



开发者指南

Foxit PDF SDK

For iOS

Microsoft® Partner
Gold Independent Software Vendor (ISV)

TABLE OF CONTENTS

1	Foxit PDF SDK 简介	1
1.1	Foxit PDF SDK.....	1
1.2	Foxit PDF SDK for iOS.....	1
1.2.1	为什么选择 Foxit PDF SDK for iOS.....	1
1.2.2	Foxit PDF SDK for iOS 的主要框架	2
1.2.3	UI Extensions 组件概述.....	3
1.2.4	Foxit PDF SDK for iOS 的主要功能	5
1.3	评估	7
1.4	授权	7
1.5	关于此文档.....	7
2	入门指南	9
2.1	系统要求	9
2.2	包结构说明.....	9
2.2.1	不支持 Mac Catalyst 的发布包.....	9
2.2.2	支持 Mac Catalyst 的发布包	11
2.3	运行 demo.....	12
2.3.1	Function demo	12
2.3.2	Viewer control demo	14
2.3.3	Complete PDF viewer demo	16
3	快速构建一个功能齐全的 PDF 阅读器	21
3.1	使用 Foxit PDF SDK for iOS 构建一个 Objective-C 的 iOS 应用程序.....	21
3.1.1	创建一个 Objective-C 语言的 iOS 工程	21
3.1.2	集成 Foxit PDF SDK for iOS 到您的应用程序.....	24
3.1.3	初始化 Foxit PDF SDK for iOS	27
3.1.4	使用 FSPDFViewCtrl 显示 PDF 文档	27

3.1.5	使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器	30
3.1.6	基于功能齐全的 PDF 阅读器添加扫描功能	33
3.2	Make an iOS app in Swift with Foxit PDF SDK for iOS	35
3.2.1	创建一个 Swift 语言的 iOS 工程	36
3.2.2	集成 Foxit PDF SDK for iOS 到您的应用程序	36
3.2.3	初始化 Foxit PDF SDK for iOS	36
3.2.4	使用 FSPDFViewCtrl 显示 PDF 文档	37
3.2.5	使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器	39
3.2.6	基于功能齐全的 PDF 阅读器添加扫描功能	41
4	使用 Mac Catalyst 快速构建一个功能齐全的 PDF 阅读器	45
4.1.1	使用 Mac Catalyst 创建一个 Mac 应用程序	45
4.1.2	集成 Foxit PDF SDK for iOS (Catalyst) 到您的应用程序	45
4.1.3	初始化 Foxit PDF SDK for iOS (Catalyst)	48
4.1.4	使用 FSPDFViewCtrl 显示 PDF 文档	48
4.1.5	使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器	49
5	自定义 UI	52
5.1	通过配置文件自定义 UI	52
5.1.1	JSON 文件介绍	52
5.1.2	配置项描述	57
5.1.3	使用配置文件实例化一个 UIExtensionsManager 对象	63
5.1.4	通过配置文件自定义 UI 的示例	64
5.2	通过 APIs 自定义 UI 元素	66
5.2.1	自定义 top/bottom toolbar	67
5.2.2	自定义隐藏一个特定的面板	72
5.2.3	自定义隐藏 View setting bar 上的 UI 元素	75
5.2.4	自定义隐藏 More Menu 菜单上的 UI 元素	77
5.3	通过源代码自定义 UI 实现	83

6 使用 SDK API	89
6.1 Render	89
6.1.1 如何将指定的 PDF 页面渲染到 bitmap	89
6.1.2 如何将指定的 PDF 页面渲染到平台设备上下文	90
6.2 Text Page	91
6.2.1 如何通过选择获取页面上的文本区域	91
6.3 Text Search	92
6.3.1 如何在 PDF 文档中搜索指定的文本模型	93
6.4 Bookmark (Outline)	93
6.4.1 如何使用深度优先顺序遍历 PDF 文档的 bookmarks	94
6.5 Reading Bookmark	95
6.5.1 如何添加自定义 reading bookmark 并枚举所有的 reading bookmarks	95
6.6 Attachment	96
6.6.1 如何将指定文件嵌入到 PDF 文档	96
6.6.2 如何从 PDF 文档中导出嵌入的附件文件，并将其另存为单个文件	97
6.7 Annotation	97
6.7.1 如何向 PDF 页面中添加注释	98
6.7.2 如何删除 PDF 页面中的注释	99
6.8 Form	100
6.8.1 如何通过 XML 文件导入表单数据或将表单数据导出到 XML 文件	100
6.9 Security	101
6.9.1 如何使用密码加密 PDF 文件	101
6.10 Signature	101
6.10.1 如何对 PDF 文档进行签名，并验证签名	102
7 创建自定义工具	104
7.1 使用 Objective-C 语言创建一个区域屏幕截图工具	104
7.2 使用 Swift 语言创建一个区域屏幕截图工具	111

8 使用 Cordova 实现 Foxit PDF SDK for iOS	117
9 使用 React Native 实现 Foxit PDF SDK for iOS	118
10 使用 Xamarin 实现 Foxit PDF SDK for iOS.....	119
11 FAQ	120
11.1 Bitcode 支持.....	120
11.2 从指定的 PDF 文件路径打开一个 PDF 文档.....	120
11.3 打开 PDF 文档时显示指定的页面.....	122
11.4 License key 和序列号无法正常工作	124
11.5 在 PDF 文档中添加 link 注释.....	124
11.6 向 PDF 文档中插入图片	125
11.7 高亮 PDF 文档中的 links 和设置高亮颜色.....	126
11.8 高亮 PDF 文档中的表单域和设置高亮颜色.....	126
11.9 支持全文索引搜索	127
11.10 打印 PDF 文档	129
11.11 夜间模式颜色设置	130
11.12 Foxit SDK Framework 上传到 Apple App Store.....	131
11.13 输出 exception/crash 日志信息	131
11.14 本地化设置.....	132
12 技术支持	133

1 Foxit PDF SDK 简介

1.1 Foxit PDF SDK

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。

使用 Foxit PDF SDK 的应用开发人员可以利用 Foxit 强大、标准化的 PDF 技术安全地显示、创建、编辑、批注、格式化、管理、打印、共享，搜索 PDF 文档，以及填写 PDF 文档。此外，Foxit PDF SDK 包括一个内置可嵌入的 PDF Viewer，使得开发过程更加简单和快速。有关更多详细信息，可以访问网站 <https://developers.foxitsoftware.cn/pdf-sdk/>。

在本指南中，我们将重点介绍 Foxit PDF SDK for iOS 平台。

1.2 Foxit PDF SDK for iOS

您是否曾经为 PDF 规范的复杂性而感到不知所措？您是否曾经为被要求在有限的时间内构建一个功能齐全的 PDF 应用而感到迷茫。如果您的答案是"Yes"，那么恭喜您！您找到了在业界中快速将 PDF 功能集成到应用程序中的优选方案。

Foxit PDF SDK for iOS 致力于帮助开发人员快速将强大的 Foxit PDF 技术集成到他们自己的移动端应用程序中。通过 Foxit 开发包，即使是对 PDF 了解有限的开发人员也可以在 iOS 或者 macOS 平台上用几行代码快速构建一个专业的 PDF 阅读器。

1.2.1 为什么选择 Foxit PDF SDK for iOS

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。

Foxit PDF SDK for iOS 提供了快速 PDF 阅读和 iOS/macOS 设备的操作控制。选择 Foxit PDF SDK for iOS 的几大理由：

- **易于集成**

开发人员可以通过几行代码无缝的将 SDK 集成到他们自己的应用程序中。

- **设计完美**

Foxit PDF SDK for iOS 拥有简单、干净和友好的风格，并且提供了最好的用户体验。

- **灵活定制**

Foxit PDF SDK for iOS 提供了应用层用户界面的源代码，可以帮助开发人员对应用程序的功能和界面外观进行灵活定制。

- **移动平台上的鲁棒性**

Foxit PDF SDK for iOS 提供了 OOM (内存溢出) 恢复机制，以确保应用程序在内存有限的移动设备上运行时仍然具备较高的鲁棒性。

- **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。

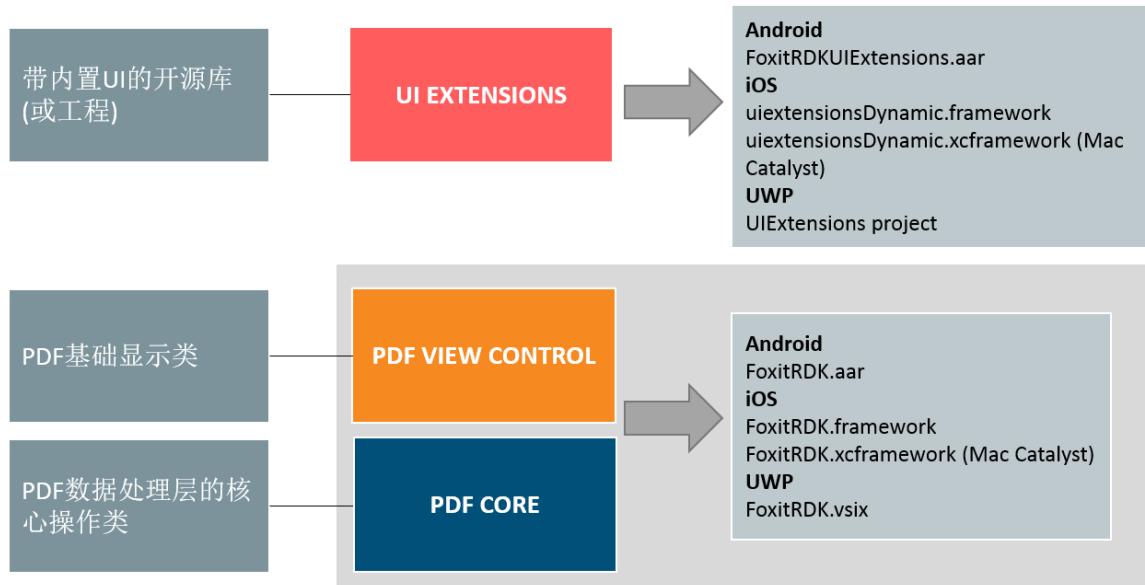
- **优秀的技术支持**

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。

1.2.2 Foxit PDF SDK for iOS 的主要框架

Foxit PDF SDK for iOS 由三个元素组成，如下图所示。Foxit PDF SDK 的所有移动平台版本共享此结构，这样便于在您的应用程序中集成，以及支持多种手机操作系统和框架。

备注：从 7.3 版本开始，Foxit PDF SDK for iOS 提供了一个新的发布包，可以支持使用 Mac Catalyst 构建 Mac 版本的 app。



Foxit PDF SDK for Android, iOS, UWP 的三种组成元素

- **PDF Core API**

PDF Core API 是 SDK 的核心部分，建立在福昕强大的底层 PDF 技术上。它提供了 PDF 基础功能操作相关的函数，包含了 PDF View 控件和 UI Extensions 组件中使用到的 PDF 核心处理功能，以确保应用程序达到高的性能和效率。该 API 可单独用于文档的渲染、分析、文本提取、文本搜索、表单填写、数字签名、压感笔迹 (PSI)、证书和密码加密、注释的创建和管理等等。

- **PDF View Control**

PDF View 控件是一个工具类，根据开发人员的需求提供开发人员与渲染的 PDF 文档进行交互所需要的功能接口。以福昕享有盛誉且使用广泛的 PDF 渲染技术为核心，View Control 支持快速高质量的渲染、缩放、滚动和页面导览功能。该 View 控件继承于平台相关 viewer 的类，例如 iOS 平台的 UIView，并且允许进行扩展来满足特定用户的需求。

- **UI Extensions 组件**

UI Extensions 组件是一个带内置 UI 的开源库，支持对内置的文本选择，标记注释、大纲导航、阅读书签、全文检索、填表、文本重排、文档附件、数字/手写签名、文档编辑和密码加密等功能进行自定义。UI Extensions 组件中的这些功能是通过使用 PDF core API 和 PDF View Control 来实现的。开发人员可以利用这些已有的 UI 实现快速构建一个 PDF 阅读器，同时可以根据需要灵活自定义其 UI 界面。

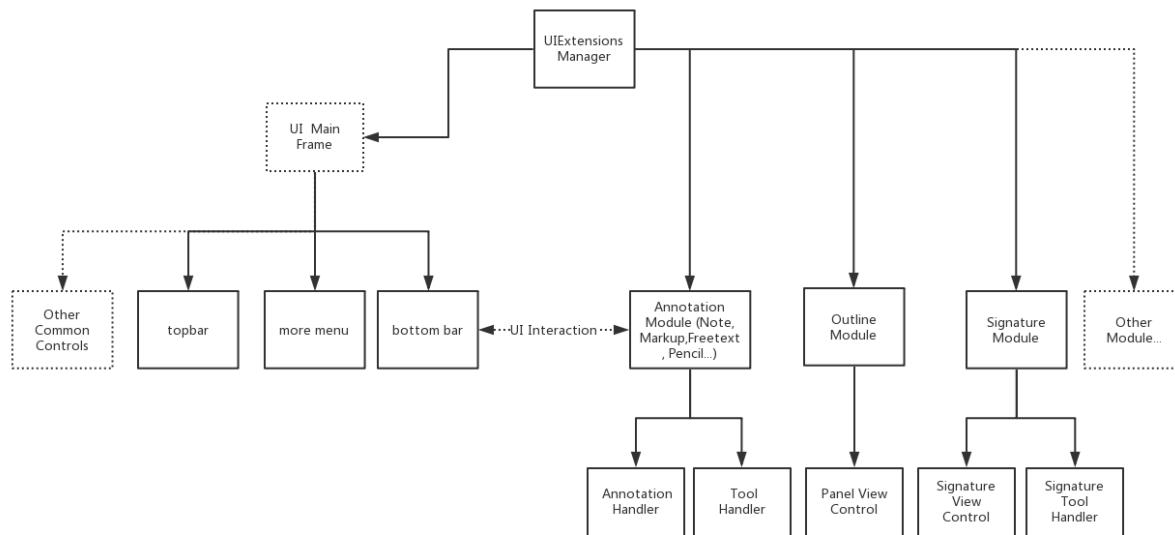
从 4.0 版本开始，Foxit PDF SDK for iOS 对 UI Extensions 组件做了一个重大的改变和优化。将基础的 UI 实现都封装到 FSPDFReader 类中，比如面板控件、工具栏设置、以及预警视图对话框等。因此，构建一个功能齐全的 PDF 阅读器变得越来越简单和容易。此外，用户可以通过一个配置文件灵活自定义他们需要的功能。

从 5.0 版本开始，Foxit PDF SDK for iOS 移除了 FSPDFReader 类，将 FSPDFReader 类中封装的 APIs 移到了 UI Extensions 组件中。内置 UI 中的任何元素都可以通过 API 来进行自定义。该版本还为开发人员提供了更高级的 APIs 和更强大的配置文件来对 UI 元素进行自定义，比如显示/隐藏某个特定的面板、top/bottom toolbar、top toolbar 中的菜单项，以及 View setting bar 和 More Menu view 中的菜单项。

1.2.3 UI Extensions 组件概述

UI Extensions 组件采用 module 机制，将每个功能细化成一个 module。当加入 UI Extensions 时，所有的 modules 除了 LocalModule(用于文件管理)会被默认自动加载。用户可以通过实现 Module 接口类来自定义 module，然后调用 UIExtensionsManager#registerModule 在当前 UIExtensions Manager 中进行注册。如果不需要使用时，可以调用 UIExtensionsManager#unregisterModule 进行反注册。

UIExtensionsManager 包含了主框架 UI，如 top/bottom toolbar，以及各个模块之间共享的 UI 组件。同时，各个功能模块也可以通过 UIExtensionsManager 来进行单独加载。功能模块在加载的时候会对主框架 UI 进行适配和调整，并且建立起消息事件响应的联系。各个功能模块可能包含了其模块特有的 UI 组件，同时也会有自己独立的消息事件处理逻辑。UIExtensionsManager 也会负责将从 View Control 组件接收到的消息和事件分发到各个功能模块中去。下面的图片讲述了 UIExtensionsManager 和 modules 之间的详细关系。



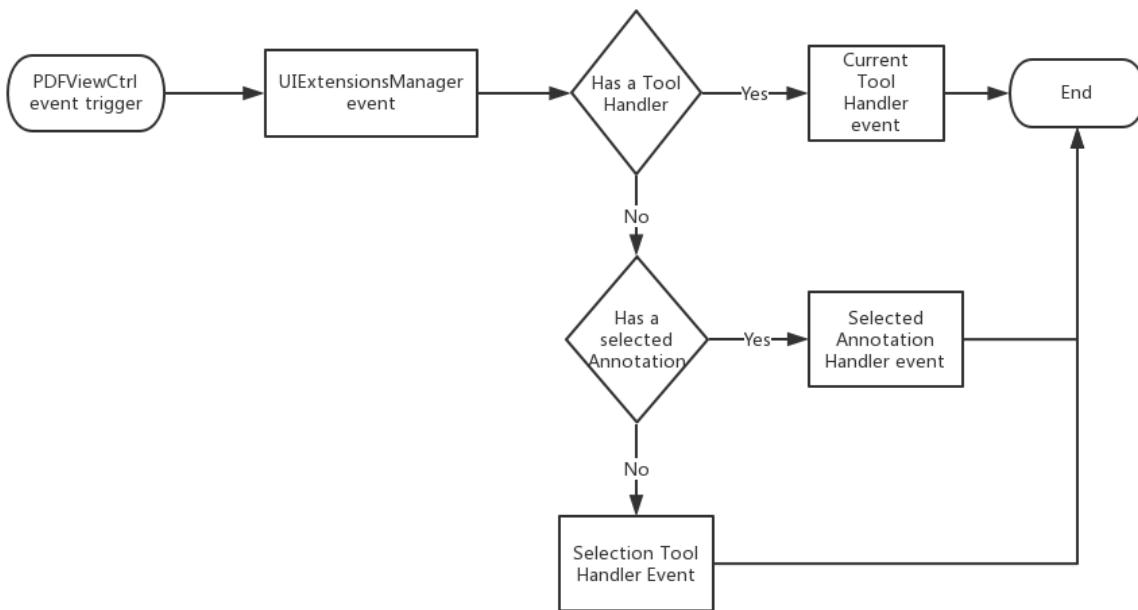
UIExtensionsManager 和 modules 之间的关系

Tool handler 与 annotation handler 处理来自 PDFViewCtrl 的触屏、手势等事件。当触屏和手势事件触发时，PDFViewCtrl 会将相应的事件传递给 UIExtensionsManager：

- 如果当前存在 tool handler，UIExtensionsManager 会将相应的事件传递给当前的 tool handle，然后事件处理过程结束。
- 如果当前有选择 annotation，UIExtensionsManager 会将相应的事件传递给当前所选择的 annotation 对应的 annotation handler，然后事件处理过程结束。
- 如果当前不存在 tool handler，也没有选中的 annotation，那么 UIExtensionsManager 会将相应的事件传递给 selection tool handler。Text Selection tool 用于文本选择相关事件的处理，例如选择一段文本添加 highlight annotation。Blank Selection tool 用于空白处相关事件的处理，例如在空白处添加 Note annotation。

备注：Tool Hander 和 Annotation Handler 不会同时响应事件。Tool Handler 主要用于 annotation 的创建(目前不支持 Link Annotation 的创建)、signature 的创建和文本选择。Annotation Handler 主要用

于 annotation 的编辑以及表单填写。下图讲述了 Tool Handler 和 Annotation Handler 之间的事件响应流程。



Tool Handler 和 Annotation Handler 之间的事件响应流程

1.2.4 Foxit PDF SDK for iOS 的主要功能

Foxit PDF SDK for iOS 包括了一些主要的功能，用来帮助应用程序开发人员在快速实现他们所需要的功能的同时减少开发成本。

备注：从 7.1 版本开始，Foxit PDF SDK for iOS 支持 iOS 13 的新特性 Dark Mode。工具条框架中的所有内置控件可以调整其外观来无缝地匹配当前系统的外观。

功能	描述
PDF Document	打开和关闭文件，设置和获取 metadata。
PDF Page	解析、渲染、阅读、编辑文档页面。
Render	平台图像设备在 bitmap 上创建图像渲染引擎。
Reflow	重排页面内容。

Crop	裁剪 PDF 页面。
Text Select	文本选择。
Text Search	文本搜索，并且支持全文索引搜索。
Outline	定位和链接到文档中的兴趣点。
Reading Bookmark	标记文档中感兴趣的页面和段落位置。
Annotation	创建、编辑和移除 annotations。
Layers	添加、编辑和移除 PDF 层内容。
Attachments	添加、编辑和移除文档级的附件。
Form	支持 JavaScript 填表，通过 XFDF/FDF/XML 文件导入和导出表单数据。 支持创建文本域、复选框、单选按钮、组合框、列表框和签名域。
XFA	支持静态和动态 XFA。
Signature	签名 PDF 文档，验证签名，添加或删除签名域。 添加和验证第三方数字签名。 支持签名的长期验证(LTV)。
Fill	用文本和符号填写扁平化表单（即非交互式表单）
Security	密码和证书加密 PDF 文档。
Pan and Zoom	调整视图中的放大倍数和位置以匹配 Pan&Zoom 缩略视图中的矩形区域。
Print	打印 PDF 文档。
RMS	支持微软 IRMv1 和 IRMv2 标准的 RMS 解密。
Comparison	对比两个 PDF 文档，并且标记文档之间的差异。
Scanning	扫描纸质文档，并将其转换为 PDF 文档。
Speak	支持阅读 PDF 文档中的文本。
Split Screen	支持分屏。
Out of Memory	从内存不足中恢复运行。

备注：*Outline* 是 PDF 规范中的技术术语，在传统的桌面 PDF 阅读器中常叫做书签。*Reading bookmarks* 常用于移动端和平板的 PDF 阅读器中，用来标记阅读进度或者用户感兴趣的段落。*Reading bookmark* 在技术上并不是 *outline*，它存储在应用程序中而不是 PDF 本身。

Foxit PDF SDK for iOS 支持鲁棒性的 PDF 应用程序

在有限内存的移动平台上开发鲁棒性的 PDF 应用程序是具有挑战性的。当内存分配失败，应用程序可能会 crash 或者意外退出。为了解决这个问题，Foxit PDF SDK for Android 提供了一种内存溢出 (OOM) 机制。

OOM 是 Foxit PDF SDK for iOS 的一个高级功能，因为其本身的复杂性。OOM 机制的关键点是 Foxit PDF SDK for iOS 会监视内存的使用情况，并在检测到 OOM 后自动执行恢复操作。在恢复的过程中，Foxit PDF SDK for iOS 会自动重新加载文档和页面，将恢复到发生 OOM 之前的原始状态。这意味着当前阅读的页面和位置，以及页面阅读模式(单页或者连续页面)都能够恢复，但是编辑相关的内容将会丢失。

Foxit PDF SDK for iOS 在 FSPDFViewCtrl 类中提供一个属性"shouldRecover"。默认情况下，"shouldRecover" 为 "YES"。如果在检测到 OOM 时您不想开启自动恢复机制，您可以将 "shouldRecover" 设置为 "No"，如下所示：

```
self.pdfViewControl = [[FSPDFViewCtrl alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
self.pdfViewControl.shouldRecover = NO;
```

此时，应用程序将会抛出异常，可能会crash或者意外退出。

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 iOS 集成到自己的应用程序中的开发人员。它旨在介绍以下章节：

- Section 1: 介绍 Foxit PDF SDK，特别是 iOS 平台的 SDK。
- Section 2: 说明包的结构，以及运行 demos。
- Section 3: 介绍如何快速创建功能齐全的 PDF 阅读器。
- Section 4: 介绍如何使用 Mac Catalyst 快速创建功能齐全的 PDF 阅读器。
- Section 5: 介绍如何自定义用户界面。

- Section 6: 介绍如何使用 Foxit PDF SDK core API。
- Section 7: 介绍如何创建自定义工具。
- Section 8: 介绍使用 Cordova 实现 Foxit PDF SDK。
- Section 9: 介绍使用 React Native 实现 Foxit PDF SDK。
- Section 10: 介绍使用 Xamarin 实现 Foxit PDF SDK。
- Section 11: 列出常见问题。
- Section 12: 提供技术支持信息。

2 入门指南

安装并集成 Foxit PDF SDK for iOS 非常简单。您只需要几分钟就能见证其强大的功能。本指南主要介绍如何在 iOS 或 macOS 平台使用 Foxit PDF SDK。本章的主要内容是包结构的介绍以及如何运行 demo。

2.1 系统要求

备注: 从 7.5.1 版本开始, Foxit PDF SDK for iOS 只支持 64 位的设备, 因为在 iOS 11 或者更高版本系统中, 所有的应用都使用 64 位架构, 具体请参阅 [苹果官方开发者手册](#)。

不支持 Mac Catalyst 的发布包:

- iOS 11.0 或更高版本
- Xcode 9.0 或更高版本

备注: 如果 iOS 是 13 或更高的版本, 那么 Xcode 需要使用 11 或更高版本。

支持 Mac Catalyst 的发布包:

- macOS 10.15 or higher
- Xcode 11 or higher

2.2 包结构说明

Foxit PDF SDK for iOS 提供了如下的两种包:

- foxitpdfsdk_8_1_ios.zip: 构建的 app 只能部署到 iPhone 或者 iPad 设备上。
- foxitpdfsdk_8_1_ios_catalyst.zip: 构建的 app 可以部署到 iPhone、iPad 或者 Mac 设备上。

