话: 1-866-MYFOXIT or 1-866-693-6948
- 邮箱: support@foxitsoftware.com