备注: 如果您需要在 macOS 系统上部署 app, 请选择 foxitpdfsdk_8_1_ios_catalyst.zip 包。

2.2.1 不支持 Mac Catalyst 的发布包

下载 "foxitpdfsdk_8_1_ios.zip" 包, 解压到一个新的目录如 "foxitpdfsdk_8_1_ios", 如 Figure 2-1 所示。其中解压包中包括如下的内容:

docs:
icc_profile

API 手册, 开发文档和升级说明文档
输出预览 (output preview) 功能所使用的默认 icc profile 文件

libs:	License 文件, AAR, UI Extensions 组件源代码
samples:	Android 示例工程
getting_started_ios.pdf:	Foxit PDF SDK for iOS 快速入门
legal.txt:	法律和版权信息
release_notes.txt:	发布信息

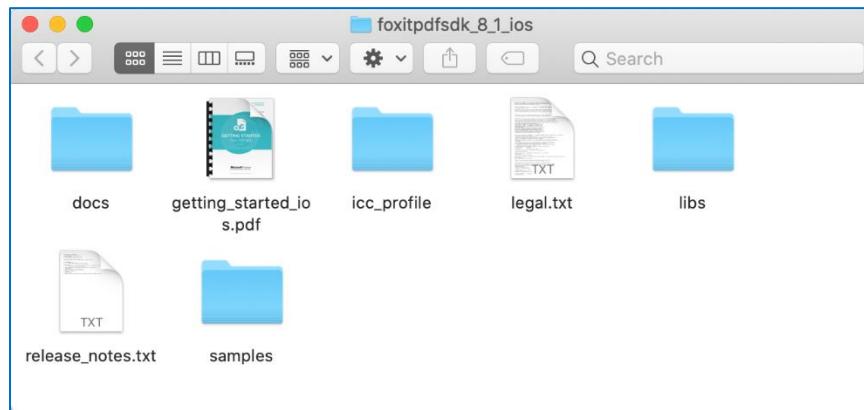


Figure 2-1

如 Figure 2-2 所示的 "libs" 文件夹下是 Foxit PDF SDK for iOS 的核心组件, cocoapods 工具的配置文件, 以及用于剥离 arm 架构的脚步文件。

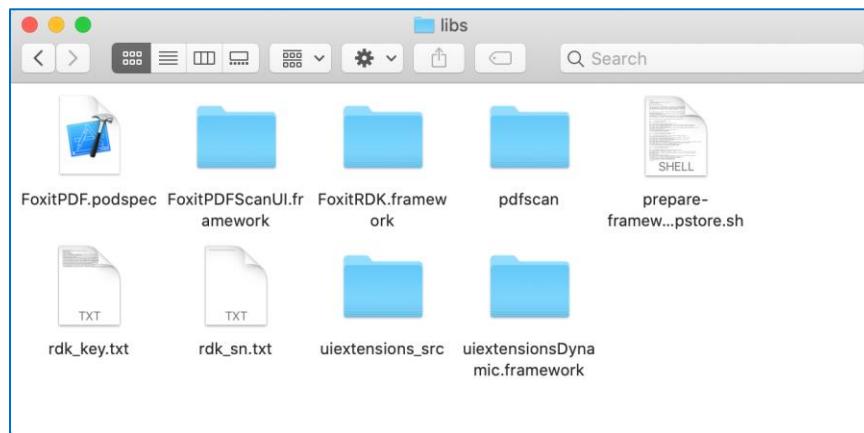


Figure 2-2

- **FoxitRDK.framework** – 该 framework 包含 Foxit PDF SDK for iOS 的动态库和相关头文件。
- **uiextensionsDynamic.framework** – 该 framework 包括 UI Extensions 动态库, 相关头文件, 以及默认内置 UI 实现需要的资源文件。
- **FoxitPDFScanUI.framework** – 该 framework 包括 Foxit PDF SDK 扫描功能动态库, 相关头文件, 以及扫描功能默认内置 UI 实现需要的资源文件。

- **FoxitPDF.podspec** – cocoaPods 工具的配置文件，用于管理第三方库。该配置文件在 "[使用 React Native 实现 Foxit PDF SDK for iOS](#)" 章节中有用到。
- **prepare-framework-to-publish-to-appstore.sh** – 用于将 arm 架构从 Foxit SDK Framework 中剥离的脚本文件。Foxit SDK Framework 包括 arm64, armv7, i386 和 x86_64 架构，但是 i386 和 x86_64 架构的 framework 不允许上传到 Apple App Store。
- **pdfscan** 工程 - 是一个开源库，包含了扫描功能相关的 UI 实现，可以帮助开发人员快速将扫描功能集成到他们的 iOS 应用中，或者根据需要自定义扫描功能的 UI。
- **uiextensions** 工程 - 在 "libs/uiextensions_src" 文件夹下。它是一个开源库，包含了一些即用型的 UI 模块实现和应用程序基本的 UI 设计，可以帮助开发人员快速将功能齐全的 PDF 阅读器嵌入到他们的 iOS 应用中。当然，开发人员也不是必须要使用默认的 UI，可以通过 "uiextensions" 工程为特定的应用灵活自定义和设计 UI。

备注：对于 iOS 13 或更高版本，您需要使用 Xcode 11 或者更高版本来编译 "uiextensions" 工程。

2.2.2 支持 Mac Catalyst 的发布包

下载 "foxitpdfsdk_8_1_ios_catalyst.zip" 包，并解压到一个新的目录。该包中的内容和 "foxitpdfsdk_8_1_ios.zip" 包类似，具体说明请参阅上一小节 "[支持 Mac Catalyst 的发布包](#)"。

如 Figure 2-3 所示的 "libs" 文件夹，其区别是 FoxitRDK.xcframework 和 uiextensionsDynamic.xcframework 支持使用 Mac Catalyst 构建 Mac 版本的 iPad app。

备注：当前，Mac 版本 app 不支持 RMS 和扫描功能。

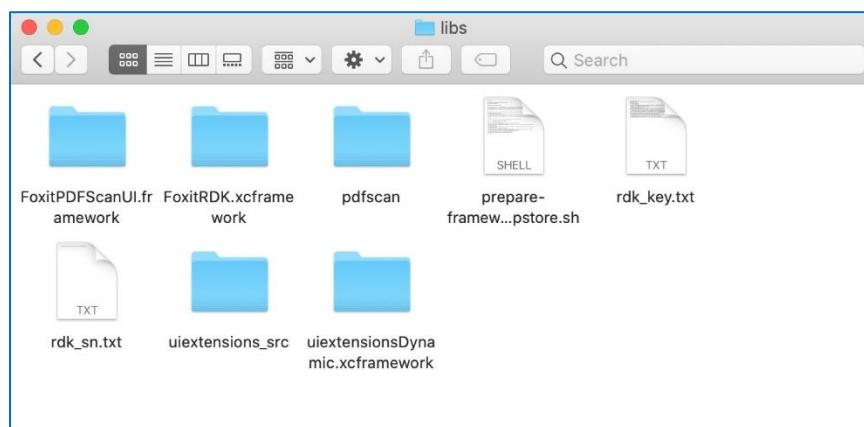


Figure 2-3

2.3 运行 demo

下载和安装 Xcode IDE (<https://developer.apple.com/download/>)。

备注: 在本指南中, 不具体介绍 Xcode 的安装步骤。如果您还没有安装, 请参考 Apple 的开发官网。

如 Figure 2-4 所示, **Foxit PDF SDK for iOS** 提供了 Objective-C 和 Swift 两种语言的三种不同类型的 demos (Function demo, Viewer Control demo, 和 Complete PDF viewer demo), 用来帮助开发人员学习如何在 iOS 平台上调用 Foxit SDK。Swift demos 在"swift" 文件夹下。

备注: Swift 语言的 complete PDF viewer demo 支持多文档阅读模式。

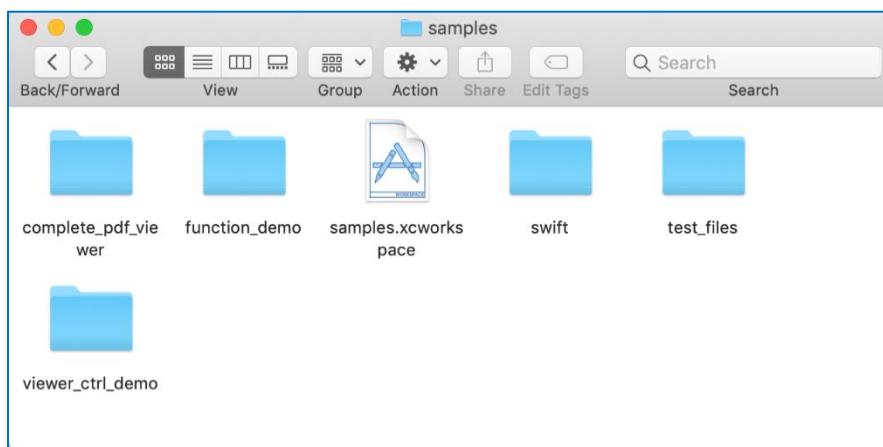


Figure 2-4

Foxit PDF SDK for iOS (Mac Catalyst) 提供了一个 Objective-C 语言的 Complete PDF viewer demo, 用来帮助开发人员学习如何调用 Foxit SDK。该 demo 位于 "samples" 文件夹下。运行该 demo, 请确保您 macOS 是 10.15 或者更高版本, 以及 Xcode 是 11 或者更高版本。

2.3.1 Function demo

Function demo (包括 Objective-C 和 Swift 两种语言) 用来展示如何使用 Foxit PDF SDK for iOS 的 core API 来实现一些 PDF 特定的功能。该 demo 包括如下的功能:

- **pdf2txt:** 从 PDF 文档中提取文本, 并保存到一个 TXT 文件中。
- **outline:** 编辑大纲(也称为书签)的外观和主题。
- **annotation:** 向 PDF 页面添加 annotations 和以 JSON 文件格式导出 annotations。
- **docinfo:** 导出 PDF 文档信息到 TXT 文件中。
- **render:** 将指定的 PDF 页面渲染为位图。
- **signature:** 向 PDF 中添加签名, 对 PDF 签名和验证签名。

- **image2pdf:** 将图像转化为 PDF 文件。
- **watermark:** 向 PDF 文件添加文本、图像和 PDF 页面水印。
- **search:** 搜索 PDF 文件。
- **graphics_objects:** 使用图形对象创建 PDF 文档。

在 Xcode 中运行该 demo，请按如下的步骤：

- a) 在 "samples/complete_pdf_viewer" 文件夹下，双击 **function_demo.xcodeproj** 在 Xcode 中打开该 demo。(对 Swift demo，在 "samples/swift/function_demo_swift"文件夹下，双击 **function_demo_swift.xcodeproj**)。

备注：另一种打开 demo 的方式是：双击 "samples" 文件夹下的 **samples_xcworkspace**，它是一个包含三个 demos 的 workspace。

- b) 点击 "Product -> Run" 在 iOS 设备或者模拟器上运行 demo。在本指南中，将使用 iPhone 11 模拟器来运行该 demo。当成功编译后，您可以看到如 Figure 2-5 所示的功能选项。

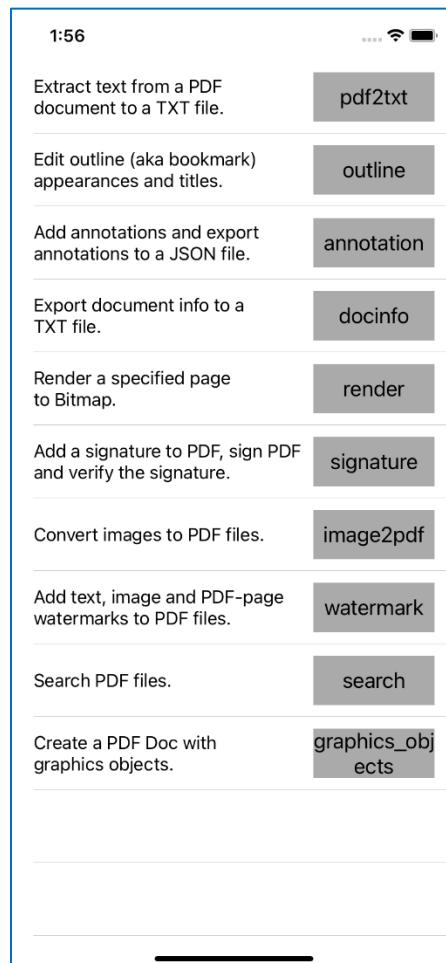


Figure 2-5

- c) 点击上图中的功能按钮去执行相应的操作。例如，点击 "pdf2txt"，则会弹出如 Figure 2-6 所示的消息框，提示转换后的文本文件保存的位置。请运行 demo 并自由体验其功能。

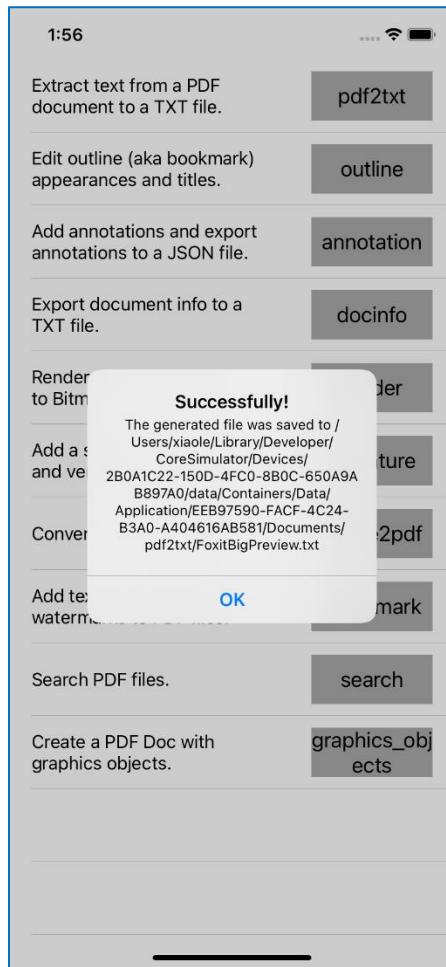


Figure 2-6

2.3.2 Viewer control demo

Viewer control demo (包括 Objective-C 和 Swift 两种语言) 用来展示如何使用 Foxit PDF SDK for iOS 实现与 View Control 功能层相关的功能，比如操作 annotations (注释、打字机、高亮、下划线、删除线、波浪线等)，大纲，阅读标记书签 (reading bookmarks) 和文本搜索。该 demo 的代码逻辑结构非常清晰和简单，开发人员可以快速定位 PDF 应用，比如 PDF 阅读器中某个功能的具体实现。并且通过这个 demo，开发人员可以更进一步的接触和了解 Foxit PDF SDK for iOS 所提供的 APIs。

在 Xcode 中运行该 demo，请参考 [Function demo](#) 中的步骤。

Demo 成功运行后会如 Figure 2-7 所示。这里，使用 iPhone 11 模拟器来运行 demo。



Figure 2-7

该 demo 提供了文本搜索、阅读标记书签、大纲、注释和数字签名等功能。例如，点击 ，选择第二个选项卡 (outline)，然后文档的大纲将显示如 Figure 2-8 所示。

备注：Outline 是 PDF 规范中的技术术语，在传统的桌面 PDF 阅读器中常叫做书签。Reading bookmarks 常用于移动端和平板的 PDF 阅读器中，用来标记阅读进度或者用户感兴趣的段落。Reading bookmark 在技术上并不是 outline，它存储在应用程序中而不是 PDF 本身。

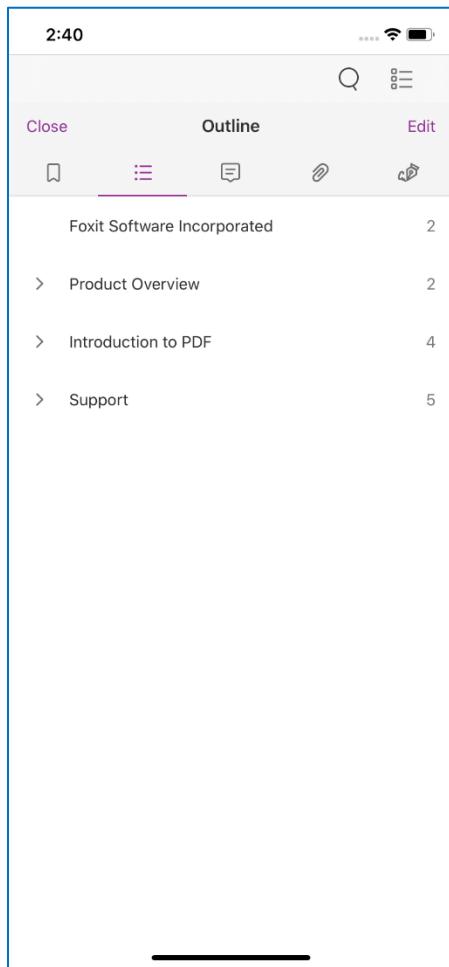


Figure 2-8

2.3.3 Complete PDF viewer demo

2.3.3.1 不支持 Mac Catalyst 的 Complete PDF Viewer demo

Complete PDF Viewer demo 阐述了如何通过使用 Foxit PDF SDK for iOS 实现一个功能齐全的 PDF 阅读器，该阅读器几乎可以作为实际移动端的 PDF 阅读器使用。该 demo 使用了 Foxit PDF SDK for iOS 所提供的所有功能和内置 UI 实现。

备注： Swift 语言的 complete PDF viewer demo 支持多文档阅读模式。

在 Xcode 中运行该 demo，请参考 [Function demo](#) 中的步骤。

这里，将使用 iPhone 11 模拟器来编译和运行 Swift 语言的 complete PDF viewer demo。当成功运行后，屏幕会列出 "Sample.pdf" 和 "complete_pdf_viewer_guide_ios.pdf" 文档。如果您想要阅读多个文档，点击 切换到多文档阅读模式 (如 Figure 2-9 所示)。

备注：如果您想用其他的PDF文档来测试该demo，您需要将其放置到设备的"Document"文件夹下。

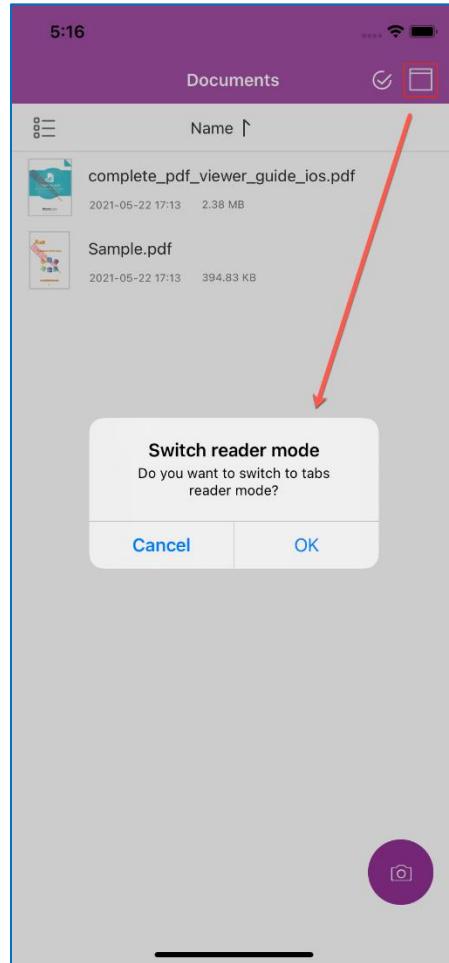


Figure 2-9

点击**OK**切换到多文档阅读模式。选择"complete_pdf_viewer_guide_ios.pdf"文档，点击Back按钮`<`，再选择"Sample.pdf"，如Figure 2-10所示。现在，您可以通过切换选项卡浏览这两个文档。

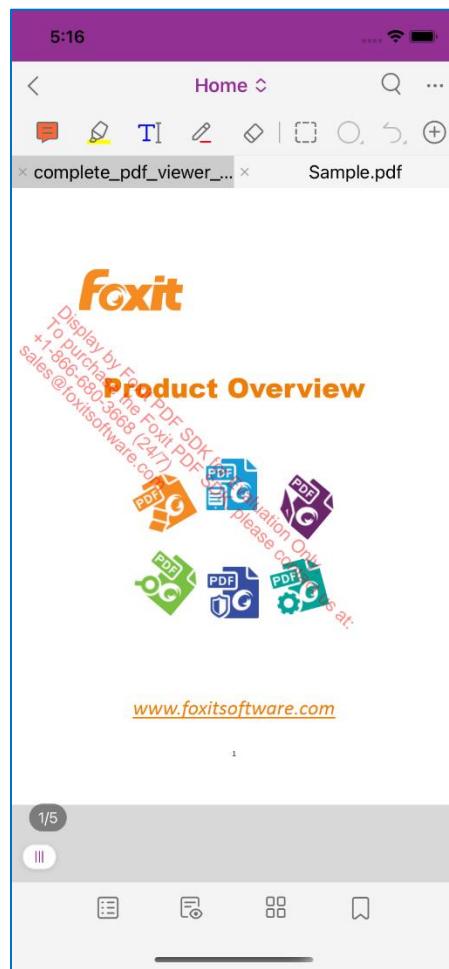


Figure 2-10

该 demo 实现了一个功能齐全的 PDF 阅读器，请随意体验。

例如，它提供了页面缩略图功能。您可以点击底部工具栏上的缩略图菜单 ，然后可以看到如 Figure 2-11 所示的文档的缩略图。

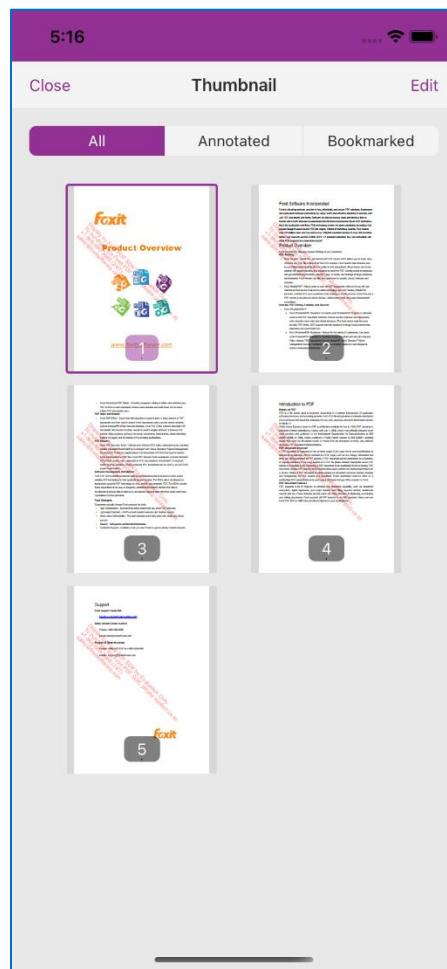


Figure 2-11

2.3.3.2 支持 Mac Catalyst 的 Complete PDF Viewer demo

在 Xcode 中运行该 demo，请按如下的步骤：

- 在 "samples/complete_pdf_viewer" 文件夹下，双击 **complete_pdf_viewer.xcodeproj** 在 Xcode 中打开该 demo。
- 点击 "Product -> Run" 在 iOS 设备或者模拟器上运行 demo，或者在您的 Mac 电脑上运行该 demo。此处，选择 "**My Mac**" 来运行 demo。当成功编译后，屏幕会列出"Sample.pdf" 和 "complete_pdf_viewer_guide_ios.pdf" 文档，如 Figure 2-12 所示。

该 demo 包含的功能与 "[不支持 Mac Catalyst 的 Complete PDF Viewer demo](#)" 类似。

备注：您可能需要使用开发配置文件 (development provisioning profile) 注册您的 Mac 电脑，以允许该 demo 可以在 Mac 上启动，以及在开发中使用某些应用服务。

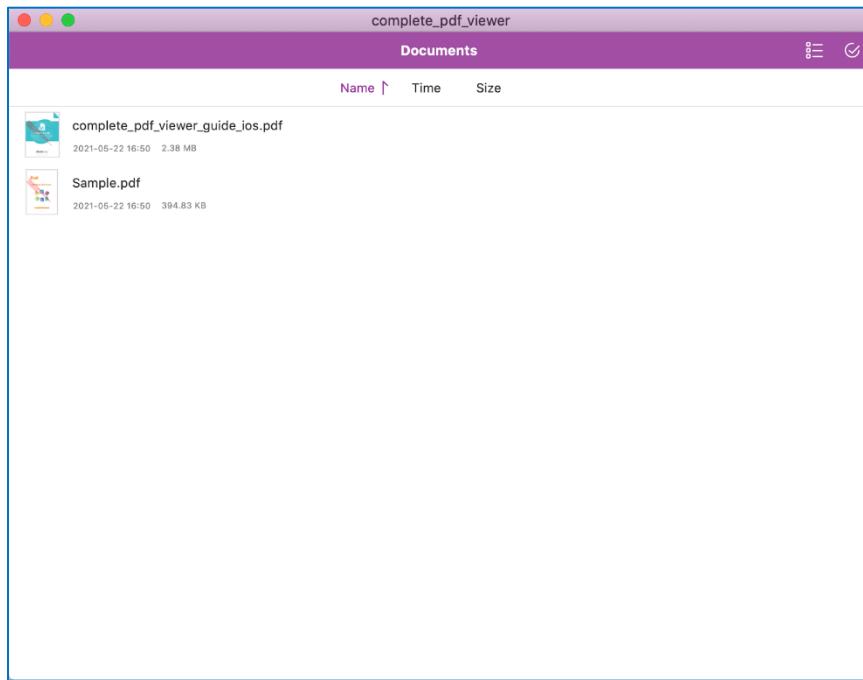


Figure 2-12

3 快速构建一个功能齐全的 PDF 阅读器

Foxit PDF SDK for iOS 将所有的 UI 实现（包括应用程序的基本 UI 和即用型 UI 功能模块）封装在 UI Extensions 组件中，因此开发人员可以轻松快速通过几行代码构建一个功能齐全的 PDF 阅读器。本章将提供详细的教程来帮助您快速开始使用 Foxit PDF SDK for iOS 创建 Objective-C 和 Swift 两种语言的功能齐全的 PDF 阅读器（只支持部署到 iPhone 或 iPad 设备上）。

3.1 使用 Foxit PDF SDK for iOS 构建一个 Objective-C 的 iOS 应用程序

本节将阐述如何使用 Foxit PDF SDK for iOS 快速构建一个 Objective-C 语言的 iOS 应用程序。其主要包括以下的步骤：

- [创建一个 Objective-C 语言的 iOS 工程](#)
- [集成 Foxit PDF SDK for iOS 到您的应用程序](#)
- [初始化 Foxit PDF SDK for iOS](#)
- [使用 FSPDFViewCtrl 显示 PDF 文档](#)
- [使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器](#)
- [基于功能齐全的 PDF 阅读器添加扫描功能](#)

3.1.1 创建一个 Objective-C 语言的 iOS 工程

在本指南中，使用 Xcode 12.0.1 创建一个新的 iOS 工程。

启动 Xcode，选择 **File -> New -> Project...**，然后选择 **iOS -> Single View App**，如 Figure 3-1 所示。点击 **Next.**

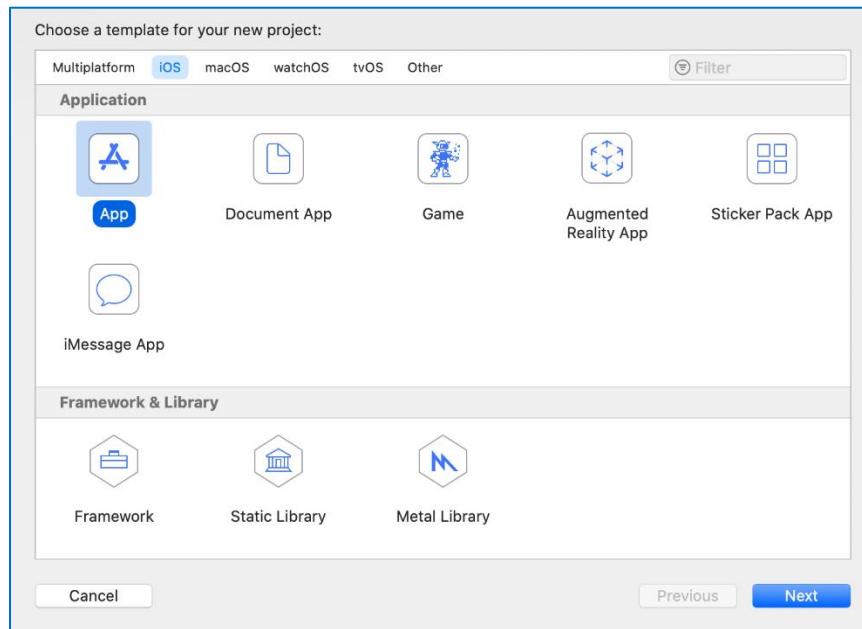


Figure 3-1

为该工程选择如 Figure 3-2 所示的选项。请确保选择 Objective-C 语言。为简单起见，我们不勾选用
于自动测试的单元测试和 UI 测试。然后，点击 **Next**。

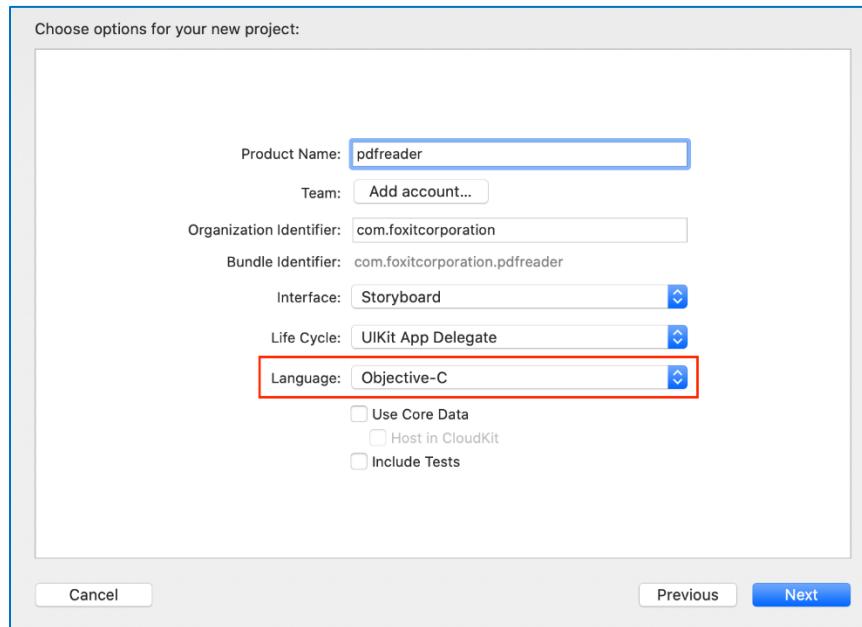


Figure 3-2

放置该工程到所需的位置。"Source control" 选项对构建您的第一个 PDF 应用程序来说不是很重要，因此不用勾选 Git repository。这里，我们将该工程放置在桌面，如 Figure 3-3 所示。然后，点击 **Create**。

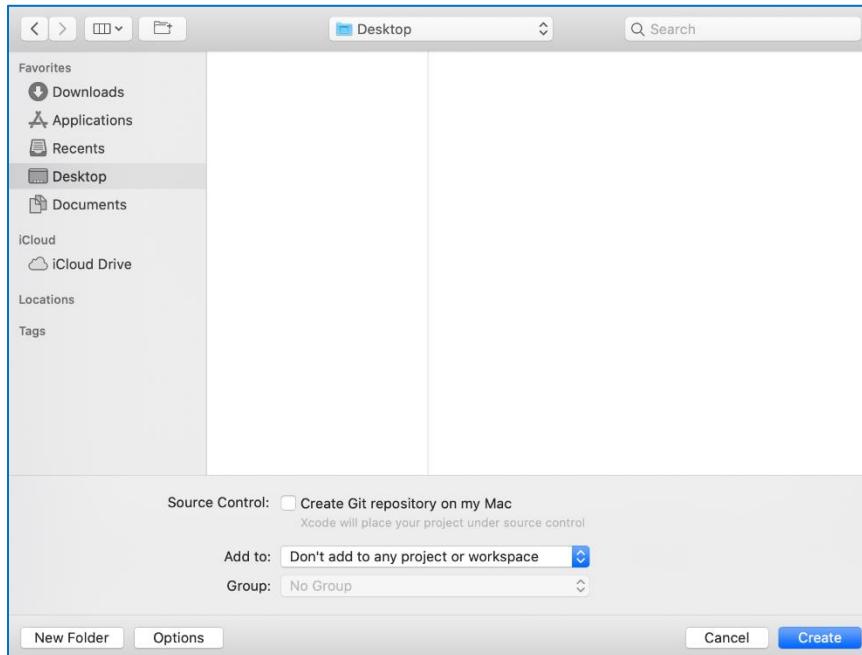


Figure 3-3

3.1.2 集成 Foxit PDF SDK for iOS 到您的应用程序

备注: 在本章中, 我们将使用默认的内置 UI 实现来开发该应用程序, 为了简单和方便(直接使用 UI Extensions 组件, 不需要源代码工程), 我们只需要添加以下的文件到 PDFReader 工程中。

- **FoxitRDK.framework** – 该 framework 包含 Foxit PDF SDK for iOS 的动态库和相关头文件。
- **uiextensionsDynamic.framework** – 该 framework 包括 UI Extensions 动态库, 相关头文件, 以及默认内置 UI 实现需要的资源文件。
- **(可选) FoxitPDFScanUI.framework** – 该 framework 包括 Foxit PDF SDK 扫描功能动态库, 相关头文件, 以及扫描功能默认内置 UI 实现需要的资源文件。

技巧:

- 在接下来的 "初始化 Foxit PDF SDK for iOS" 和 "使用 PDFViewController 显示 PDF 文档" 两小节中, 不需要使用 UI Extensions 组件 (**uiextensionsDynamic.framework**), 因此您可以先将 **FoxitRDK.framework** 添加到工程中。然后在您需要使用 UI Extensions 组件时再添加 **uiextensionsDynamic.framework**, 比如在 "使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器" 章节中。
- 扫描 (Scan) 模块需要使用 **FoxitPDFScanUI.framework**, 比如在 "基于功能齐全的 PDF 阅读器 添加扫描功能" 章节中。

添加上述的三个动态 framework 文件到 *pdfreader* 工程, 请按照如下的步骤:

- a) 右击 "pdfreader" 工程, 选择 **Add Files to "pdfreader"...**, 如 Figure 3-4 所示。

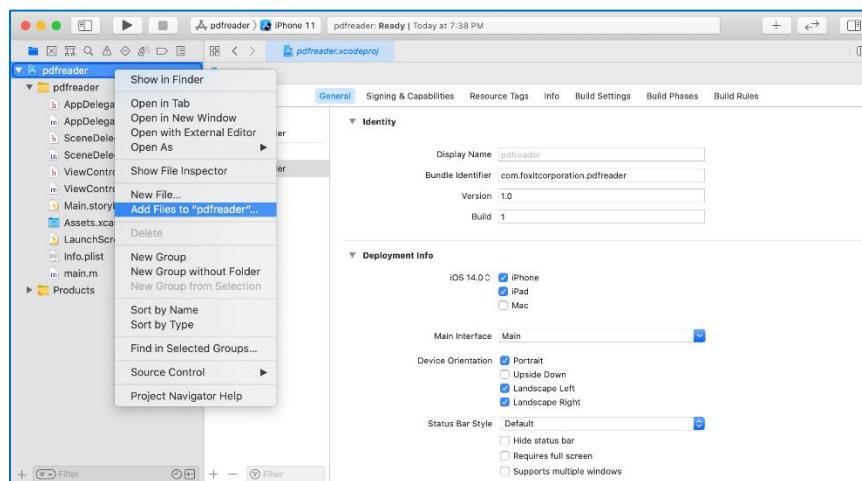


Figure 3-4

- b) 定位到下载解压包中的 "libs" 文件夹下，选择 "**FoxitRDK.framework**"，然后点击 **Add**，如 Figure 3-5 所示。

备注：请确保勾选 "**Copy items if needed**" 选项。

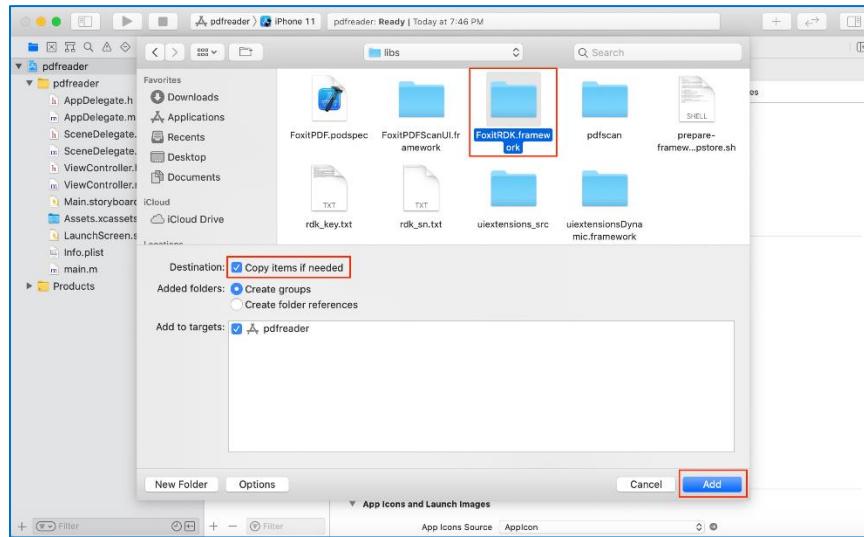


Figure 3-5

- c) 重复步骤 a) 和 b)，添加 "**uiextensionsDynamic.framework**" 和 "**FoxitPDFScanUI.framework**"。然后，*pdfreader* 工程将如 Figure 3-6 所示。

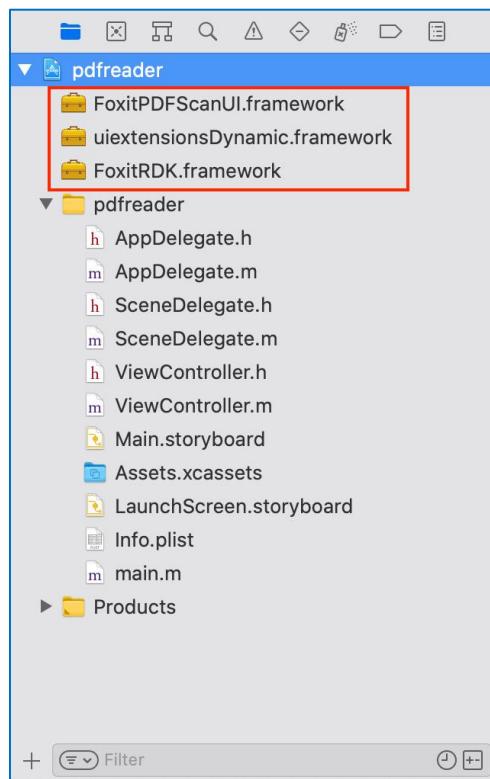


Figure 3-6

- d) 嵌入动态 framework。单击工程，在 **General** 选项卡下找到 **Frameworks, Libraries, and Embedded Content**，然后选择 "Embed & Sign"，如 Figure 3-7 所示。

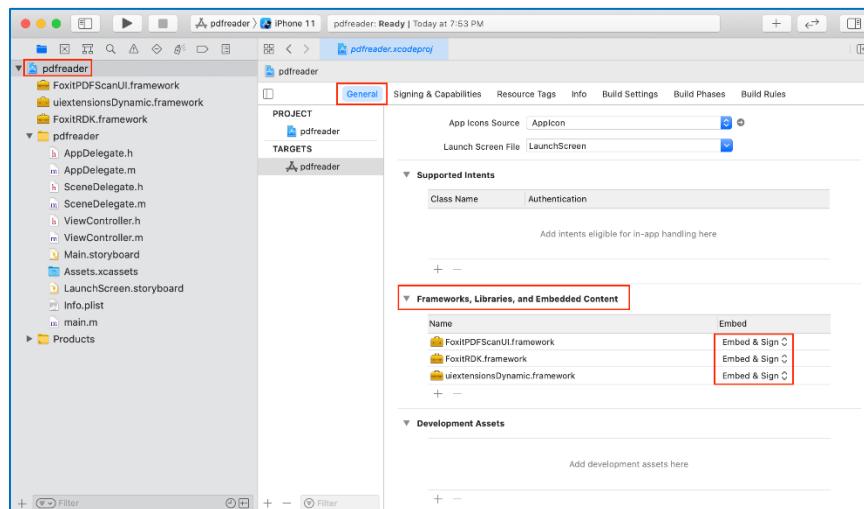


Figure 3-7

至此，我们已成功在 *pdfreader* 工程中添加了 "**FoxitRDK.framework**"，"**uiextensionsDynamic.framework**" 和 "**FoxitPDFScanUI.framework**"。

3.1.3 初始化 Foxit PDF SDK for iOS

在调用任何 API 之前，应用程序必须使用 license 初始化 Foxit PDF SDK for iOS。[*FSLibrary initialize:sn key:key*] 函数用于 SDK 库的初始化。试用 license 文件在下载包的"libs"文件夹下。当试用期结束后，您需要购买正式 license 以继续使用该 SDK。在 AppDelegate.m 文件的 didFinishLaunchingWithOptions 方法中完成 SDK 的初始化。

```
#import "AppDelegate.h"
#import <FoxitRDK/FSPDFObjC.h>

@interface AppDelegate : NSObject<UIApplicationDelegate>

@end

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // The value of "sn" can be found in the "rdk_sn.txt".
    // The value of "key" can be found in the "rdk_key.txt".
    NSString *sn = @"";
    NSString *key = @_;

    FSErrorCode eRet = [FSLibrary initialize:sn key:key];
    if (FSErrSuccess != eRet) {
        return NO;
    }
    return YES;
}

@end
```

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

3.1.4 使用 FSPDFViewController 显示 PDF 文档

到目前为止，我们已经在 *pdfreader* 工程中添加了 Foxit PDF SDK for iOS frameworks，并且完成了 SDK 库的初始化。现在，我们将使用 FSPDFViewController 通过几行代码来显示一个 PDF 文档。

备注：如果只需要显示一个 PDF 文档，则不需要 UI Extensions 组件。

首先，在工程中添加一个 PDF 文档作为测试文档。例如，使用下载解压包中 "samples\test_files" 文件夹下的 "Sample.pdf" 文档。右击 *pdfreader* 工程，选择 **Add Files to "pdfreader"**... 添加该文档。当添加完成后，您可以在 Xcode 的 **Copy Bundle Resources** 看到该文档，如 Figure 3-8 所示。

备注：您可以直接添加PDF到**Copy Bundle Resources**。单击pdfreader工程，在**Build Phases**选项卡下找到**Copy Bundle Resources**，点击+按钮，选择需要添加的文档。您可以使用任何PDF文档，只需要将其添加到Xcode的**Copy Bundle Resources**。

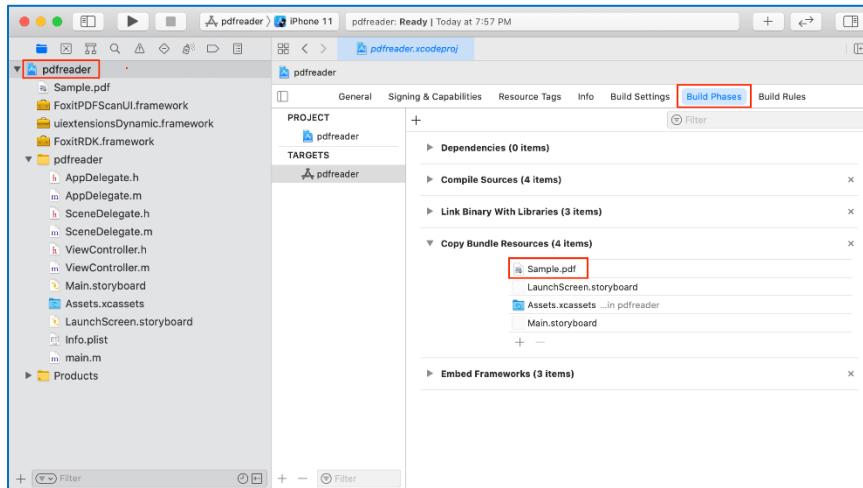


Figure 3-8

其次，在 ViewController.m 中加入如下的代码来打开并显示一个 PDF 文档。显示一个 PDF 文档很简单，您只需要获取一个 PDF 文档路径，实例化一个 FSPDFViewCtrl 对象，然后调用 [FSPDFViewCtrl openDoc: filePath passwork: password completion: completion] 函数即可。

更新 ViewController.m，如下所示：

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>

@interface ViewController : UIViewController

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF.
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];

    // Initialize a FSPDFViewCtrl object with the size of the entire screen.
    FSPDFViewCtrl* pdfViewCtrl;
    pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame:[self.view bounds]];

    // Open an unencrypted PDF document.
    [pdfViewCtrl openDoc:pdfPath password:nil completion:nil];
}
```

```
// Add the pdfViewCtrl to the root view.  
[self.view addSubview:pdfViewCtrl];  
  
}  
  
- (void)didReceiveMemoryWarning {  
    [super didReceiveMemoryWarning];  
    // Dispose of any resources that can be recreated.  
}  
  
@end
```

太棒了！我们已经使用 Foxit PDF SDK for iOS 通过几行代码完成了一个显示 PDF 文档的简单的 iOS 应用程序 (Objective-C 语言)。下一步是在真机设备或者模拟器上运行该工程。

在本指南中，将使用 iPhone 11 模拟器来编译和运行该工程。当成功编译后，您将看到"Sample.pdf" 文档显示如 Figure 3-9 所示。该示例应用程序具有一些基本的 PDF 功能，比如放大/缩小和翻页。您可以进行体验。

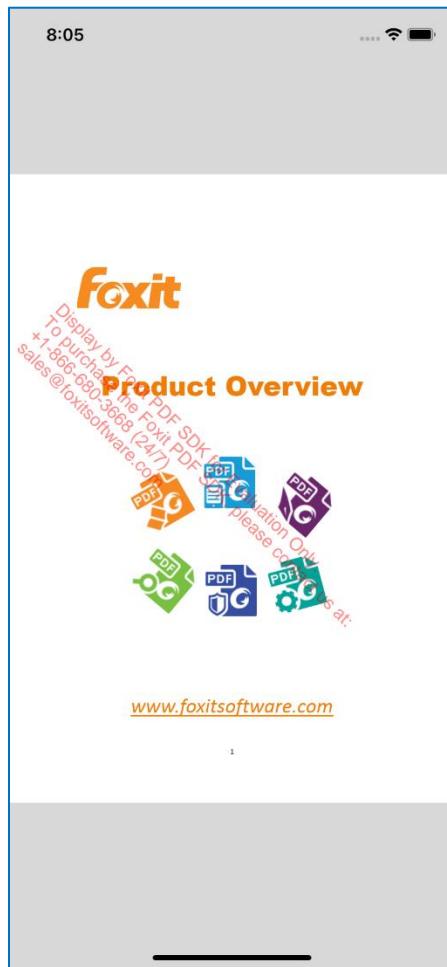


Figure 3-9

3.1.5 使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器

Foxit PDF SDK for iOS 带有内置的 UI 设计，包括应用程序的基础 UI 和功能模块 UI。内置 UI 通过 Foxit PDF SDK for iOS 实现，并且封装在 UI Extensions 组件中。因此，构建一个功能齐全的 PDF 阅读器变得越来越简单。您只需要实例化一个 UIExtensionsManager 对象，然后将其设置给 FSPDFViewCtrl。

实例化一个 UIExtensionsManager 对象，并且设置给 FSPDFViewCtrl

在 "ViewController.m" 文件中，您只需要添加如下的代码：

```
#import <uiextensionsDynamic/uiextensionsDynamic.h>

UIExtensionsManager* extensionsManager;
...

extensionsManager = [[UIExtensionsManager alloc] initWithPDFViewControl:pdfViewCtrl];
```

```
pdfViewCtrl.extensionsManager = extensionsManager;
```

添加访问摄像头、麦克风和照片库的权限

为了在 iOS 9.0 或者更高版本的设备上访问摄像头、麦克风和照片库，您需要在 "**Info.plist**" 文件中添加如下的配置。

```
<key>NSCameraUsageDescription</key>
<string>The App needs to access your Camera, please allow</string>

<key>NSMicrophoneUsageDescription</key>
<string>The App needs to access your Microphone, please allow</string>

<key>NSPhotoLibraryAddUsageDescription</key>
<string>The App needs to add pictures into your Photo Library, please allow</string>

<key>NSPhotoLibraryUsageDescription</key>
<string>The App needs to access your Photo Library, please allow</string>
```

ViewController.m 的完整更新如下：

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
#import <uiextensionsDynamic/uiextensionsDynamic.h>

@interface ViewController : UIViewController

@end

@implementation ViewController
{
    UIExtensionsManager* extensionsManager;
}

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF.
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];

    // Initialize a FSPDFViewCtrl object with the size of the entire screen.
    FSPDFViewCtrl* pdfViewCtrl;
    pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame:[self.view bounds]];

    // Open an unencrypted PDF document.
    [pdfViewCtrl openDoc:pdfPath password:nil completion:nil];
    // Add the pdfViewCtrl to the root view.
    [self.view addSubview:pdfViewCtrl];

    // Instantiate a UIExtensionsManager object and set it to pdfViewCtrl
    pdfViewCtrl.extensionsManager = extensionsManager;
}
```

```
extensionsManager = [[UIExtensionsManager alloc] initWithPDFViewControl:pdfViewCtrl];
pdfViewCtrl.extensionsManager = extensionsManager;
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

我们在 iPhone 11 模拟器上运行该工程。现在，它是一个功能齐全的 PDF 阅读器，如 Figure 3-10 所示，包含 Complete PDF Viewer demo 中所有的功能。请随意进行体验。

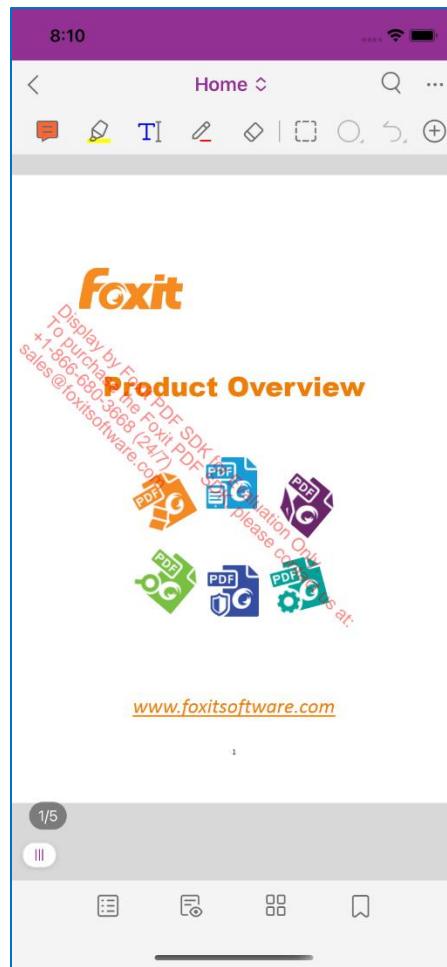


Figure 3-10

3.1.6 基于功能齐全的 PDF 阅读器添加扫描功能

扫描功能是一个单独的模块，没有封装在 UI Extensions 组件中。因此，如果您需要在工程中使用该功能，那么请添加如下的核心代码来调用 scan 模块：

```
#import <FoxitPDFScanUI/PDFScanManager.h>

// Initialize the scan module.
[PDFScanManager initializeScanner:0 serial2:0];
[PDFScanManager initializeCompression:0 serial2:0];

// Get the PDFScan controller.
UIViewController *VC = [[PDFScanManager shareManager] getPDFScanView];
if (VC) [self presentViewController:VC animated:YES completion:nil];

[PDFScanManager setSaveAsCallBack:^(NSError * _Nonnull error, NSString * _Nonnull savePath) {
    // <add your code>
}];
```

对于 **PDFScanManager::initializeScanner** 和 **PDFScanManager::initializeCompression** 接口，如果您将参数设置为 0，则扫描后的图片会带有水印。如果您需要去掉水印，请联系 Foxit 销售或者技术支持团队来获取授权的 key。

基于上一节，添加一个新的 button 来调用 scan 模块。

更新 **ViewController.m**，如下所示：

(假设您已经将 "samples/complete_pdf_viewer/Source/Assets.xcassets" 文件夹下的 "**scan.imageset**" 图片文件拷贝到 "pdfreader/pdfreader/Assets.xcassets" 文件夹下。)

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
#import <uiextensionsDynamic/uiextensionsDynamic.h>
#import <FoxitPDFScanUI/PDFScanManager.h>

@interface ViewController : UIViewController

@end

@implementation ViewController
{
    UIExtensionsManager* extensionsManager;
    UIButton *openScanBtn;
}

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF.
```

```
NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];  
  
// Initialize a FSPDFViewCtrl object with the size of the entire screen.  
FSPDFViewCtrl* pdfViewController;  
pdfViewController = [[FSPDFViewCtrl alloc] initWithFrame:[self.view bounds]];  
  
// Open an unencrypted PDF document.  
[pdfViewController openDoc:pdfPath password:nil completion:nil];  
  
// Add the pdfViewController to the root view.  
[self.view addSubview:pdfViewController];  
  
// Instantiate a UIExtensionsManager object and set it to pdfViewController  
UIExtensionsManager *extensionsManager = [[UIExtensionsManager alloc] initWithPDFViewControl:pdfViewController];  
pdfViewController.extensionsManager = extensionsManager;  
  
// Create a scan button.  
UIButton *openScanBtn = [[UIButton alloc] initWithFrame:CGRectMake(self.view.frame.size.width - 80,  
self.view.frame.size.height - 140, 60, 60)];  
[openScanBtn setImage:[UIImage imageNamed:@"scan"] forState:UIControlStateNormal];  
[openScanBtn addTarget:self action:@selector(openScan:) forControlEvents:UIControlEventTouchUpInside];  
  
// Add the scan button to the root view.  
[self.view addSubview:openScanBtn];  
  
// Initialize the scan module.  
[PDFScanManager initializeScanner:0 serial2:0];  
[PDFScanManager initializeCompression:0 serial2:0];  
  
}  
  
- (IBAction)openScan:(UIButton *)sender{  
    // Get the PDFScan controller.  
    UIViewController *VC = [[PDFScanManager shareManager] getPDFScanView];  
    if (VC) [self presentViewController:VC animated:YES completion:nil];  
  
    [PDFScanManager setSaveAsCallBack:^(NSError *_Nonnull error, NSString *_Nonnull savePath) {  
        if (savePath) {  
            if (VC.presentingViewController) {  
                [VC.presentingViewController dismissViewControllerAnimated:NO completion:nil];  
            }  
            [VC dismissViewControllerAnimated:NO completion:nil];  
        }  
    }];  
}  
  
- (void)didReceiveMemoryWarning {  
    [super didReceiveMemoryWarning];  
    // Dispose of any resources that can be recreated.  
}
```

```
@end
```

在 iPhone 11 模拟器上运行该工程，然后您将看到如 Figure 3-11 所示的界面，点击 scan 按钮可以开始扫描文档。

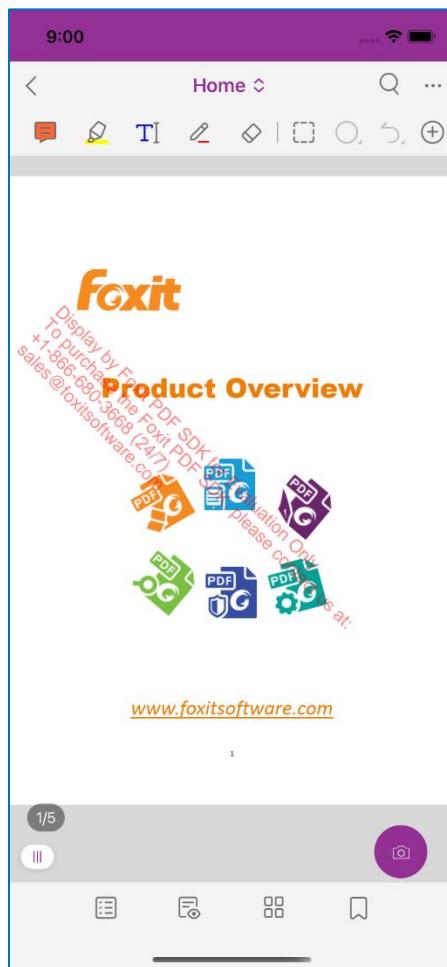


Figure 3-11

3.2 Make an iOS app in Swift with Foxit PDF SDK for iOS

如今，Swift 因其简单清晰、更易于阅读的语法，越来越受到 iOS 开发人员的欢迎。为了更好地支持 Swift 开发人员，本节将帮助您使用 Foxit PDF SDK for iOS 快速构建一个 Swift 语言的 iOS 应用程序。其主要包括以下的步骤：

- 创建一个 Swift 语言的 iOS 工程
- 集成 Foxit PDF SDK for iOS 到您的应用程序
- 初始化 Foxit PDF SDK for iOS

- 使用 `FSPDFViewCtrl` 显示 PDF 文档
- 使用 `UI Extensions` 组件构建一个功能齐全的 PDF 阅读器
- 基于功能齐全的 PDF 阅读器添加扫描功能

3.2.1 创建一个 Swift 语言的 iOS 工程

在本指南中，使用 Xcode 12.0.1 创建一个新的 iOS 工程。

创建一个 Swift 语言的 iOS 工程，请参阅 3.1.1 小节 "[创建一个 Objective-C 语言的 iOS 工程](#)"。唯一不同的是需要选择 Swift 语言，如 Figure 3-12 所示。

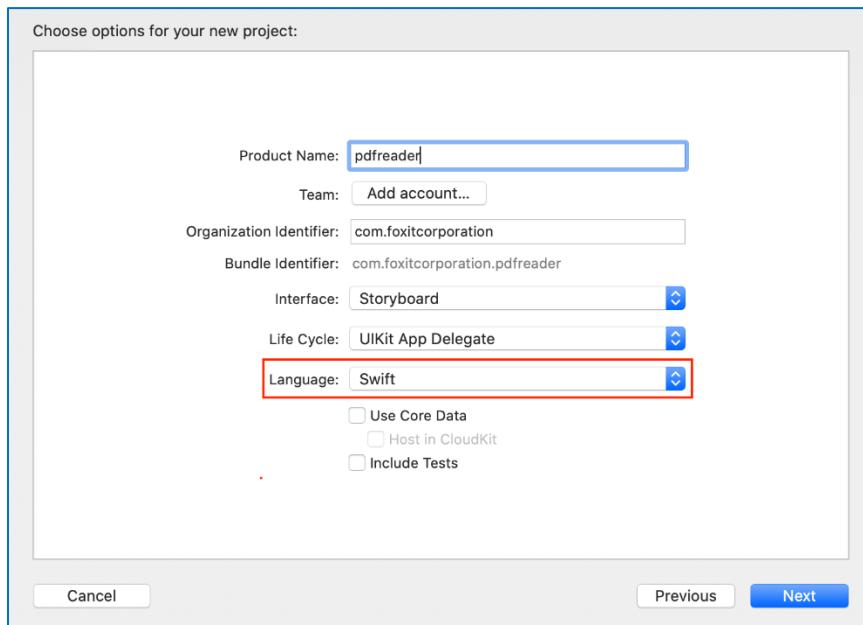


Figure 3-12

3.2.2 集成 Foxit PDF SDK for iOS 到您的应用程序

集成 Foxit PDF SDK for iOS 到您的 Swift 应用程序，请参阅 3.1.2 小节 "[集成 Foxit PDF SDK for iOS 到您的应用程序](#)" 将 "`FoxitRDK.framework`"，"`uiextensionsDynamic.framework`" 和 "`FoxitPDFScanUI.framework`" 添加到 `pdfreader` 工程。

3.2.3 初始化 Foxit PDF SDK for iOS

在调用任何 API 之前，应用程序必须使用 `license` 初始化 Foxit PDF SDK for iOS。`FSLibrary::initialize` 函数用于 SDK 库的初始化。试用 `license` 文件在下载包的"libs"文件夹下。当试用期结束后，您需要购买正式 `license` 以继续使用该 SDK。在 `AppDelegate.swift` 文件的 `application` 方法中完成 SDK 的初始化。

```

import FoxitRDK
...
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    let sn = ""
    let key = ""
    let eRet = FSLibrary.initialize(sn, key:key)
    if .errSuccess != eRet {
        return false
    }
    return true
}

```

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

3.2.4 使用 FSPDFViewCtrl 显示 PDF 文档

到目前为止，我们已经在 *pdfreader* 工程中添加了 Foxit PDF SDK for iOS frameworks，并且完成了 SDK 库的初始化。现在，我们将使用 FSPDFViewCtrl 通过几行代码来显示一个 PDF 文档。

备注：如果只需要显示一个 PDF 文档，则不需要 *UI Extensions* 组件。

首先，在工程中添加一个 PDF 文档作为测试文档。例如，使用下载解压包中 "samples\test_files" 文件夹下的 "Sample.pdf" 文档。右击 *pdfreader* 工程，选择 **Add Files to "pdfreader"**...添加该文档。当添加完成后，您可以在 Xcode 的 **Copy Bundle Resources** 看到该文档，如 Figure 3-13 所示。

备注：您可以直接添加 PDF 到 **Copy Bundle Resources**。单击 *pdfreader* 工程，在 **Build Phases** 选项卡下找到 **Copy Bundle Resources**，点击 + 按钮，选择需要添加的文档。您可以使用任何 PDF 文档，只需要将其添加到 Xcode 的 **Copy Bundle Resources**。

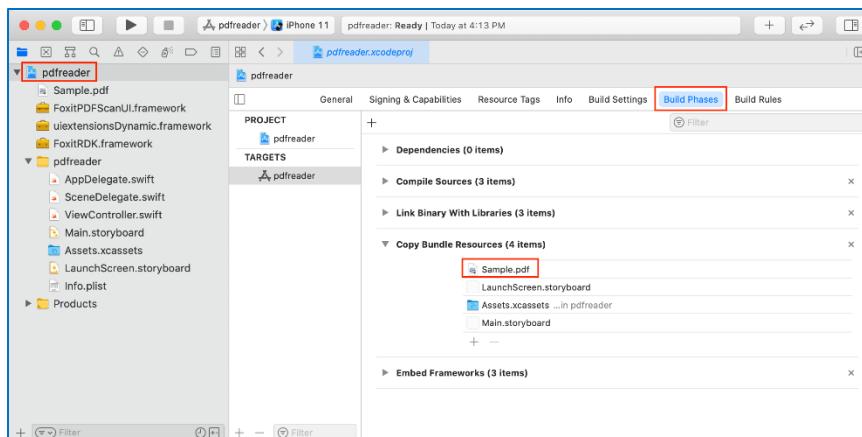


Figure 3-13

其次，在 ViewController.swift 中加入如下的代码来打开并显示一个 PDF 文档。显示一个 PDF 文档很简单，您只需要获取一个 PDF 文档路径，实例化一个 FSPDFViewCtrl 对象，然后调用 FSPDFViewCtrl::openDoc 函数打开和渲染该 PDF 文档。

更新 ViewController.swift，如下所示：

```
import UIKit
import FoxitRDK

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Get the path of a PDF.
        let pdfPath = Bundle.main.path(forResource: "Sample", ofType: "pdf")!

        // Initialize a FSPDFViewCtrl object with the size of the entire screen.
        var pdfViewCtrl: FSPDFViewCtrl!
        pdfViewCtrl = FSPDFViewCtrl.init(frame:self.view.bounds)

        // Set the document to display.
        pdfViewCtrl.openDoc(pdfPath, password: nil, completion: nil)

        // Add the pdfViewCtrl to the root view.
        self.view.insertSubview(pdfViewCtrl, at: 0)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

太棒了！我们已经使用 Foxit PDF SDK for iOS 通过几行代码完成了一个显示 PDF 文档的简单的 iOS 应用程序 (Swift 语言)。下一步是在真机设备或者模拟器上运行该工程。

在本指南中，将使用 iPhone 11 模拟器来编译和运行该工程。当成功编译后，您将看到"Sample.pdf" 文档显示如 Figure 3-14 所示。该示例应用程序具有一些基本的 PDF 功能，比如放大/缩小和翻页。您可以进行体验。

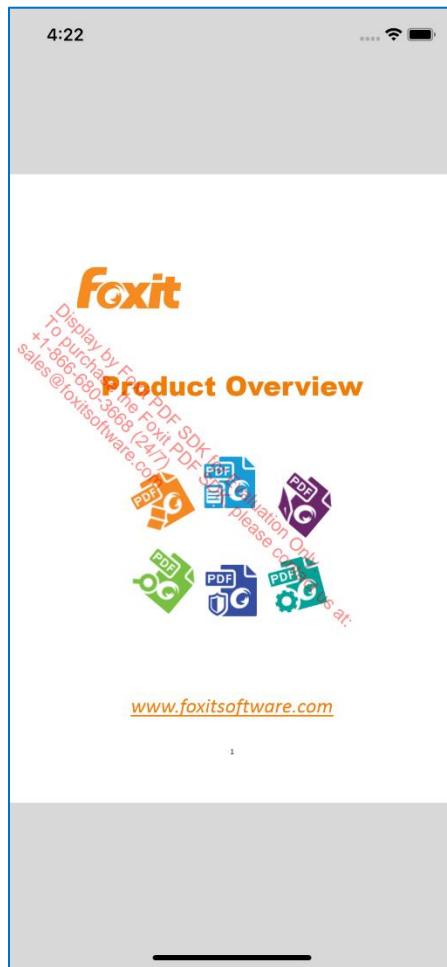


Figure 3-14

3.2.5 使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器

Foxit PDF SDK for iOS 带有内置的 UI 设计，包括应用程序的基础 UI 和功能模块 UI。内置 UI 通过 Foxit PDF SDK for iOS 实现，并且封装在 UI Extensions 组件中。因此，构建一个功能齐全的 PDF 阅读器变得越来越简单。您只需要实例化一个 `UIExtensionsManager` 对象，然后将其设置给 `FSPDFViewCtrl`。

实例化一个 `UIExtensionsManager` 对象，并且设置给 `FSPDFViewCtrl`

在 "ViewController.swift" 文件中，您只需要添加如下的代码：

```
import uiextensionsDynamic  
...  
var extensionsManager: UIExtensionsManager!  
...
```

```
extensionsManager = UIExtensionsManager(pdfViewControl: pdfViewCtrl)
pdfViewCtrl.extensionsManager = extensionsManager;
```

添加访问摄像头、麦克风和照片库的权限

为了在 iOS 9.0 或者更高版本的设备上访问摄像头、麦克风和照片库，您需要在 "**Info.plist**" 文件中添加如下的配置。

```
<key>NSCameraUsageDescription</key>
<string>For adding photographs to your PDF files.</string>

<key>NSMicrophoneUsageDescription</key>
<string>RDK need to add record permissions,please allow</string>

<key>NSPhotoLibraryAddUsageDescription</key>
<string>RDK need to add picture permissions,please allow</string>

<key>NSPhotoLibraryUsageDescription</key>
<string>For adding pictures to your PDF files.</string>
```

ViewController.swift 的完整更新如下：

```
import UIKit
import FoxitRDK
import uiextensionsDynamic

class ViewController: UIViewController {

    var extensionsManager: UIExtensionsManager!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Get the path of a PDF.
        let pdfPath = Bundle.main.path(forResource: "Sample", ofType: "pdf")!

        // Initialize a FSPDFViewCtrl object with the size of the entire screen.
        var pdfViewCtrl: FSPDFViewCtrl!
        pdfViewCtrl = FSPDFViewCtrl.init(frame:self.view.bounds)

        // Set the document to display.
        pdfViewCtrl.openDoc(pdfPath, password: nil, completion: nil)

        // Add the pdfViewCtrl to the root view.
        self.view.insertSubview(pdfViewCtrl, at: 0)

        // Initialize a UIExtensionsManager object and set it to pdfViewCtrl.
        extensionsManager = UIExtensionsManager(pdfViewControl: pdfViewCtrl)
        pdfViewCtrl.extensionsManager = extensionsManager;
    }
}
```

```
override func didReceiveMemoryWarning() {  
    super.didReceiveMemoryWarning()  
    // Dispose of any resources that can be recreated.  
}  
}
```

我们在 iPhone 11 模拟器上运行该工程。现在，它是一个功能齐全的 PDF 阅读器，如 Figure 3-15 所示，包含 Complete PDF Viewer demo 中所有的功能。请随意进行体验。

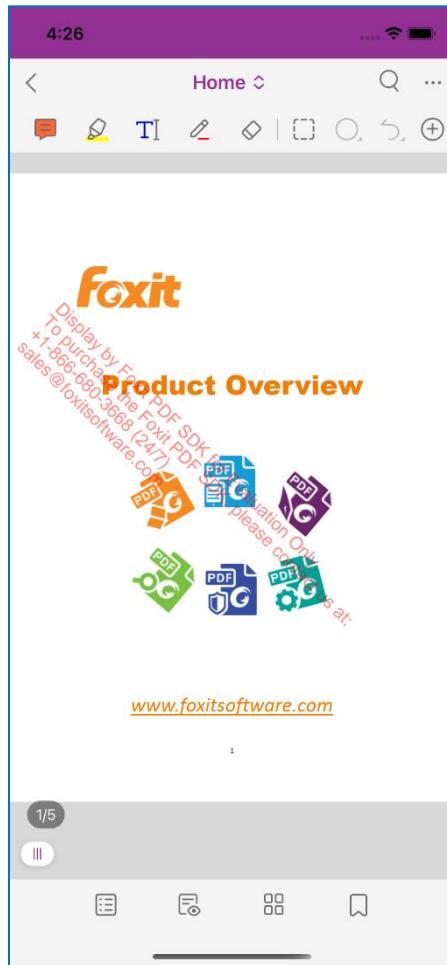


Figure 3-15

3.2.6 基于功能齐全的 PDF 阅读器添加扫描功能

扫描功能是一个单独的模块，没有封装在 UI Extensions 组件中。因此，如果您需要在工程中使用该功能，那么请添加如下的核心代码来调用 scan 模块：

```
import FoxitPDFScanUI  
  
// Initialize the scan module.
```

```
PDFScanManager.initializeScanner(0, serial2: 0);
PDFScanManager.initializeCompression(0, serial2: 0);

// Get the PDFScan controller.
let VC = PDFScanManager.share().getPDFScanView();
self.present(VC, animated: true, completion: nil);
PDFScanManager.saveAsCallBack = { (error, savePath) -> () in
    // <add your code>
}
```

对于 **PDFScanManager.initializeScanner** 和 **PDFScanManager.initializeCompression** 接口，如果您将参数设置为 0，则扫描后的图片会带有水印。如果您需要去掉水印，请联系 Foxit 销售或者技术支持团队来获取授权的 key。

基于上一节，添加一个新的 button 来调用 scan 模块。

更新 ViewController.swift，如下所示：

(假设您已经将 "samples/swift/complete_pdf_viewer_swift/Sources/Assets.xcassets/" 文件夹下的 "**scan.imageset**" 图片文件拷贝到 "pdfreader/pdfreader/Assets.xcassets" 文件夹下。)

```
import UIKit
import FoxitRDK
import uiextensionsDynamic
import FoxitPDFScanUI

class ViewController: UIViewController {

    var extensionsManager: UIExtensionsManager!
    var openScanBtn: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Get the path of a PDF.
        let pdfPath = Bundle.main.path(forResource: "Sample", ofType: "pdf")!

        // Initialize a FSPDFViewCtrl object with the size of the entire screen.
        var pdfViewCtrl: FSPDFViewCtrl!
        pdfViewCtrl = FSPDFViewCtrl.init(frame:self.view.bounds)

        // Set the document to display.
        pdfViewCtrl.openDoc(pdfPath, password: nil, completion: nil)

        // Add the pdfViewCtrl to the root view.
        self.view.insertSubview(pdfViewCtrl, at: 0)

        // Initialize a UIExtensionsManager object and set it to pdfViewCtrl.
        extensionsManager = UIExtensionsManager(pdfViewControl: pdfViewCtrl)
        pdfViewCtrl.extensionsManager = extensionsManager;
```

```
// Create a scan button.  
openScanBtn = UIButton(frame: CGRect(x: view.frame.size.width - 80, y: view.frame.size.height - 140, width: 60, height: 60))  
openScanBtn.setImage(UIImage(named: "scan"), for: .normal);  
openScanBtn.addTarget(self, action: #selector(openScan(_:)), for: .touchUpInside)  
  
// Add the scan button to the root view.  
self.view.addSubview(openScanBtn);  
  
// Initialize the scan module.  
PDFScanManager.initializeScanner(0, serial2: 0);  
PDFScanManager.initializeCompression(0, serial2: 0);  
  
}  
  
@IBAction func openScan(_ sender: UIButton) {  
    let VC = PDFScanManager.share().getPDFScanView();  
    self.present(VC, animated: true, completion: nil);  
    PDFScanManager.saveAsCallBack = { (error, savePath) -> () in  
        if ((savePath) != nil) {  
            if ((VC.presentingViewController) != nil) {  
                VC.presentingViewController?.dismiss(animated: false, completion: nil)  
            }  
            VC.dismiss(animated: false, completion: nil)  
        }  
    }  
}  
  
override func didReceiveMemoryWarning() {  
    super.didReceiveMemoryWarning()  
    // Dispose of any resources that can be recreated.  
}  
}
```

在 iPhone 11 模拟器上运行该工程，然后您将看到如 Figure 3-16 所示的界面，点击 scan 按钮可以开始扫描文档。

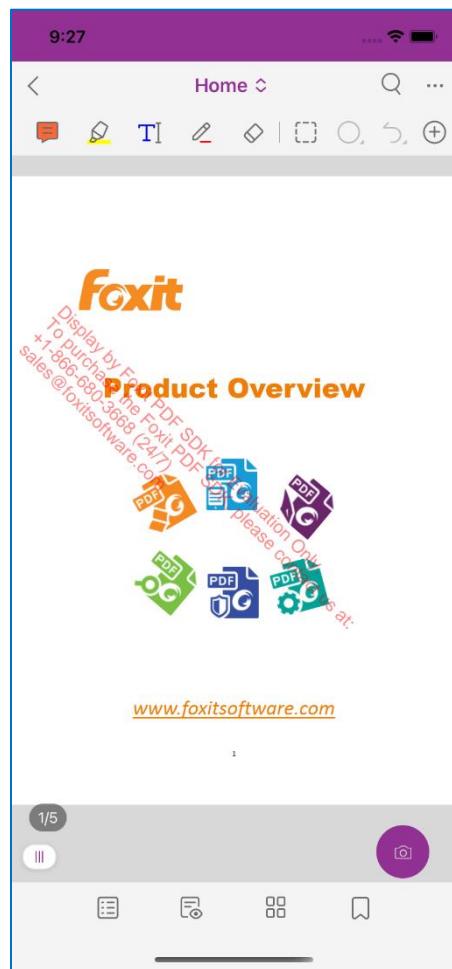


Figure 3-16

4 使用 Mac Catalyst 快速构建一个功能齐全的 PDF 阅读器

上一章节介绍了如何使用 Foxit PDF SDK for iOS 创建一个功能齐全的 PDF 阅读器，该阅读器应用程序只能部署到 iPhone 和 iPad 设备上。本节将使用 Foxit PDF SDK for iOS (Mac Catalyst) 构建一个功能齐全的 PDF 阅读器 (使用 Objective-C 语言)，该阅读器应用程序可以部署到 iPhone、iPad 和 Mac 设备上。其主要包括以下的步骤：

- [使用 Mac Catalyst 创建一个 Mac 应用程序](#)
- [集成 Foxit PDF SDK for iOS \(Catalyst\) 到您的应用程序](#)
- [初始化 Foxit PDF SDK for iOS \(Catalyst\)](#)
- [使用 FSPDFViewCtrl 显示 PDF 文档](#)
- [使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器](#)

4.1.1 使用 Mac Catalyst 创建一个 Mac 应用程序

本节将使用 Mac Catalyst 创建一个 Mac 应用程序，当前 Xcode 版本使用的是 11.5。请参阅 3.1.1 小节 "[创建一个 Objective-C 语言的 iOS 工程](#)" 创建一个 Mac 工程，并命名为 "**pdfreader_catalyst**"。

4.1.2 集成 Foxit PDF SDK for iOS (Catalyst) 到您的应用程序

备注：在本章中，我们将使用默认的内置 UI 实现来开发该应用程序，为了简单和方便(直接使用 UI Extensions 组件，不需要源代码工程)，我们只需要添加以下的文件到 *pdfreader_catalyst* 工程中。

- **FoxitRDK.xcframework** – 该 framework 包含 Foxit PDF SDK for iOS 的动态库和相关头文件。
- **uiextensionsDynamic.xcframework** – 该 framework 包括 UI Extensions 动态库，相关头文件，以及默认内置 UI 实现需要的资源文件。
- **(可选) FoxitPDFScanUI.framework** – 该 framework 包括 Foxit PDF SDK 扫描功能动态库，相关头文件，以及扫描功能默认内置 UI 实现需要的资源文件。

备注：*FoxitPDFScanUI.framework* 仅支持 iOS 平台，目前不支持 macOS 平台。

添加上述的三个动态 framework 文件到 *pdfreader_catalyst* 工程，请按照如下的步骤：

- 添加对 Mac 设备的支持。单击工程，在 **General** 选项卡下找到 **Deployment Info**，勾选 **Mac** 复选框，如 Figure 4-1 所示。(如果您的应用程序只需要支持 iPhone/iPad，则不需要勾选 **Mac** 复选框)

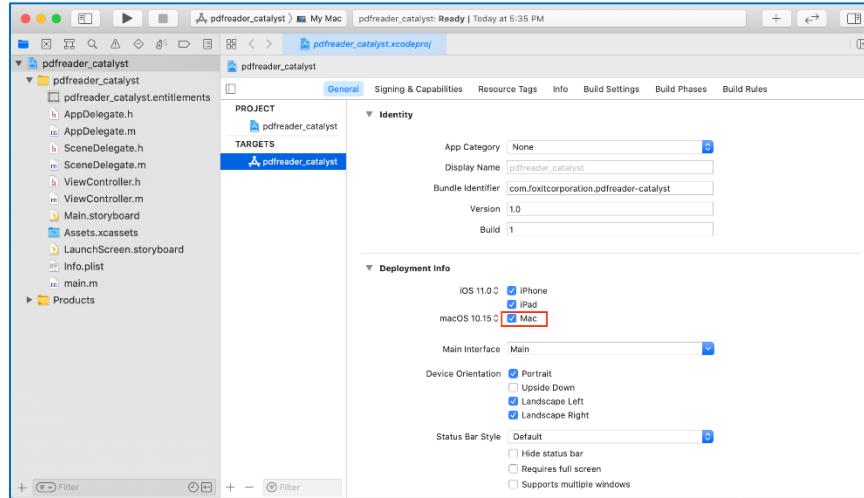


Figure 4-1

- 将发布包 "libs" 目录下的 **FoxitRDK.xcframework**, **uiextensionsDynamic.xcframework** 和 **FoxitPDFScanUI.framework** 拖拽到 *pdfreader_catalyst* 工程中。

备注：当拖拽 framework 时，请确保勾选 "**Copy items if needed**" 选项，如 Figure 4-2 所示。

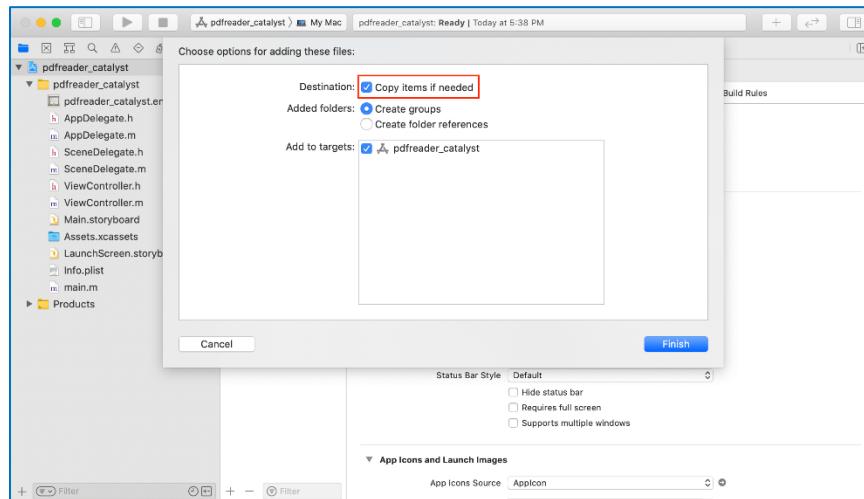


Figure 4-2

- c) 嵌入动态 framework。单击工程，在 **General** 选项卡下找到 **Frameworks, Libraries, and Embedded Content**，然后选择 "Embed & Sign" 以及支持的平台，如 Figure 4-3 所示。

备注：*FoxitPDFScanUI.framework* 只支持 iOS 平台。

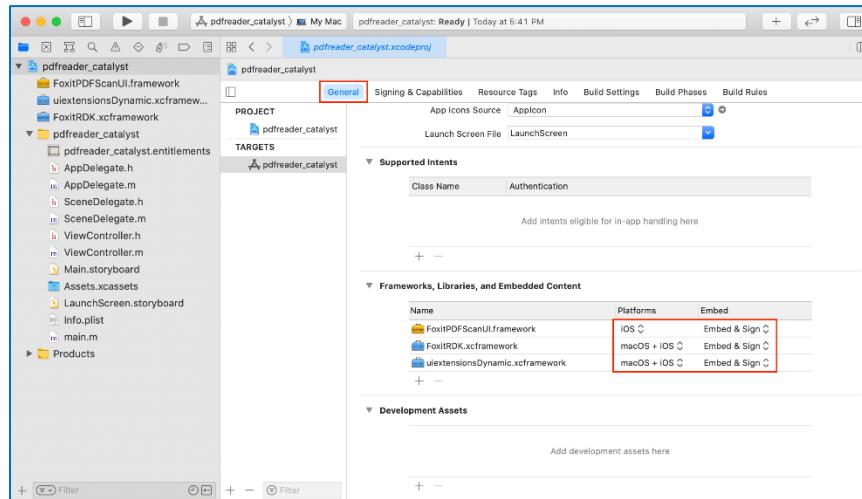


Figure 4-3

- d) 检查 Framework 搜索路径。点击工程，在 **Build Settings** 选项卡下找到 **Search Paths**，检查 **Framework search paths** 是否有设置，如果没有，请正确设置，如 Figure 4-4 所示。

备注：您需要将 \$(PROJECT_DIR) 路径设置为递归搜索。双击如下图中的路径，然后进行设置。

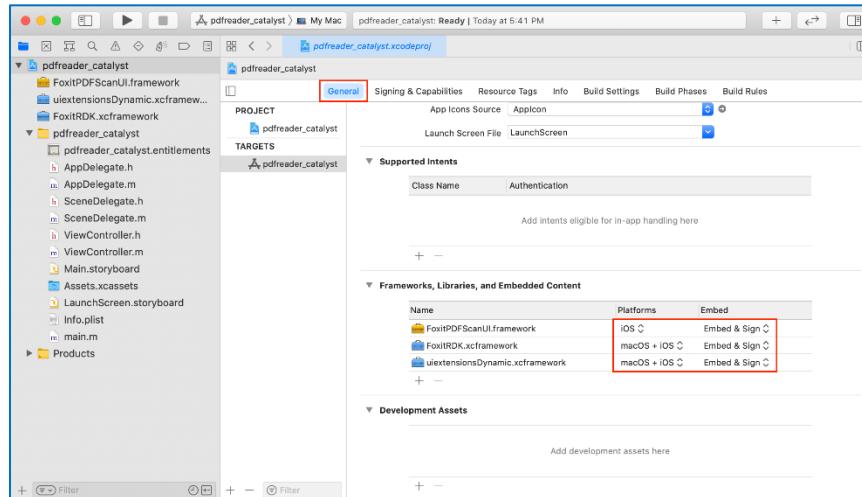


Figure 4-4

到此，我们已成功在 `pdfreader_catalyst` 工程中添加了 "`FoxitRDK.xcframework`"，
"`uiextensionsDynamic.xcframework`" 和 "`FoxitPDFScanUI.framework`"。

4.1.3 初始化 Foxit PDF SDK for iOS (Catalyst)

初始化 Foxit PDF SDK for iOS (Catalyst) 与 3.1.3 小节 "[初始化 Foxit PDF SDK for iOS](#)" 是一样的，请参阅该小节。

4.1.4 使用 FSPDFViewController 显示 PDF 文档

使用 FSPDFViewController 显示一个 PDF 文档与 [3.1.4 小节](#)是一样的。请参阅 3.1.4 小节 "[使用 FSPDFViewController 显示 PDF 文档](#)" 添加一个 PDF 文档到工程中，然后在 `ViewController.m` 中添加以下的代码。

更新 `ViewController.m`，如下所示：

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>

@interface ViewController : UIViewController

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF.
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];

    // Initialize a FSPDFViewController object with the size of the entire screen.
    FSPDFViewController* pdfViewController;
    pdfViewController = [[FSPDFViewController alloc] initWithFrame:[self.view bounds]];

    // Open an unencrypted PDF document.
    [pdfViewController openDoc:pdfPath password:nil completion:nil];

    // Add the pdfViewController to the root view.
    [self.view addSubview:pdfViewController];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

然后，编译和运行工程。该工程可以在 iPhone/iPad, 或者 Mac 上运行。本节中，选择 "My Mac" 来运行该工程。当成功编译后，您将看到 "Sample.pdf" 文档显示如 Figure 4-5 所示。

备注：您可能需要使用开发配置文件(*development provisioning profile*) 注册您的 Mac 电脑，以允许该应用程序可以在 Mac 上启动，以及在开发中使用某些应用服务。



Figure 4-5

4.1.5 使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器

构建一个功能齐全的 PDF 阅读器与 3.1.5 小节 "[使用 UI Extensions 组件构建一个功能齐全的 PDF 阅读器](#)" 是一样的。更新 ViewController.m，如下所示：

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
#import <uiextensionsDynamic/uiextensionsDynamic.h>

@interface ViewController : UIViewController

@end

@implementation ViewController
{
    UIExtensionsManager* extensionsManager;
}

- (void)viewDidLoad {
```

```
[super viewDidLoad];

// Get the path of a PDF.
NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];

// Initialize a FSPDFViewCtrl object with the size of the entire screen.
FSPDFViewCtrl* pdfViewCtrl;
pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame:[self.view bounds]];

// Open an unencrypted PDF document.
[pdfViewCtrl openDoc:pdfPath password:nil completion:nil];

// Add the pdfViewCtrl to the root view.
[self.view addSubview:pdfViewCtrl];

// Instantiate a UIExtensionsManager object and set it to pdfViewCtrl
extensionsManager = [[UIExtensionsManager alloc] initWithPDFViewControl:pdfViewCtrl];
pdfViewCtrl.extensionsManager = extensionsManager;
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

然后，编译工程，并在 Mac 上运行工程。当成功编译后，您将看到 "Sample.pdf" 文档显示如 Figure 4-6 所示。

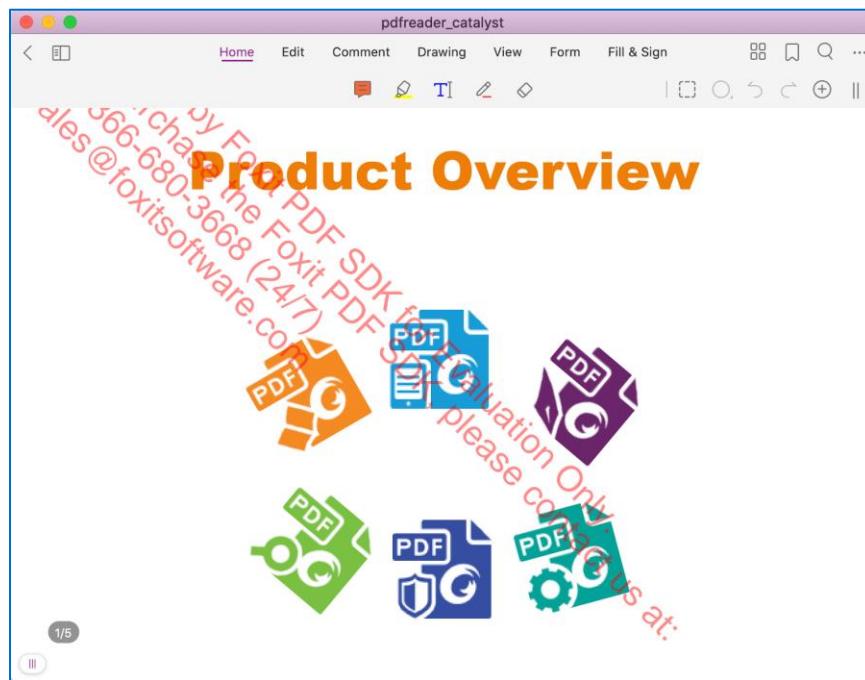


Figure 4-6

5 自定义 UI

Foxit PDF SDK for iOS 为开发人员提供了一个简单、干净和友好的用户界面，可以快速构建一个功能齐全的 PDF 应用程序而不需花费太多的时间在设计上。此外，自定义用户页面也非常简单。Foxit PDF SDK for iOS 提供了 UI Extensions 组件的源代码(包含即用型 UI 模块的实现)，这样开发人员可以根据需要灵活自定义界面的外观。

从 4.0 版本开始，开发人员可以通过一个配置文件对功能进行自定义。

从 5.0 版本开始，内置 UI 中的任何元素都是可配置的，并且为开发人员提供了更高级的 APIs 和更强大的配置文件用来进一步自定义 UI 元素，比如显示/隐藏某个特定的面板、top/bottom toolbar、top toolbar 中的菜单项，以及 View setting bar 和 More Menu view 中的菜单项。

从 6.3 版本开始，配置文件被进一步改进，提供了更多设置选项对 UI 进行自定义，包括权限管理和 UI 元素的属性。

从 8.0 版本开始，UI Extensions Component 的内置 UI 进行了全新的改版。

以下部分将介绍如何通过配置文件、APIs、源代码自定义功能模块、权限管理和 UI 元素。

5.1 通过配置文件自定义 UI

通过配置文件，开发人员可以轻松选择功能模块，设置权限管理和 UI 元素的属性，而无需编写任何额外的代码或者重新设计应用程序的 UI。

5.1.1 JSON 文件介绍

配置文件可以作为 JSON 文件提供，也可以直接在代码中编写。我们建议您使用 JSON 文件格式，因为其可以更直观，更清晰的查看和配置各个选项。

您可以参考 Foxit PDF SDK for iOS 包中"samples\complete_pdf_viewer\Source\Resource\"文件夹下的 JSON 文件。其内容如下所示：

```
{  
  "modules": {  
    "readingbookmark": true,  
    "outline": true,  
    "annotations": {  
      "highlight": true,  
      "underline": true,  
      "squiggly": true,  
    }  
  }  
}
```

```
"strikeout": true,
"insert": true,
"replace": true,
"line": true,
"rectangle": true,
"oval": true,
"arrow": true,
"pencil": true,
"eraser": true,
"typewriter": true,
"textbox": true,
"callout": true,
"note": true,
"stamp": true,
"polygon": true,
"cloud": true,
"polyline": true,
"measure": true,
"image": true,
"audio": true,
"video": true,
"redaction": true
},
"thumbnail": true,
"attachment": true,
"signature": true,
"fillSign": true,
"search": true,
"navigation": true,
"form": true,
"selection": true,
"encryption": true,
"multipleSelection": true
},
"permissions": {
"runJavaScript": true,
"copyText": true,
"disableLink": false
},
"uiSettings": {
"pageMode": "Single",
"continuous": false,
"colorMode": "Normal",
"zoomMode": "FitWidth",
"mapForegroundColor": "#5d5b71",
"mapBackgroundColor": "#00001b",
"reflowBackgroundColor": "#ffffff",
"disableFormNavigationBar": false,
"highlightForm": true,
"highlightLink": true,
"highlightLinkColor": "#16007fff",
```

```
"highlightFormColor": "#200066cc",
"fullscreen" : true,
"annotations": {
    "continuouslyAdd" : true,
    "highlight": {
        "color" : "#ffff00",
        "opacity" : 1.0
    },
    "areaHighlight": {
        "color" : "#ffff00",
        "opacity" : 1.0
    },
    "underline": {
        "color" : "#66cc33",
        "opacity" : 1.0
    },
    "squiggly": {
        "color" : "#993399",
        "opacity" : 1.0
    },
    "strikeout": {
        "color" : "#ff0000",
        "opacity" : 1.0
    },
    "insert": {
        "color" : "#993399",
        "opacity" : 1.0
    },
    "replace": {
        "color" : "#0000ff",
        "opacity" : 1.0
    },
    "line": {
        "color" : "#ff0000",
        "opacity" : 1.0,
        "thickness" : 2
    },
    "rectangle": {
        "color" : "#ff0000",
        "opacity" : 1.0,
        "thickness" : 2
    },
    "oval": {
        "color" : "#ff0000",
        "opacity" : 1.0,
        "thickness" : 2
    },
    "arrow": {
        "color" : "#ff0000",
        "opacity" : 1.0,
        "thickness" : 2
    }
}
```

```
},
"pencil": {
    "color" : "#ff0000",
    "opacity" : 1.0,
    "thickness" : 2
},
"highlighter": {
    "color" : "#ffff00",
    "opacity" : 0.5,
    "thickness" : 12
},
"polygon": {
    "color" : "#ff0000",
    "opacity" : 1.0,
    "thickness" : 2
},
"cloud": {
    "color" : "#ff0000",
    "opacity" : 1.0,
    "thickness" : 2
},
"polyline": {
    "color" : "#ff0000",
    "opacity" : 1.0,
    "thickness" : 2
},
"typewriter": {
    "textColor" : "#0000ff",
    "opacity" : 1.0,
    "textFace" : "Courier",
    "textSize" : 18.0
},
"textbox": {
    "color" : "#ff0000",
    "textColor": "#0000ff",
    "opacity" : 1.0,
    "textFace" : "Courier",
    "textSize" : 18.0
},
"callout": {
    "color": "#ff0000",
    "textColor": "#0000ff",
    "opacity" : 1.0,
    "textFace" : "Courier",
    "textSize" : 18.0
},
"note": {
    "color" : "#ff0000",
    "opacity" : 1.0,
    "icon" : "Comment"
},
```

```
"attachment": {  
    "color" : "#ff0000",  
    "opacity" : 1.0,  
    "icon" : "Pushpin"  
},  
"image": {  
    "rotation" : 0,  
    "opacity" : 1.0  
},  
"measure": {  
    "color" : "#ff0000",  
    "opacity" : 1.0,  
    "thickness" : 2,  
    "scaleFromUnit" : "inch",  
    "scaleToUnit" : "inch",  
    "scaleFromValue" : 1,  
    "scaleToValue" : 1  
},  
"redaction": {  
    "fillColor" : "#000000",  
    "textColor": "#ff0000",  
    "textFace" : "Courier",  
    "textSize" : 12  
}  
},  
"form": {  
    "textField": {  
        "textColor": "#000000",  
        "textFace": "Courier",  
        "textSize": 0  
    },  
    "checkBox": {  
        "textColor": "#000000"  
    },  
    "radioButton": {  
        "textColor": "#000000"  
    },  
    "comboBox": {  
        "textColor": "#000000",  
        "textFace": "Courier",  
        "textSize": 0,  
        "customText": false  
    },  
    "listBox": {  
        "textColor": "#000000",  
        "textFace": "Courier",  
        "textSize": 0,  
        "multipleSelection": false  
    }  
},  
"signature": {
```

```
        "color" : "#000000",
        "thickness" : 4
    }
}
```

备注:

- 上述 JSON 文件中的值是配置项的默认值。如果某些配置项不在 JSON 文件中，则将使用其默认值。例如，如果您注释掉“highlight”: true，但高亮功能仍然是可用的。
- 只有附件(attachment)注释是不受“annotations”的子项控制的。点击顶部工具栏 Home 旁边的图标，选择 Comment，可以看到附件注释，如 Figure 5-1 所示。

“attachment”: true，“控制了 attachments 面板和 attachment 注释。如果您将其设置为“false”，则两者都将被禁用。如果您需要隐藏 Comment 中的所有工具，那么您需要将“annotations”和“attachment”同时设置为“false”。

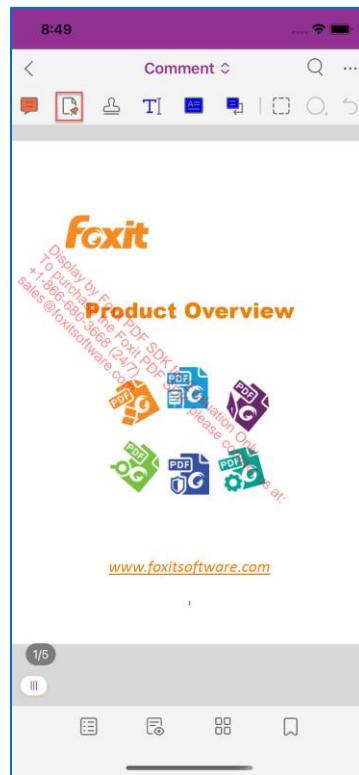


Figure 5-1

5.1.2 配置项描述

JSON 配置文件包括三个部分：功能模块，权限管理和 UI 设置（例如，UI 元素属性）。本节将详细介绍这些配置项。

配置功能模块

备注：功能模块项的值类型是 **bool**，其中 "true" 表示启用该功能模块，"false" 表示将禁用该功能模块。默认值为 "true"。

功能模块	描述
readingbookmark	用户定义书签
outline	PDF 文档书签
annotations (highlight, underline, squiggly, strikeout, insert, replace, line, rectangle, oval, arrow, pencil, eraser, typewriter, textbox, callout, note, stamp, polygon, cloud, polyline, measure, image, audio, video, redaction)	注释模块集合
thumbnail	PDF 页面缩略图显示和页面管理
attachment	PDF 文档附件和附件注释
signature	电子签名和手写签名
fillSign	用文本和符号填写扁平化表单（即非交互式表单）
search	文本搜索
navigation	PDF 页面导航
form	表单填写和表单数据导入导出
selection	文本选择
encryption	PDF 加密
multipleSelection	选择多个 annotations

配置权限管理

备注：配置项的值类型为 **bool**，其中 "true" 表示将启用该权限，"false" 表示将禁用该权限。

runJavaScript 和 **copyText** 的默认值为 "true"，**disableLink** 的默认值为 "false"。

权限管理	描述
runJavaScript	是否允许执行 JavaScript
copyText	是否允许复制文本
disableLink	是否禁用超链接

配置 UI 项及其属性

UI 配置子项	描述/属性	值类型	可选值	默认值	备注
displayMode	页面显示模式	String	Single/ Facing/ CoverLeft/ CoverMiddle/ CoverRight/ Reflow	Single	动态 XFA 文件不支持 Reflow 模式。

UI 配置子项	描述/属性	值类型	可选值	默认值	备注	
continuous	是否连续的显示单页页面	Bool	true/false	false	True 表示连续显示, false 表示不连续显示。该配置项在 "Reflow" 模式下无效。	
zoomMode	页面缩放模式	String	FitWidth/FitPage	FitWidth		
colorMode	页面颜色显示模式	String	Normal/Night/Map	Normal	"Night" 是一种特殊的"Map"模式。	
mapForegroundColor	页面显示的前景颜色	RGB	---	#5d5b71	只有在"colorMode"设置为"Map"时, 该配置项才有效。	
mapBackgroundColor	页面显示的背景颜色	RGB	---	#00001b	只有在"colorMode"设置为"Map"时, 该配置项才有效。	
reflowBackgroundColor	Reflow (重排) 页面的背景	RGB	---	#ffffff		
disableFormNavigationBar	是否禁用表单的辅助导航栏	Bool	true/false	false		
highlightForm	是否高亮表单域	Bool	true/false	true		
highlightFormColor	表单高亮颜色	ARGB		#200066cc	包括 alpha 通道, 并且对动态 xfa 文件无效。	
highlightLink	是否高亮超链接	Bool	true/false	true		
highlightLinkColor	超链接高亮颜色	ARGB		#16007fff	包括 alpha 通道。	
fullscreen	是否全屏显示	Bool	true/false	true	当"fullscreen" 设置为"true"时, 文档将以全屏方式显示。如果用户点击页面, 工具栏将会出现。如果 5 秒内无任何动作, 工具栏和其他辅助工具按钮将自动隐藏。	
annotations	continuous/ yAdd		Bool	true/false	true	是否连续添加某个注释
	highlight	color	RGB		#fffff00	
		opacity	numeric	[0.0-1.0]	1.0	
	areaHighlight	color	RGB		#fffff00	
		opacity	numeric	[0.0-1.0]	1.0	
	underline	color	RGB		#66cc33	
		opacity	numeric	[0.0-1.0]	1.0	
	squiggly	color	RGB		#993399	
		opacity	numeric	[0.0-1.0]	1.0	
	strikeout	color	RGB		#ff0000	

UI 配置子项	描述/属性	值类型	可选值	默认值	备注
	opacity	numeric	[0.0-1.0]	1.0	
insert	color	RGB		#993399	
	opacity	numeric	[0.0-1.0]	1.0	
replace	color	RGB		#0000ff	
	opacity	numeric	[0.0-1.0]	1.0	
line	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
rectangle	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
oval	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
arrow	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
pencil	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
highlighter	color	RGB		#ffff00	
	opacity	numeric	[0.0-1.0]	0.5	
	thickness	numeric	[1-12]	12	
polygon	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
cloud	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
polyline	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
typewriter	textColor	RGB		#0000ff	
	opacity	numeric	[0.0-1.0]	1.0	
	textFace	String	Courier/ Courier-Bold/ Courier- BoldOblique/ Courier- Oblique/ Helvetica/ Helvetica- Bold/ Helvetica- BoldOblique/ Helvetica- Oblique/	Courier	文本字体名称。 如果设置为非法值， 则使用默认字体。

UI 配置子项	描述/属性	值类型	可选值	默认值	备注
textbox			Times-Roman/ Times-Bold/ Times-Italic/ Times-BoldItalic		
	textSize	Integer	>=1	18	
	color	RGB		#ff0000	
	textColor	RGB		#0000ff	
	opacity	numeric	[0.0-1.0]	1.0	
	textFace	String	Courier/ Courier-Bold/ Courier-BoldOblique/ Courier-Oblique/ Helvetica/ Helvetica-Bold/ Helvetica-BoldOblique/ Helvetica-Oblique/ Times-Roman/ Times-Bold/ Times-Italic/ Times-BoldItalic	Courier	文本字体名称。 如果设置为非法值，则使用默认字体。
	textSize	Integer	>=1	18	
	callout	color	RGB	#ff0000	
		textColor	RGB	#0000ff	
		opacity	numeric	[0.0-1.0]	1.0
		textFace	String	Courier/ Courier-Bold/ Courier-BoldOblique/ Courier-Oblique/ Helvetica/ Helvetica-Bold/ Helvetica-BoldOblique/ Helvetica-Oblique/ Times-Roman/	Courier

UI 配置子项	描述/属性	值类型	可选值	默认值	备注
			Times-Bold/ Times-Italic/ Times-BoldItalic		
note	textSize	Integer	>=1	18	
	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	icon	String	Comment/ Key/ Note/ Help/ NewParagraph/ Paragraph/ Insert	Comment	如果设置为非法值，则使用默认值。
attachment	color	RGB		#ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	icon	String	Graph/ PushPin/ Paperclip/ Tag	PushPin	
image	rotation	numeric	0/90/180/270	0	如果设置为非法值，则使用默认值。
	opacity	numeric	[0.0-1.0]	1.0	
measure	color	RGB		ff0000	
	opacity	numeric	[0.0-1.0]	1.0	
	thickness	numeric	[1-12]	2	
	scaleFromUnit	String	pt/m/cm/mm/ /inch/p/ft/yd	inch	缩放的基准单位。 如果设置为非法值，则使用默认值。
	scaleToUnit	String	pt/m/cm/mm/ /inch/p/ft/yd	inch	缩放的目标单位。 如果设置为非法值，则使用默认值。
	scaleFromValue	numeric		1	缩放的基准数值
	scaleToValue	numeric		1	缩放的目标数值
redaction	fillColor	RGB		#000000	
	textColor	RGB		#ff0000	
	textFace	String	Courier/ Helvetica/ Times	Courier	文本字体名称。 如果设置为非法值，则使用默认字体。
	textSize	Integer	>=1	12	
form	textField	textColor	RGB	#000000	
		textFace	String	Courier/ Helvetica/ Times	文本字体名称。 如果设置为非法值，则使用默认字体。

UI 配置子项		描述/属性	值类型	可选值	默认值	备注	
	textSize	Integer	>=0	0	0 表示自动调整字体大小。		
	checkBox	textColor	RGB	#000000			
	radioButton	textColor	RGB	#000000			
	comboBox	textColor	RGB	#000000			
		textFace	String	Courier/ Helvetica/ Times	Courier	文本字体名称。 如果设置为非法值，则使用默认字体。	
		textSize	Integer	>=0	0	0 表示自动调整字体大小。	
		customText			false	True 表示允许自定义文本。 False 表示不允许自定义文本。	
	listBox	textColor	RGB	#000000			
		textFace	String	Courier/ Helvetica/ Times	Courier	文本字体名称。 如果设置为非法值，则使用默认字体。	
		textSize	Integer	>=0	0	0 表示自动调整字体大小。	
		multipleSelection			false	True 表示支持多选。 False 表示不支持多选。	
signature		color	RGB	#000000			
		thickness	numeric	[1-12]	4		

5.1.3 使用配置文件实例化一个 **UIExtensionsManager** 对象

在 [section 3.1.5](#) (Objective-C) 和 [section 3.2.5](#) (Swift) 小节中，我们已经介绍了如何实例化 **UIExtensionsManager**，而且使用这种方式，所有的内置 UI 框架将会被默认加载。在本节中，我们将提供另外一种使用配置文件来实例化一个 **UIExtensionsManager**，以便开发人员可以根据需要轻松自定义 UI。

请参阅以下代码使用配置文件实例化 **UIExtensionsManager** 对象。

备注：您需要准备一个 JSON 配置文件，然后将其添加到工程中。在这里，我们假设您已经添加了一个名为 "uiextensions_config.json" 的 JSON 文件。

在 **ViewController.m** 中添加如下的代码: ([Objective-C](#))

```
UIExtensionsManager* extensionsManager;
...
// Instantiate a FSPDFViewCtrl object with the size of the entire screen.
```

```
FSPDFViewCtrl* pdfViewCtrl;
pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame: [self.view bounds]];

// Get the path of the JSON configuration file.
NSString* configPath = [[NSBundle mainBundle] pathForResource:@"uiextensions_config" ofType:@"json"];

// Initialize a UIExtensionsManager object and set it to pdfViewCtrl.
extensionsManager = [[UIExtensionsManager alloc] initWithPDFViewControl:pdfViewCtrl configuration:[NSData
dataWithContentsOfFile:configPath]];
if (nil == extensionsManager) {
    return;
}
pdfViewCtrl.extensionsManager = extensionsManager;
```

在 **ViewController.swift** 中添加如下的代码: (**Swift**)

```
var extensionsManager: UIExtensionsManager!
...

// Initialize a FSPDFViewCtrl object with the size of the entire screen
var pdfViewCtrl: FSPDFViewCtrl!
pdfViewCtrl = FSPDFViewCtrl.init(frame:self.view.bounds)

// Get the path of the JSON configuration file.
let configPath = Bundle.main.path(forResource: "uiextensions_config", ofType: "json")
var data: Data?
if nil != configPath {
    data = NSData(contentsOfFile: configPath!) as Data?
}

// Initialize a UIExtensionsManager object and set it to pdfViewCtrl.
extensionsManager = UIExtensionsManager.init(pdfViewControl: pdfViewCtrl, configuration: data)
if nil == extensionsMgr {
    return
}
pdfViewCtrl.extensionsManager = extensionsManager;
```

备注: 在上述代码中, 我们使用一个配置文件来实例化 **UIExtensionsManager**。如果您不想使用配置文件, 可参考 [section 3.1.5 \(Objective-C\)](#) 和 [section 3.2.5 \(Swift\)](#) 小节。

5.1.4 通过配置文件自定义 UI 的示例

在本节中, 我们将向您展示如何在您的项目中自定义功能模块、权限管理和 UI 设置(例如, UI 元素属性)。您会发现这些自定义都非常容易的实现, 您只需要修改配置文件。下面列出了一些操作示例。

Note: For your convenience, we will try it in the "**complete_pdf_viewer**" (Objective-C) and "**complete_pdf_viewer_swift**" (Swift) demos found in the "samples" folder.

备注: 为了方便起见, 我们将在 "samples" 文件夹下的 "**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中进行演示。

在 Xcode 中打开该 demos。在 "complete_pdf_viewer\Resource" 或者 "complete_pdf_viewer_swift\Resource" 文件夹下找到配置文件 "uiextensions_config.json"。

示例 1: 禁用"readingbookmark" 和 "navigation" 功能模块。

在 JSON 文件中, 将 "readingbookmark" 和 "navigation" 的值设置为 "false", 如下所示:

```
"readingbookmark": true,
"navigation": true,
```

然后, 重新编译和运行该 demo。如下列出了前后对比图:

修改前:



修改后:



"readingbookmark" 和 "navigation" 功能模块被移除了。

示例 2: 禁用超链接。

在 JSON 文件中, 将 "disableLink" 的值设置为 "true", 如下所示:

```
"permissions": {
  "runJavaScript": true,
  "copyText": true,
  "disableLink": true
},
```

然后, 重新编译和运行该 demo, 您会发现当您单击超链接时没有任何响应。

示例 3：将高亮颜色从黄色设置为红色。

在 JSON 文件中，将 "highlight" 的 color 属性设置为 "#ff0000"，如下所示：

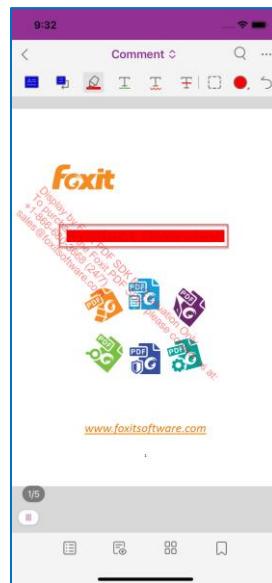
```
"highlight": {  
    "color": "#ff0000",  
    "opacity": 1.0  
},
```

然后，重新编译和运行该 demo。如下列出了前后对比图：

修改前：



修改后：



高亮颜色变为红色了。

5.2 通过 APIs 自定义 UI 元素

在 4.0 版本中，Foxit PDF SDK for Android 支持自定义显示或者隐藏整个 top toolbar 或者 bottom toolbar，从 5.0 版本开始，提供了 APIs 去自定义显示或者隐藏一个特定的面板，top/bottom toolbar、View setting bar 和 More Menu view 上面的菜单项，方便开发人员在内置 UI 框架下对 UI 元素进行修改。

从 8.0 版本开始，UI Extensions Component 的内置 UI 进行了全新的改版。

备注：为了方便起见，我们将在 "samples" 文件夹下的 "**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中向您展示如何通过 APIs 对 UI 元素进行自定义。我们假设您没有修改过 demos 中的 "uiextensions_config.json" 文件，也就是说 UI Extensions 组件中的所有内置 UI 都是启用的。

5.2.1 自定义 top/bottom toolbar

对于 top/bottom toolbar (如 Figure 5-2 所示), 您可以执行以下操作:

1. 显示或者隐藏 top/bottom toolbar。
2. 显示或者隐藏 top/bottom toolbar 上某个特定的菜单项。
3. 移除在 top toolbar 中心位置的特定 tab。

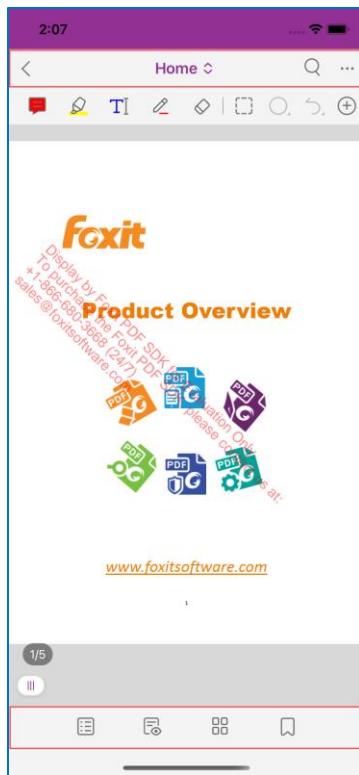


Figure 5-2

Table 5-1 列出了用于自定义 top/bottom toolbar 相关的 APIs。

Table 5-1

<code>(void)enableTopToolbar:(BOOL)isEnabled</code>	启用或者禁用 top toolbar。
<code>(void)enableBottomToolbar:(BOOL)isEnabled</code>	启用或者禁用 bottom toolbar。
<code>(void)setToolbarItemHiddenWithTag:(FS_TOOLBAR_ITEM_TAG)itemTag hidden:(BOOL)isHidden</code>	显示或者隐藏 top toolbar 上的菜单项。

`setToolbarItemHiddenWithTag` 接口中的 "itemTag" 参数值可以设置如下表所示, 其分别对应 top/bottom toolbar 上面的功能菜单。

Item name	itemTag
Back	FS_TOOLBAR_ITEM_TAG_BACK
Search	FS_TOOLBAR_ITEM_TAG_SEARCH
More	FS_TOOLBAR_ITEM_TAG_MORE
Home	FS_TOOLBAR_ITEM_TAG_HOME
Edit	FS_TOOLBAR_ITEM_TAG_EDIT
Comment	FS_TOOLBAR_ITEM_TAG_COMMENT
Drawing	FS_TOOLBAR_ITEM_TAG_DRAWING
Form	FS_TOOLBAR_ITEM_TAG_FORM
Fill & Sign	FS_TOOLBAR_ITEM_TAG_SIGN
Panel	FS_TOOLBAR_ITEM_TAG_PANEL
View	FS_TOOLBAR_ITEM_TAG_VIEW (for tablets) FS_TOOLBAR_ITEM_TAG_VIEW_SETTINGS (for phones)
Thumbnail	FS_TOOLBAR_ITEM_TAG_THUMBNAIL
Bookmark	FS_TOOLBAR_ITEM_TAG_READING_BOOKMARK

在下面的示例中，我们将在"samples"文件夹下的"**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中向您展示如何通过 APIs 对 top/bottom bar 进行自定义。

在 Xcode 中打开该 demos。将示例代码加入到 "**ViewController.m**" (Objective-C) 或者 "**TabsViewController.swift**" (Swift) 中 (加在 UIExtensionsManager 初始化之后)。

备注：手机和平板上面的内置 UI 有一些不同。下面列举的示例大部分适用于手机和平板，只有一个示例仅适用于手机端。在本手册中，如果在手机和平板上的自定义结果类似，则只列出在手机上的效果图。

示例 1：隐藏整个 top toolbar. (适用于手机和平板)

Objective-C:

```
[self.extensionsMgr enableTopToolbar:false];
```

Swift:

```
extensionsManager.enableTopToolbar(false)
```

修改前:



修改后:



示例 2: 隐藏整个 bottom toolbar. (仅适用于手机)

Objective-C:

```
[self.extensionsMgr enableBottomToolbar:false];
```

Swift:

```
extensionsManager.enableBottomToolbar(false)
```

修改前:



修改后:



示例 3: 隐藏 top toolbar 中的 more menu 菜单. (适用于手机和平板)

Objective-C:

```
[self.extensionsMgr setToolbarItemHiddenWithTag:FS_TOOLBAR_ITEM_TAG_MORE hidden:YES];
```

Swift:

```
extensionsManager.setToolbarItemHiddenWithTag(FS_TOOLBAR_ITEM_TAG_MORE, hidden: true);
```

修改前:



修改后:



示例 4：隐藏 top toolbar 中间位置列表中的 "From" tab。(适用于手机和平板)

Objective-C:

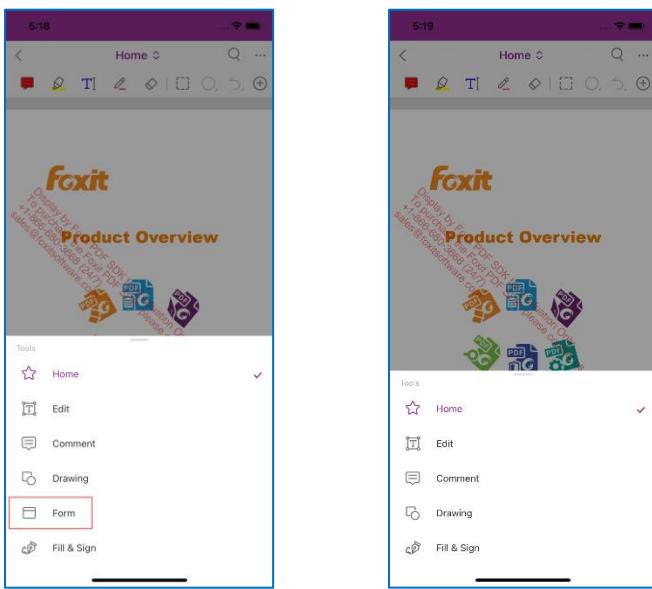
```
[self.extensionsMgr setToolbarItemHiddenWithTag:FS_TOOLBAR_ITEM_TAG_FORM hidden:YES];
```

Swift:

```
extensionsManager.setToolbarItemHiddenWithTag(FS_TOOLBAR_ITEM_TAG_FORM, hidden: true)
```

修改前:

修改后:



示例 5: 隐藏手机端在 **bottom toolbar** 上的 "view" 菜单，或者隐藏平板端在 **top toolbar** 中间位置列表中的 "view" tab。(适用于手机和平板)

Objective-C:

```
if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPad) {
    [self.extensionsMgr setToolbarItemHiddenWithTag:FS_TOOLBAR_ITEM_TAG_VIEW hidden:YES];
}
else {
    [self.extensionsMgr setToolbarItemHiddenWithTag:FS_TOOLBAR_ITEM_TAG_VIEW_SETTINGS hidden:YES];
}
```

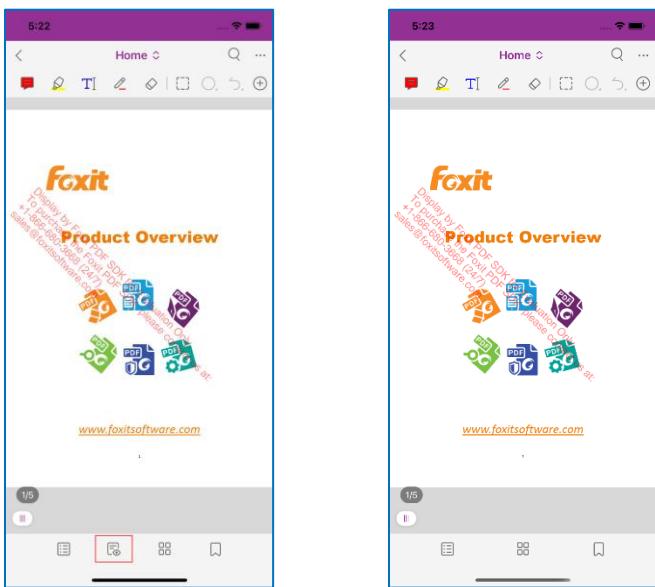
Swift:

```
if UIDevice.current.userInterfaceIdiom == .pad {
    extensionsManager.setToolbarItemHiddenWithTag(FS_TOOLBAR_ITEM_TAG_VIEW, hidden: true)
}
else {
    extensionsManager.setToolbarItemHiddenWithTag(FS_TOOLBAR_ITEM_TAG_VIEW_SETTINGS, hidden: true)
}
```

对于手机设备：

修改前:

修改后:



对于平板设备：

修改前:



修改后:



对于 top/bottom toolbar 中的其他菜单项，您可以参考上面的示例，只需要修改 `setToolbarItemHiddenWithTag` 接口中的 "itemTag" 参数的值。

5.2.2 自定义隐藏一个特定的面板

隐藏一个特定的面板 (见 Figure 5-3，包括了"Bookmarks"、"Outline"、"Annotations"、"Attachments" 和 "Digital Signatures" 面板，手机端点击底部工具栏上的 图标查看，平板端点击左上工具栏上的 图标来查看，)，您只需要使用下面的 API：

```
(void)setPanelHidden:(BOOL)isHidden type:(FSPanelType)type
```

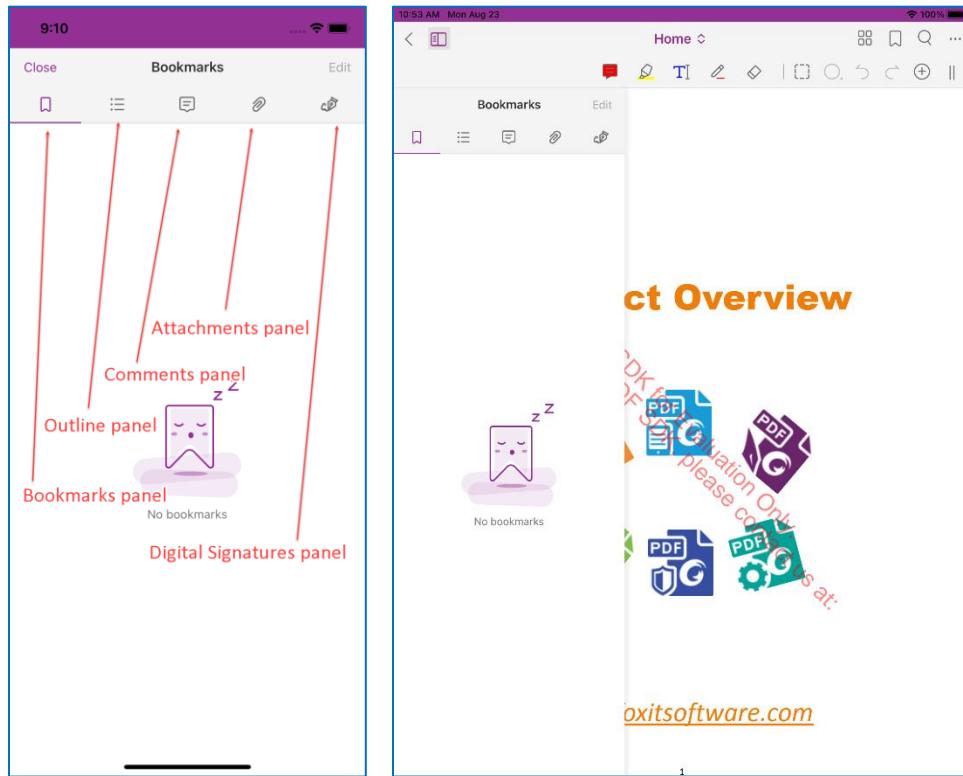


Figure 5-3

在本节中，我们在 "samples" 文件夹下的 "**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中提供了一个示例代码用来展示如何通过 APIs 来隐藏一个特定的面板。以"Outline"面板为例，对于其他的面板，您只需要修改 **FSPanelType** 的值。Panels 和 **FSPanelType** 之间的对应关系如下表所示：

Panel	FSPanelType
Bookmarks	FSPanelTypeReadingBookmark
Outline	FSPanelTypeOutline
Annotations	FSPanelTypeAnnotation
Attachments	FSPanelTypeAttachment
Digital Signatures	FSPanelTypeDigitalSignature

在 Xcode 中打开该 demos。将示例代码加入到 "**ViewController.m**" (Objective-C) 或者 "**TabsViewController.swift**" (Swift) 中 (加在 UIExtensionsManager 初始化之后)。

备注：手机和平板上面的内置 UI 有一些不同。下面列举的示例适用于手机和平板，在本手册中，如果在手机和平板上的自定义结果类似，则只列出在手机上的效果图。

示例 1：隐藏 "Outline" 面板。(适用于手机和平板)

Objective-C:

```
[self.extensionsMgr.panelController setPanelHidden:true type:FSPanelTypeOutline];
```

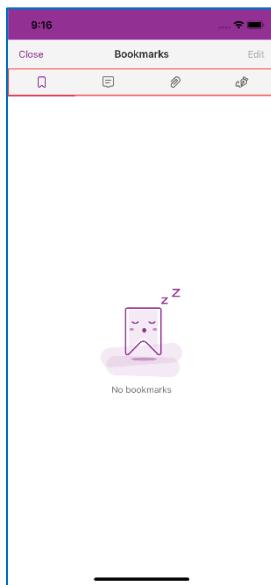
Swift:

```
extensionsManager.panelController.setPanelHidden(true, type: .outline);
```

修改前:



修改后:



5.2.3 自定义隐藏 View setting bar 上的 UI 元素

隐藏 View setting bar 上的 UI 元素(见 Figure 5-4, 手机端点击底部工具栏上的 查看, 平板端点击顶部工具栏中靠近 Home 的 图标找到 view), 您只需要使用下面的 API:

```
(void)setItem:(SettingItemType)itemType hidden:(BOOL)hidden;
```

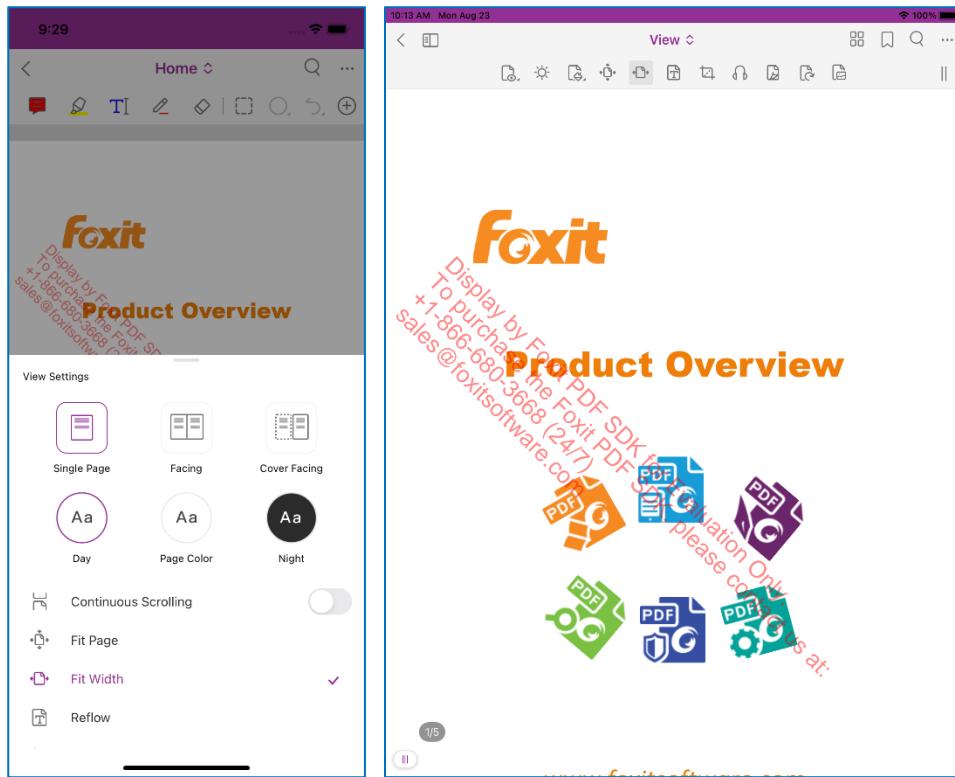


Figure 5-4

参数 "itemType" 的值可以参考如下的表格, 其对应 View setting bar 上的菜单项。

item	itemType
Single page mode	SINGLE
Facing mode	DOUBLEPAGE
Cover Facing mode	COVERPAGE
Day mode	DAYMODE
Page Color	PAGECOLOR
Night mode	NIGHTMODE
Continuous Scrolling mode	CONTINUOUS
Fit page mode	FITPAGE
Fit width mode	FITWIDTH
Reflow mode	REFLOW
Crop mode	CROPPAGE
Speak	SPEECH

Auto Flip	AUTOFLIP
Rotate View	ROTATE
Pan and Zoom	PANZOOM

在本节中，我们在 "samples" 文件夹下的 "**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中以 "Reflow" 菜单为例展示如何通过 APIs 来隐藏 View setting bar 上的 UI 元素。对于其他的 UI 元素，您只需要修改参数 "**itemType**"。

在 Xcode 中打开该 demos。将示例代码加入到 "**ViewController.m**" (Objective-C) 或者 "**TabsViewController.swift**" (Swift) 中 (加在 `UIExtensionsManager` 初始化之后)。

示例 1：隐藏 View setting bar 上面的 Reflow 菜单。(适用于手机和平板)

Objective-C:

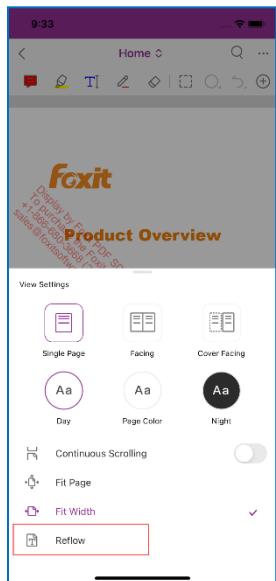
```
[self.extensionsMgr.settingBar setItem:REFLOW hidden:YES];
```

Swift:

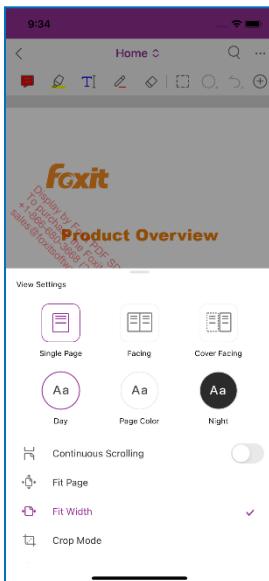
```
extensionsManager.settingBar.setItem(.REFLOW, hidden: true)
```

手机端：

修改前:

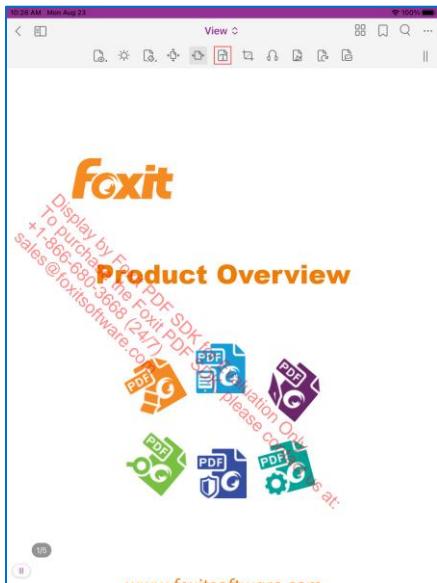


修改后:



平板端：

修改前：



修改后：



5.2.4 自定义隐藏 More Menu 菜单上的 UI 元素

隐藏 More Menu View 上的 UI 元素 (见 Figure 5-5, 在右上角的工具栏中点击 查看), 您可以使 Table 5-2 中列出的 APIs:

Table 5-2

<code>(void)setMoreViewItemHiddenWithGroup:(NSUInteger)groupTag hidden:(BOOL)isHidden</code>	根据"groupTag"的值显示或者隐藏该 group。
<code>(void)setMoreViewItemHiddenWithGroup:(NSUInteger)groupTag andItemTag:(NSUInteger)itemTag hidden:(BOOL)isHidden</code>	根据"groupTag" 和 "itemTag"的值显示或者隐藏 group 中的子菜单项。
<code>(void) setIndividualMenuItemHiddenWithTag:(NSUInteger)itemTag hidden: (BOOL)isHidden</code>	显示或者隐藏没有子菜单的 group。

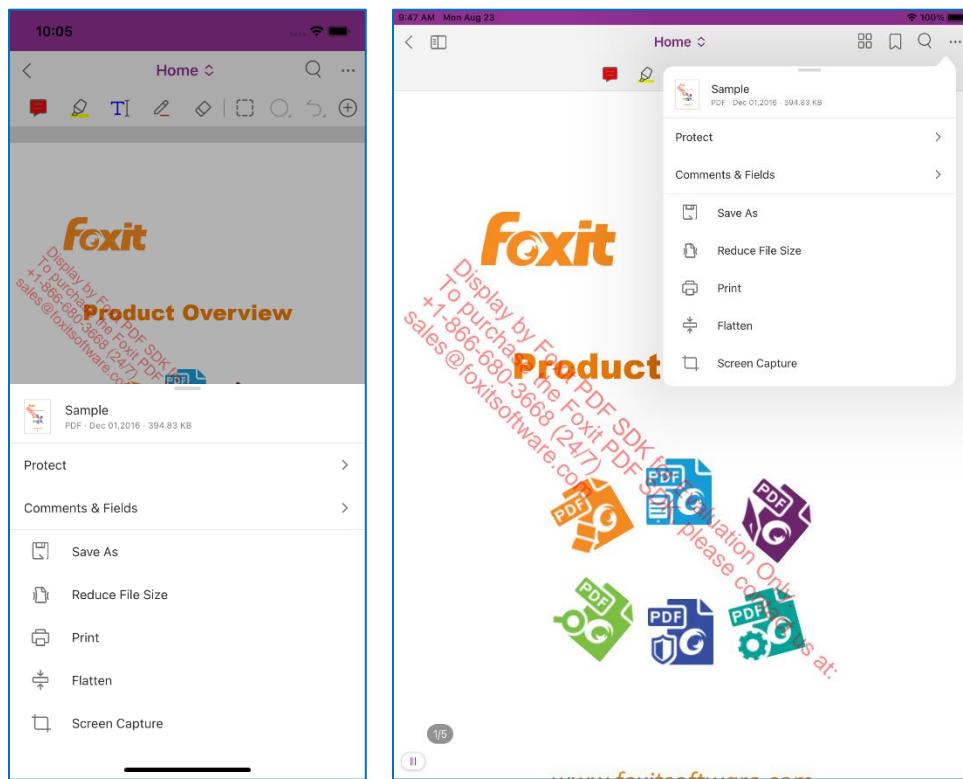


Figure 5-5

setMoreViewItemHiddenWithGroup 接口中的 "groupTag" 和 "itemTag" 参数可以设置如下所示：

groupTag	NSUInteger
TAG_GROUP_PROTECT	10
TAG_GROUP_COMMENT_FIELD	20

groupTag	itemTag	NSUInteger
TAG_GROUP_PROTECT	TAG_ITEM_REDACTION	10
	TAG_ITEM_PASSWORD	20
	TAG_ITEM_CERTIFICATE	30
TAG_GROUP_COMMENT_FIELD	TAG_ITEM_IMPORTCOMMENT	40
	TAG_ITEM_EXPORTCOMMENT	50
	TAG_ITEM_SUMARIZECOMMENT	60
	TAG_ITEM_RESETFORM	70
	TAG_ITEM_IMPORTFORM	80
	TAG_ITEM_EXPORTFORM	90

setIndividualMenuItemHiddenWithTag 接口中的 "itemTag" 参数可以设置如下所示：

itemTag	NSUInteger

TAG_ITEM_SAVE_AS	110
TAG_ITEM_REDUCEFILESIZE	120
TAG_ITEM_WIRELESSPRINT	130
TAG_ITEM_FLATTEN	140
TAG_ITEM_SCREENCAPTURE	150

在本节中，我们在 "samples" 文件夹下的 "**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中以 "TAG_GROUP_FILE" (**File**) 和 "TAG_ITEM_FILEINFO" (**File Information**) 为例展示如何通过 APIs 来显示或者隐藏 More Menu view 上的 UI 元素。对于其他的 UI 元素，您可以参考下面的示例，只需要修改 **setMoreViewItemHiddenWithGroup** 接口中的参数值。

在本节中，我们提供 3 个示例：

- **Example1** 和 **Example2** 用来展示如何通过 APIs 来隐藏 More Menu view 中特定的组或者菜单。我们在 "samples" 文件夹下的 "**complete_pdf_viewer**" (Objective-C) 和 "**complete_pdf_viewer_swift**" (Swift) demos 中进行示例展示。以 **TAG_GROUP_PROTECT** 组和 **TAG_ITEM_REDACTION** 菜单为例，对于其他的组和菜单，请参考该示例，只需要修改 **setMoreViewItemHiddenWithGroup** 接口中的参数值。
- **Example3** 用来展示如何通过 APIs 来隐藏 More Menu view 中单个的菜单项。以 **TAG_ITEM_FLATTEN** 菜单为例，对于其他的单个菜单项，请参考该示例，只需要修改 **setIndividualMenuItemHiddenWithTag** 接口中的参数值。

在 Xcode 中打开该 demos。将示例代码加入到 "**ViewController.m**" (Objective-C) 或者 "**TabsViewController.swift**" (Swift) 中 (加在 **UIExtensionsManager** 初始化之后)。

备注：手机和平板上面的内置 UI 有一些不同。下面列举的示例适用于手机和平板，在本手册中，如果在手机和平板上的自定义结果类似，则只列出在手机上的效果图。

示例 1：隐藏 More Menu view 上面的 "**TAG_GROUP_PROTECT**" 组。(适用于手机和平板)

Objective-C:

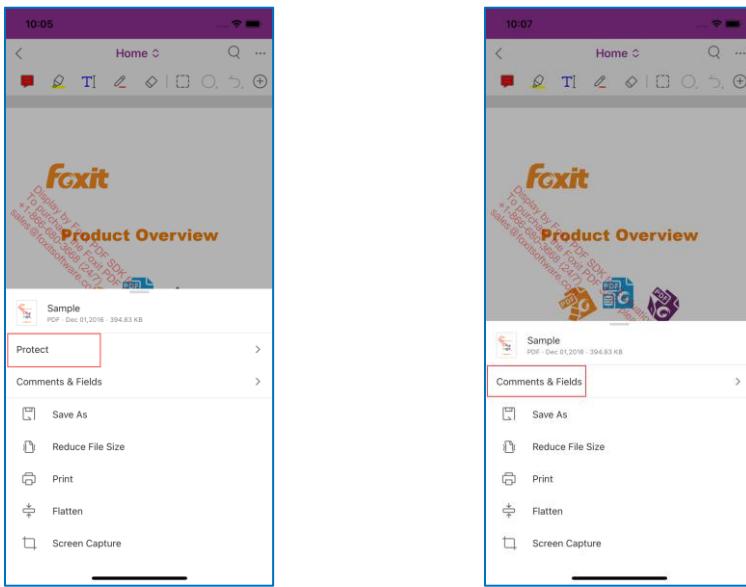
```
[self.extensionsMgr.more setMoreViewItemHiddenWithGroup:TAG_GROUP_PROTECT hidden:YES];
```

Swift:

```
extensionsManager.more.setMoreViewItemHiddenWithGroup(UInt(TAG_GROUP_PROTECT), hidden: true);
```

修改前:

修改后:



示例 2：隐藏 More Menu view 上面的 "TAG_ITEM_REDACTION" 菜单。(适用于手机和平板)

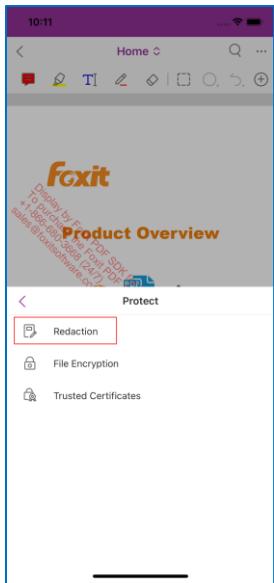
Objective-C:

```
[self.extensionsMgr.more setMoreViewItemHiddenWithGroup:TAG_GROUP_PROTECT  
andItemTag:TAG_ITEM_REDACTION hidden:YES];
```

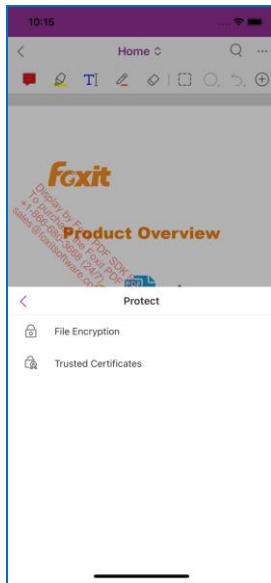
Swift:

```
extensionsManager.more.setMoreViewItemHiddenWithGroup(UInt(TAG_GROUP_PROTECT),  
andItemTag:UInt(TAG_ITEM_REDACTION), hidden: true);
```

修改前:



修改后:



示例 3：隐藏 More Menu view 上面的单个菜单项 "TAG_ITEM_FLATTEN"。(适用于手机和平板)

Objective-C:

```
[self.extensionsMgr.more setIndividualMenuItemHiddenWithTag:TAG_ITEM_FLATTEN hidden:YES];
```

Swift:

```
extensionsManager.more.setIndividualMenuItemHiddenWithTag(UInt(TAG_ITEM_FLATTEN), hidden: true);
```

修改前:



修改后:



5.3 通过源代码自定义 UI 实现

在前面的章节中，我们详细地介绍了如何通过配置文件或者 APIs 对 UI 进行自定义。这些修改是基于 Foxit PDF SDK for iOS 的内置 UI 框架的。如果您不想使用当前现成的 UI 框架，您可以通过修改 UI Extensions 组件中的源代码来重新设计 UI。

有一点需要注意。UI Extensions 组件的源代码是用 Objective-C 编写的，因此，您需要使用 Objective-C 来修改 UI 布局。如果您是 Swift 开发人员并且对 Objective-C 不太熟悉，那么您可能只能自定义那些不需要编写代码的 UI 外观，比如图标和其他 UI 资源等。

为了自定义 UI 实现，您可以按照下面的步骤：

首先，在您的工程中添加如下的文件，

- **FoxitRDK.framework** – 该 framework 包含 Foxit PDF SDK for iOS 的动态库和相关头文件。该 framework 在"libs" 文件夹下。
- **uiextensions** 工程 – 是一个开源库，包含了一些即用型的 UI 模块实现和应用程序基本的 UI 设计，可以帮助开发人员快速将功能齐全的 PDF 阅读器嵌入到他们的 iOS 应用中。当然，开发人员也不是必须要使用默认的 UI，可以通过"uiextensions"工程为特定的应用灵活自定义和设计 UI。该工程在 "libs/uiextensions_src" 文件夹下。

技巧: 内置 UI 的自定义可以直接在 **uiextensions** 工程中完成，然后编辑该工程，将生成的 **uiextensionsDynamic.framework** 添加到您的应用程序，这样就不用将整个 **uiextensions** 项目添加到您的应用程序中。

其次，在 **uiextensions** 工程中定位到您需要自定义的 UI 的相关代码或者图片，然后根据您的需求进行修改。

为方便起见，我们将在"sample"文件夹下的"**viewer_ctrl_demo**" 中向您展示如何自定义 UI 实现。

UI 自定义示例

步骤 1: 向 "**viewer_ctrl_demo**" 中添加 **uiextensions** 工程。

备注: 为了方便查看自定义修改结果，我们将添加 **uiextensions** 工程到 **demo** 中。该 **demo** 已经包含了 **FoxitRDK.framework**，因此，只需要添加 **uiextensions** 工程。

在 Xcode 中打开"**viewer_ctrl_demo**" 工程。将下载解压包中"libs/uiextensions_src"下的 "**uiextensions.xcodeproj**" 拖拽到"**viewer_ctrl_demo**" 工程中，如 Figure 5-6 所示。

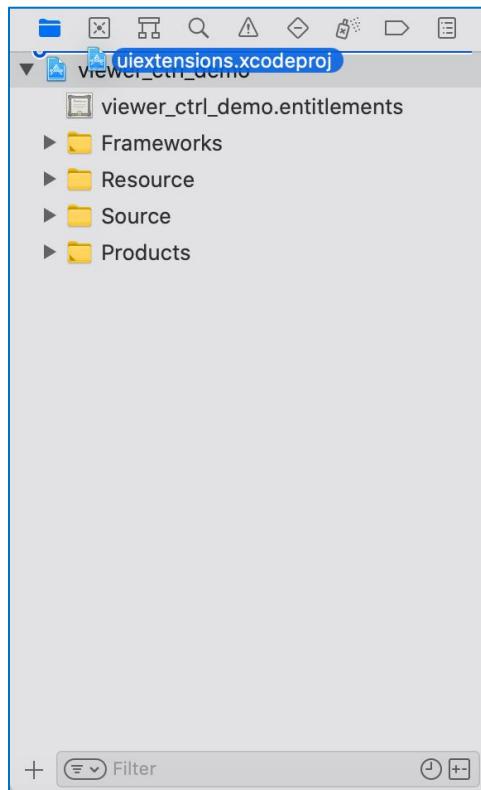


Figure 5-6

然后会弹出一个对话框提示您是否保存该工程到一个新的 workspace，如 Figure 5-7 所示。点击 **Save**。

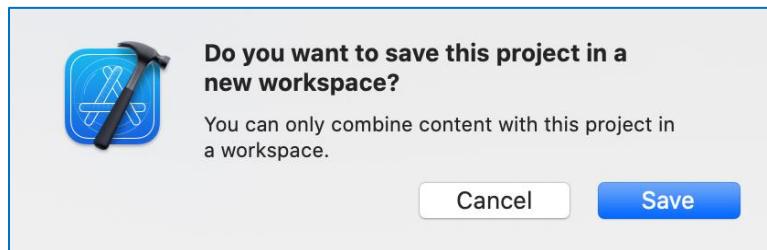


Figure 5-7

保存新的 workspace 到"samples" 文件夹，并且命名为"custom_viewer"，如 Figure 5-8 所示。点击 **Save**。

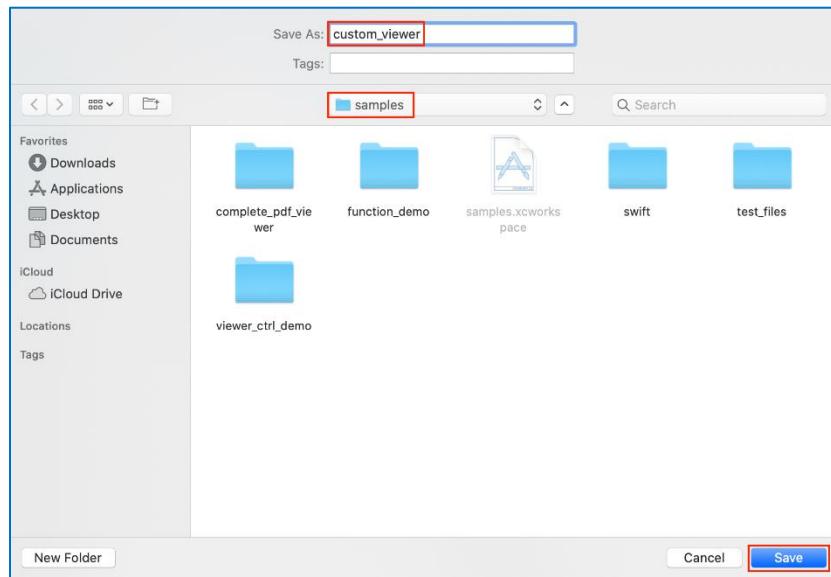


Figure 5-8

则，新的 workspace 将如 Figure 5-9 所示。

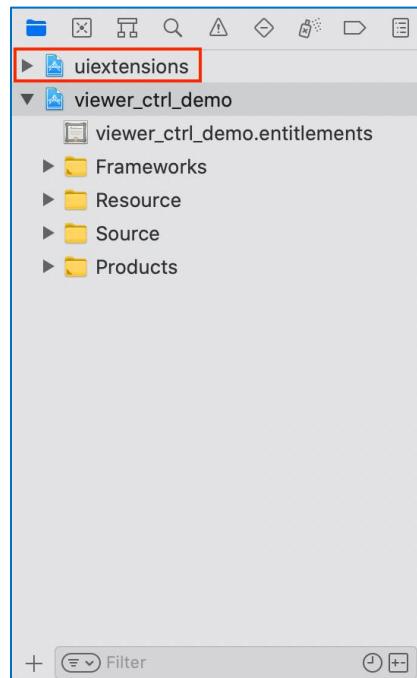


Figure 5-9

恭喜您！您已经完成了第一步。

步骤 2：查找和修改与您需要自定义的 UI 相关的布局文件。定位到您需要自定义的 UI 的相关代码或者图片，然后根据需求进行修改。

现在，我们将向您展示一个简单的示例，在搜索面板中修改 button 的图标，如 Figure 5-10 所示。

技巧：如果您只需要更改 UI 元素的图标，可以直接使用 **uiextensionsDynamic.framework**，而不需要导入 **uiextensions** 工程源代码。但同样的的是，您都需要找到图标的名称。



Figure 5-10

要替换图标，我们只需要找到存储该 button 图标的位置，然后使用另一个具有相同名称的图标来替换它。

本节使用 iPhone 11 模拟器来运行该 demo。在 **uiextensions** 工程中，点击 "**uiextensionsDynamic -> Resource -> png -> Search.xcassets**"，如 Figure 5-11 所示。很容易找到我们需要替换的图片。资源文件是根据功能来存储的，因此您可以通过图标的名称找到相关的代码。

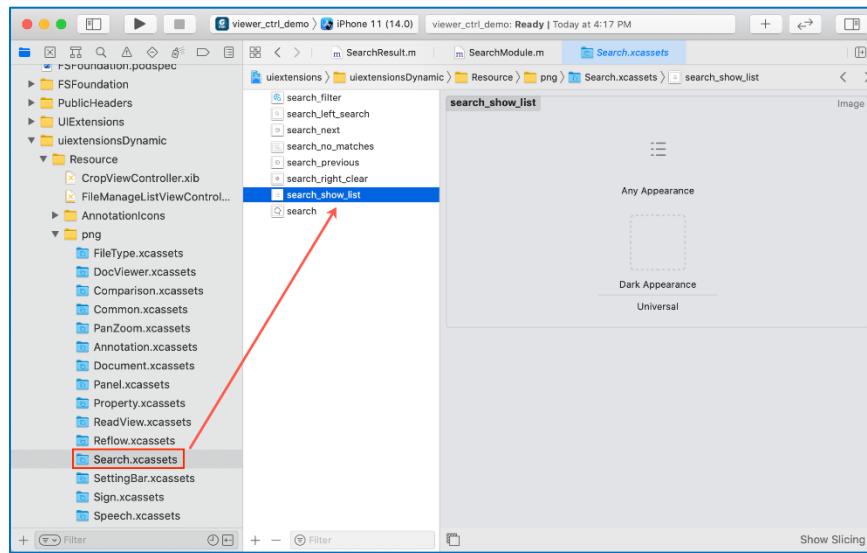


Figure 5-11

现在，只需要在 "libs\uiextensions_src\UIExtensions\Resource\png\Search.xcassets" 文件夹下，用您自己的图标文件替换 "**search_show_list.imageset**" 即可。例如，使用顶部的搜索按钮 (**search.imageset**) 来替换它。

替换完成后，首先编译和运行 **uiextensionsDynamic_aggregate** 工程，如 Figure 5-12 所示。

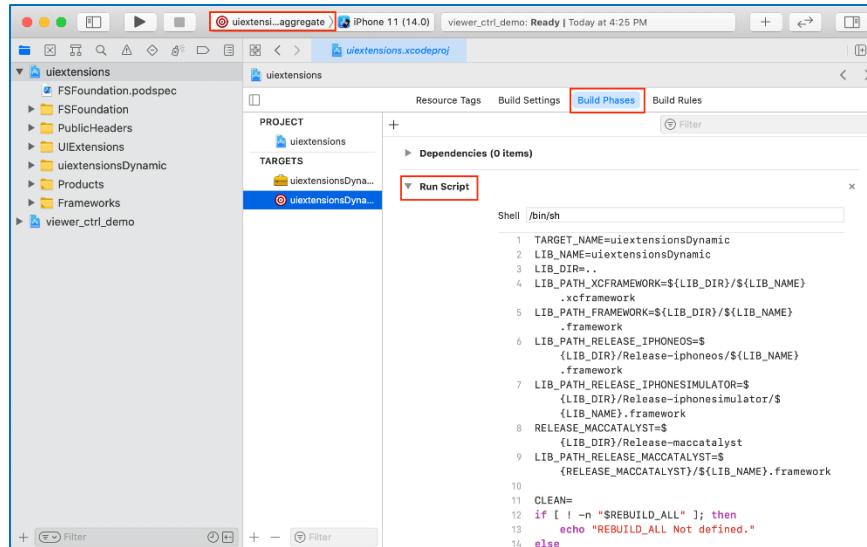


Figure 5-12

备注：*uiextensions* 工程可以通过编译 **uiextensions_aggregate** 生成通用的 ".a" 库，也可以通过编译 **uiextensionsDynamic_aggregate** 生成通用的 framework，可用于模拟器或者 iOS 真机设备。在本节中，我们编译 **uiextensionsDynamic_aggregate**。*uiextension* 工程中用于生成通用 framework 的脚本如 Figure 5-12 所示。

当成功编译 **uiextensionsDynamic_aggregate** 工程后，下载解压包"libs"文件夹下的 **uiextensionsDynamic.framework** 将会被新生成的 framework 覆盖。

然后，编译和运行"**viewer_ctrl_demo**"工程。成功编译后，使用搜索功能，可以看到底部的搜索按钮的图标已更改，如 Figure 5-13 所示。

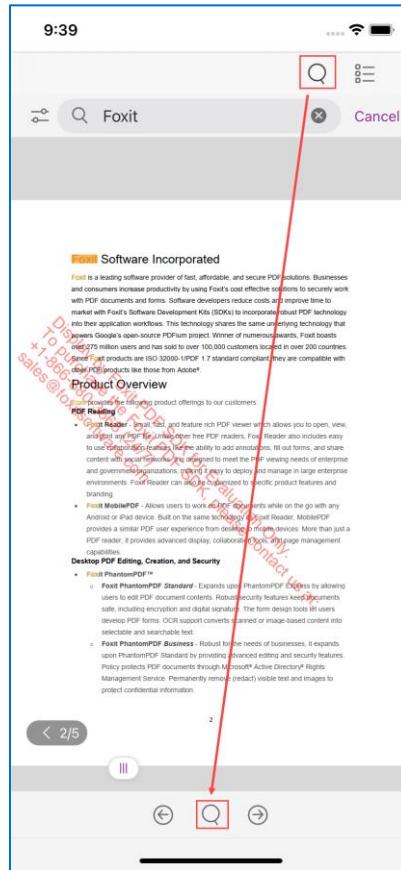


Figure 5-13

这只是一个简单的示例用来展示如何自定义 UI 实现。您可以作为参考，通过 **uiextensions** 工程您可以自由的对特定的应用程序进行 UI 自定义和设计。

6 使用 SDK API

Foxit PDF SDK for iOS 将所有功能实现封装在 UI Extensions 组件中。如果您对功能实现的详细过程感兴趣，请参考本节内容。

在本节中，我们将介绍 Foxit PDF SDK for iOS 的主要功能，并列举相关示例来展示如何使用 Foxit PDF SDK Core API 实现这些功能。

6.1 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 `FSRenderer::setRenderContentFlags` 接口来决定是否同时渲染页面和注释，然后使用 `FSRenderer::startRender` 接口进行渲染。
`FSRenderer::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 `FSRenderer::renderAnnot` 接口。
- 在位图上渲染，使用 `FSRenderer::startRenderBitmap` 接口。
- 渲染一个重排的页面，使用 `FSRenderer::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 `FSFiller` 对象来填表，则应使用 `FSFiller::render` 接口来渲染当前获取到焦点的表单控件，而不是使用 `FSRenderer::renderAnnot` 接口。

Example:

6.1.1 如何将指定的 PDF 页面渲染到 bitmap

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
-(FSBitmap*)renderPageToBitmap:(FSPDFPage*) pdfPage drawWidth:(int)drawPageWidth
drawHeight:(int)drawPageHeight
{
    // If the page hasn't been parsed yet, throw an exception.
```

```

if (![pdfPage isParsed])
    @throw [NSError exceptionWithName:NSGenericException reason:@"PDF Page should be parsed
first" userInfo:nil];

// Pepar matrix to render on the bitmap.
FSMatrix2D* matrix = [pdfPage getDisplayMatrix:0 top:0 width:drawPageWidth height:drawPageHeight
rotate:FSRotation0];
// Create a bitmap according to the required drawPageWidth and drawPageHeight.
FSBitmap* bitmap = [[FSBitmap alloc] initWithWidth:drawPageWidth height:drawPageHeight
format:FSBitmapDIBRgb];
// Fill the bitmap with white color.
[bitmap fillRect:0xFFFFFFFF rect:nil];
FSRenderer* renderer = [[FSRenderer alloc] initWithBitmap:bitmap is_rgb_order:YES];
// Set the render flag, both page content and annotation will be rendered.
[renderer setRenderContentFlags:FSRendererRenderPage | FSRendererRenderAnnot];
// Start to render the page progressively.
FSProgressive* progress = [renderer startRender:pdfPage matrix:matrix pause:nil];
if (progress) {
    FSProgressiveState state = [progress resume];
    while (state == FSProgressiveToBeContinued) {
        state = [progress resume];
    }
    if (state != FSProgressiveFinished)
        return nil;
}
return bitmap;
}

```

6.1.2 如何将指定的 PDF 页面渲染到平台设备上下文

```

#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...

-(void)renderPageToContext:(FSPDFPage*) pdfPage context:(CGContextRef)context
{
    // If the page hasn't been parsed yet, throw an exception.
    if (![pdfPage isParsed])
        @throw [NSError exceptionWithName:NSGenericException reason:@"PDF Page should be parsed
first" userInfo:nil];

    // We set the width of drawing page to be equal to screen width, the drawing page height is calculated according to
the ratio of page height and width.
    CGFloat scale = [UIScreen mainScreen].scale;
    int drawPageWidth = (int)[[UIScreen mainScreen] bounds].size.width * scale;
    float pageWidth = [pdfPage getWidth];
    float pageHeight = [pdfPage getHeight];
}

```

```

int drawPageHeight = (int)drawPageWidth * (pageHeight/pageWidth) * scale;

// Erase the background of context with white color.
CGContextSaveGState(context);
CGContextSetRGBFillColor(context, 1.0, 1.0, 1.0, 1.0);
CGContextFillRect(context, CGRectMake(0, 0, drawPageWidth, drawPageHeight));

// Render to screen in the device coordinate, left:0, top:0, right:drawPageWidth, bottom:drawPageHeight.
FSMatrix2D* matrix = [pdfPage getDisplayMatrix:0 top:0 width:drawPageWidth height:drawPageHeight
rotate:FSRotation0];
FSRenderer* renderer = [[FSRenderer alloc] initWithContext:context device_type:FSRendererDeviceDisplay];
[renderer setRenderContentFlags:FSRendererRenderPage | FSRendererRenderAnnot];
// Start to render the page progressively.
FSProgressive* progress = [renderer startRender:pdfPage matrix:matrix pause:nil];
if(progress) {
    FSProgressiveState state = [progress resume];
    while (state == FSProgressiveToBeContinued) {
        state = [progress resume];
    }
}
CGContextRestoreGState(context);
}

```

6.2 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 `FSTextPage` 对象中。`FSTextPage` 类可用于检索 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 `FSTextSearch` 对象来构建 `FSTextPage` 对象。
- 访问类似超文本链接的文本，使用 `FSTextPage` 对象来构建 `FSPageTextLinks` 对象。
- 高亮 PDF 页面上的选中文本，构建一个 `FSTextPage` 对象来计算选中文本区域。

Example:

6.2.1 如何通过选择获取页面上的文本区域

```

#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...

// Get the text area on a page by selection. The starting selection position and ending selection position are specified by startPos and endPos.
-(NSArray<FSRectF*>*)getTextRectsBySelection:(FSPDFPage*)page startPos:(FSPointF*)startPos
endPos:(FSPointF*)endPos {

```

```
if (![page isParsed])
    @throw [NSError exceptionWithName:NSGenericException reason:@"PDF Page should be parsed
first" userInfo:nil];

// Create a text page from the parsed PDF page.
FSTextPage* textPage = [[FSTextPage alloc] initWithPage:page flags:FSTextPageParseTextNormal];
if (!textPage || [textPage isEmpty])
    return nil;
int startCharIndex = [textPage getIndexAtPos:startPos.x y:startPos.y tolerance:5];
int endCharIndex = [textPage getIndexAtPos:endPos.x y:endPos.y tolerance:5];
// API getTextRectCount requires that start character index must be lower than or equal to end character index.
startCharIndex = startCharIndex < endCharIndex ? startCharIndex : endCharIndex;
endCharIndex = endCharIndex > startCharIndex ? endCharIndex : startCharIndex;
int count = [textPage getTextRectCount:startCharIndex count:endCharIndex-startCharIndex];
if (count)
{
    NSMutableArray<FSRectF*>* array = [[NSMutableArray<FSRectF*> alloc] init];
    for (int i=0; i<count; i++) {
        FSRectF* rect = [textPage getTextRect:i];
        if (!rect || [rect isEmpty])
            continue;
        [array addObject:rect];
    }
    // The return rects are in PDF unit, if caller need to highlight the text rects on the screen, then these rects should
    // be converted in device unit first.
    return array;
}
return nil;
}

...
```

6.3 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 `FSTextSearch::setPattern`、`FSTextSearch::setStartPage` (仅对 PDF 文档中的文本搜索有用)、`FSTextSearch::setEndPage` (仅对 PDF 文档中的文本搜索有用)、和 `FSTextSearch::setSearchFlags` 接口。
- 进行搜索，使用 `FSTextSearch::findNext` 和 `FSTextSearch::findPrev` 接口。
- 获取搜索结果，使用 `FSTextSearch::getMatchXXX()` 接口。

Example:

6.3.1 如何在 PDF 文档中搜索指定的文本模型

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
NSString *pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:pdfPath];

// Create a text search handler for searching in PDF document.
FSTextSearch *textSearch = [[FSTextSearch alloc] initWithDocument:doc cancel:nil
flags:FSTextPageParseTextNormal];
// Set the start page index which searching will begin. By default, end page will be the last page.
[textSearch setStartPage:0];
// Set the text to be searched.
[textSearch setPattern:@"foxit"];
// Set the search flags to be matching case and matching whole word.
[textSearch setSearchFlags:FSTextSearchSearchMatchCase | FSTextSearchSearchMatchWholeWord];
// Start to search from the start page to end.
while([textSearch findNext]) {
    // If true, then we found a matched result.
    // Get the found page index.
    int pageIndx = [textSearch getMatchPageIndex];
    // Get the start character index of the matched text on the found page.
    int startCharIndex = [textSearch getMatchStartCharIndex];
    // Get the end character index of the matched text on the found page.
    int endCharIndex = [textSearch getMatchEndCharIndex];
    // Get the rectangular region of the matched text on the found page.
    FSRectFArray* matchRects = [textSearch getMatchRects];
}
}
```

6.4 Bookmark (Outline)

Foxit PDF SDK 提供了名为 Bookmarks 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部分。PDF 书签也称为 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 `FSPDFDoc::getRootBookmark` 以获取整个 bookmark 树的 root。这里，“root bookmark”是一个抽象对象，它只有一些 child bookmarks，没有 next sibling bookmarks，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 `FSBookmark::getFirstChild`。

在检索 root bookmark 后，就可以调用以下的接口去访问其他的 bookmarks：

- 访问 parent bookmark，使用 `FSBookmark::getParent` 接口。
- 访问第一个 child bookmark，使用 `FSBookmark::getFirstChild` 接口。
- 访问 next sibling bookmark，使用 `FSBookmark::getNextSibling` 接口。
- 插入一个新的 bookmark，使用 `FSBookmark::insert` 接口。
- 移动一个 bookmark，使用 `FSBookmark::moveTo` 接口。

Example:

6.4.1 如何使用深度优先顺序遍历 PDF 文档的 bookmarks

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
- (void)DepthFistTravelBookmarkTree:(FSBookmark*)bookmark document:(FSPDFDoc*)doc {
    if(!bookmark || [bookmark isEmpty])
        return;
    [self DepthFistTravelBookmarkTree:[bookmark getChild] document:doc];
    while(true) {
        // Get bookmark title.
        NSString* title = [bookmark getTitle];
        FSDestination* dest = [bookmark getDestination];
        if(dest && ![dest isEmpty])
        {
            float left,right,top,bottom;
            float zoom;
            int pageIndex = [dest getPageIndex:doc];
            // left, right, top, bottom, zoom are only meaningful with some special zoom modes.
            FSDestinationZoomMode mode = [dest getZoomMode];
            switch (mode) {
                case FSDestinationZoomXYZ:
                    left = [dest getLeft];
                    top = [dest getTop];
                    zoom = [dest getZoomFactor];
                    break;
                case FSDestinationZoomFitPage:
                    break;
                case FSDestinationZoomFitHorz:
                    top = [dest getTop];
                    break;
                case FSDestinationZoomFitVert:
                    left = [dest getLeft];
                    break;
                case FSDestinationZoomFitRect:
                    left = [dest getLeft];
                    bottom = [dest getBottom];
                    break;
            }
        }
    }
}
```

```

        right = [dest getRight];
        top = [dest getTop];
        break;
    case FSDestinationZoomFitBBox:
        break;
    case FSDestinationZoomFitBHorz:
        top = [dest getTop];
        break;
    case FSDestinationZoomFitBVert:
        left = [dest getLeft];
        break;
    default:
        break;
    }
}
bookmark = [bookmark getNextSibling];
if(bookmark == nil || [bookmark isEmpty])
    break;
[self DepthFirstTravelBookmarkTree:[bookmark getChild] document:doc];
}
}

```

6.5 Reading Bookmark

Reading bookmark 不是 PDF bookmark，换句话说，它不是 PDF outlines。Reading bookmark 是应用层的书签。它存储在目录的元数据（XML 格式）中，允许用户根据他们的阅读偏好添加或删除 reading bookmark，并通过选择 reading bookmark 可以轻松导航到一个 PDF 页面。

为了检索 reading bookmark，可以调用 [FSPDFDoc::getReadingBookmarkCount](#) 接口来计算其个数，并且可以调用 [FSPDFDoc::getReadingBookmark](#) 接口以索引方式获取相应的 reading bookmark。

此类提供了接口用来获取/设置 reading bookmarks 属性，比如标题，目标页面索引，以及创建/修改日期时间。

Example:

6.5.1 如何添加自定义 reading bookmark 并枚举所有的 reading bookmarks

```

#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
// Add a new reading bookmark to pdf document, the returned bookmark stores the title and the page index.
- (FSReadingBookmark*)addReadingBookmark:(FSPDFDoc*)pdfDoc title:(NSString*)title
pageIndex:(int)pageIndex {

```

```

int count = [pdfDoc getReadingBookmarkCount];
return [pdfDoc insertReadingBookmark:count title:title dest_page_index:pageIndex];
}

// Enumerate all the reading bookmarks from the pdf document.
- (void)getReadingBookmark:(FSPDFDoc*) pdfDoc {
    int count = [pdfDoc getReadingBookmarkCount];
    for(int i=0; i<count; i++) {
        FSReadingBookmark* bm = [pdfDoc getReadingBookmark:i];
        if([bm isEmpty]) continue;
        // Get bookmark title.
        NSString* title = [bm getTitle];
        // Get the page index which associated with the bookmark.
        int pageIndex = [bm getPageIndex];
        // Get the creation date of the bookmark.
        FSDateTime* creationDate = [bm getDate:YES];
        // Get the modification date of the bookmark.
        FSDateTime* modificationDate = [bm getDate:NO];
    }
}

```

6.6 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:

6.6.1 如何将指定文件嵌入到 PDF 文档

```

#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...

NSString* filePath = @"/xxx/fileToBeEmbedded.xxx";
FSPDFNameTree* nameTree = [[FSPDFNameTree alloc] initWithDocument:self.fspdfdoc
type:FSPDFNameTreeEmbeddedFiles];
FSAttachments* attachments = [[FSAttachments alloc] initWithDoc:self.fspdfdoc nametree:nameTree];
FSFileSpec* fileSpec = [[FSFileSpec alloc] initWithDocument:self.fspdfdoc];
[fileSpec setFileName:[filePath lastPathComponent]];
if(![fileSpec embed:filePath])
    return;
[attachments addEmbeddedFile:[filePath lastPathComponent] file_spec:fileSpec];
...

```

6.6.2 如何从 PDF 文档中导出嵌入的附件文件，并将其另存为单个文件

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
// Extract the embedded attachment file.
int count = [attachments getCount];
for(int i=0; i<count; i++) {
    NSString* key = [attachments getKey:i];
    if(key) {
        FSFileSpec* fileSpec = [attachments getEmbeddedFile:key];
        NSString* exportedFile =[@"/somewhere/" stringByAppendingString:[fileSpec getFileName]];
        if(fileSpec && ![fileSpec isEmpty]) {
            fileSpec exportToFile:exportedFile];
        }
    }
}
...
}
```

6.7 Annotation

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。PDF 包括如 Table 6-1 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。Table 6-1 中的 "Markup" 列用来说明是否为标记注释。

Foxit PDF SDK 支持 PDF Reference 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 6-1

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

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

备注：Foxit SDK 支持名为 PSI (pressure sensitive ink, 压感笔迹) 的自定义注释类型。在 PDF 规范中没有对该注释进行描述。通常，PSI 用于手写功能，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:

6.7.1 如何向 PDF 页面中添加注释

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
NSString *pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:pdfPath];
FSPDFPage *pdfPage = [doc getPage:0];
// Add text annot.
FSRectF *rect = [[FSRectF alloc] initWithLeft1:100 bottom1:100 right1:120 top1:120];
FSNote *note = [[FSNote alloc] initWithAnnot:[pdfPage addAnnot:FSAannotNote rect:rect]];
if (!note || [note isEmpty]) {
    return;
}
[note setIconName:@"Comment"];
// Set color to blue.
[note setBorderColor:0xff0000ff];
[note setContent:@"This is the note comment, write any content here."];
[note resetAppearanceStream];
```

```

// The following code demonstrates how to add hightlight annotation on the searched text.
FSTextSearch *textSearch = [[FSTextSearch alloc] initWithDocument:doc cancel:nil
flags:FSTextPageParseTextNormal];
if (!textSearch || [textSearch isEmpty]) {
    return;
}
// Suppose that the text for highlighting is "foxit".
[textSearch setPattern:@"foxit"];
BOOL bMatched = [textSearch findNext];
if (bMatched) {
    FSRectFArray *rects = [textSearch getMatchRects];
    int rectCount = [rects getSize];
    // Fill the quadpoints array according to the text rects of matched result.
FSQuadPointsArray* arrayOfQuadPoints = [[FSQuadPointsArray alloc] init];
for (int i = 0; i < rectCount; i++) {
    FSRectF *rect = [rects objectAtIndex:i];
    FSQuadPoints *quadPoints = [[FSQuadPoints alloc] init];
    FSPointF *point = [[FSPointF alloc] init];
    [point set:[rect getLeft] y:[rect getTop]];
    [quadPoints setFirst:point];
    [point set:[rect getRight] y:[rect getTop]];
    [quadPoints setSecond:point];

    [point set:[rect getLeft] y:[rect getBottom]];
    [quadPoints setThird:point];
    [point set:[rect getRight] y:[rect getBottom]];
    [quadPoints setFourth:point];
    [arrayOfQuadPoints addObject:quadPoints];
}
// Just set an empty rect to markup annotation, the annotation rect will be calculated according to the quadpoints that set to it later.
FSRectF *rect = [[FSRectF alloc] initWithLeft1:0 bottom1:0 right1:0 top1:0];
FSTextMarkup *textMarkup = [[FSTextMarkup alloc] initWithAnnot:[pdfPage addAnnot:FSAnnotHighlight
rect:rect]];
// Set the quadpoints to this markup annot.
[markup setQuadPoints:arrayOfQuadPoints];
// set to red.
[markup setBorderColor:0xffff0000];
// set to thirty-percent opacity.
[markup setOpacity:0.3f];
// Generate the appearance.
[markup resetAppearanceStream];
}

```

6.7.2 如何删除 PDF 页面中的注释

```
#import "ViewController.h"
```

```
#import <FoxitRDK/FSPDFViewControl.h>
...
NSString *pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:pdfPath];
FSPDFPage *pdfPage = [doc getPage:0];
// Remove an annot by index.
FSAnnot* annot = [pdfPage getAnnot:0];
if(!annot || [annot isEmpty])
    return;
// Remove the first annot, so the second annot will become first.
[pdfPage removeAnnot:annot];
```

6.8 Form

Form (AcroForm) 是用于收集用户交互信息的表单域的集合。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。FSForm 类提供了 APIs 用来检索表单域或表单控件，导入/导出表单数据，以及其他功能，例如：

- 检索表单域，使用 `FSForm::getCount` 和 `FSForm::getForm` 接口。
- 检索 PDF 页面中的表单控件，使用 `FSForm::getControlCount` 和 `FSForm::getControl` 接口。
- 从 XML 文件导入表单数据，使用 `FSForm::importFromXML` 接口；导出表单数据到 XML 文件，使用 `FSForm::exportToXML` 接口。
- 检索 form filler 对象，使用 `FSForm::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 `FSPDFDoc::importFromFDF` 和 `FSPDFDoc::exportToFDF` 接口。

Example:

6.8.1 如何通过 XML 文件导入表单数据或将表单数据导出到 XML 文件

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
NSString *pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
FSPDFDoc *doc = [[FSPDFDoc alloc] initWithPath:pdfPath];
// Check if the document has a form.
BOOL hasForm = [doc hasForm];
if(hasForm) {
    // Create a form object from document.
    FSForm* form = [[FSForm alloc] initWithDocument:doc];
```

```
// Export the form data to a XML file.
[form exportToXML:@"/somewhere/export.xml"];
// Or import the form data from a XML file.
[form importFromXML:@"/somewhere/export.xml"];
}
```

6.9 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。

Example:

6.9.1 如何使用密码加密 PDF 文件

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
// Encrypt the source pdf document with specified owner password and user password, the encrypted PDF will be
// saved to the path specified by parameter savePath.
-(BOOL) encryptPDF:(FSPDFDoc*) pdfDoc ownerPassword:(NSString*)ownerPassword
userPassword:(NSString*)userPassword savePath:(NSString*)savePath
{
    if(!pdfDoc || (!ownerPassword && !userPassword) || !savePath)
        return NO;
    // The encryption setting data. Whether to encrypt meta data:YES, User permission: modify,assemble,fill form.
    // Cipher algorithm:AES 128.
    FSStdEncryptData* encryptData = [[FSStdEncryptData alloc] initWithIs_encrypt_metadata:YES
user_permissions:(FSPDFDocPermModify | FSPDFDocPermAssemble | FSPDFDocPermFillForm)
cipher:FSSecurityHandlerCipherAES key_length:16];
    FSStdSecurityHandler * stdSecurity = [[FSStdSecurityHandler alloc] init];
    if (![stdSecurity initialize:encryptData user_password:userPassword owner_password:ownerPassword])
        return NO;
    [pdfDoc setSecurityHandler:stdSecurity];
    if (![pdfDoc saveAs:savePath save_flags:FSPDFDocSaveFlagNormal])
        return NO;
    return YES;
}
```

6.10 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:

6.10.1 如何对 PDF 文档进行签名，并验证签名

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>
...
- (void)addNewSignatureAndSign:(FSPDFPage*)page rect:(FSRectF*)rect {
    // Add a new signature on the specified page rect.
    FSSignature* signature = [page addSignature:rect];
    // Set the appearance flags, if the specified flag is on, then the associated key will be displayed on the signature appearance.
    [signature setAppearanceFlags:FSSignatureAPFlagLabel | FSSignatureAPFlagDN | FSSignatureAPFlagText |
     FSSignatureAPFlagLocation | FSSignatureAPFlagReason | FSSignatureAPFlagSigner];
    // Set signer.
    [signature setKeyValue:FSSignatureKeyNameSigner value:@"Foxit"];
    // Set location.
    [signature setKeyValue:FSSignatureKeyNameLocation value:@"AnyWhere"];
    // Set reason.
    [signature setKeyValue:FSSignatureKeyNameReason value:@"AnyReason"];
    // Set contact info.
    [signature setKeyValue:FSSignatureKeyNameContactInfo value:@"AnyInfo"];
    // Set domain name.
    [signature setKeyValue:FSSignatureKeyNameDN value:@"AnyDN"];
    // Set description.
    [signature setKeyValue:FSSignatureKeyNameText value:@"AnyContent"];
    // Filter "Adobe.PPKLite" is supported by default.
    [signature setFilter:@"Adobe.PPKLite"];
    // SubFilter "adbe.pkcs7.sha1" or "adbe.pkcs7.detached" are supported by default.
    [signature setSubFilter:@"adbe.pkcs7.detached"];

    // The input PKCS#12 format certificate, which contains the public and private keys.
    NSString* certPath = @"/somewhere/cert.pfx";
    // Password for that certificate.
    NSString* certPassword = @"123";
    NSString* signedPDFPath = @"/somewhere/signed.pdf";
    // Start to sign the signature, if everything goes well, the signed PDF will be saved to the path specified by "save_path".
    FSProgressive* progress = [signature startSign:certPath cert_password:certPassword]
```

```
digest_algorithm:FSSignatureDigestSHA1 save_path:signedPDFPath client_data:nil pause:nil];
if(progress) {
    FSProgressiveState state = [progress resume];
    while(state == FSProgressiveToBeContinued)
        state = [progress resume];
    if(state != FSProgressiveFinished)
        return;
}

// Get the signatures from the signed PDF document, then verify them all.
FSPDFDoc* pdfDoc = [[FSPDFDoc alloc] initWithPath:signedPDFPath];
FSErrorCode err = [pdfDoc load:nil];
if(err != FSErrSuccess) return;
int count = [pdfDoc getSignatureCount];
for(int i=0; i<count; i++) {
    FSSignature* signature = [pdfDoc getSignature:i];
    if(signature) {
        FSProgressive *progress = [signature startVerify:nil pause:nil];
        if(progress != nil) {
            FSProgressiveState state = [progress resume];
            while (FSProgressiveToBeContinued == state) {
                state = [progress resume];
            }
            if(state != FSProgressiveFinished)
                continue;
        }
        int verifiedState = [signature getState];
        if(verifiedState & FSSignatureStateVerifyValid)
            NSLog(@"Signature %d is valid.", i);
    }
}
}
```

7 创建自定义工具

使用 Foxit PDF SDK for iOS 创建自定义工具非常简单。UI Extensions Component 中已经实现了一些工具，开发人员可以在这些工具的基础上进行二次开发，或者参考这些工具来创建新的工具。为了快速创建自己的工具，我们建议您参阅 "libs/uiextensions_src" 文件夹下的 **uiextensions** 工程。

创建一个新的自定义工具，最主要的是创建一个类，然后实现 "**IToolHandler**" 接口。

在本节中，我们通过创建一个区域屏幕截图工具来展示如何使用 Foxit PDF SDK for iOS 创建一个自定义工具。该工具帮助用户在 PDF 页面中选择一个区域进行截图，并将其保存为图片。现在，让我们开始吧。

7.1 使用 Objective-C 语言创建一个区域屏幕截图工具

为方便起见，我们将基于 "samples" 文件夹下的 "**viewer_ctrl_demo**" 工程来构建该工具。实现该工具的步骤如下：

- 创建名为 **ScreenCaptureToolHandler** 的类，该类实现 "**IToolHandler**" 接口。
- 处理 **onPageViewLongPress** 和 **onDraw** 事件。
- 实例化 **ScreenCaptureToolHandler** 对象，然后将其注册到 **UIExtensionsManager**。
- 将 **ScreenCaptureToolHandler** 对象设置为当前的 tool handler。

步骤 1: 创建一个名为 **ScreenCaptureToolHandler** 的类，该类实现 "**IToolHandler**" 接口。

- a) 在 Xcode 中打开 "**viewer_ctrl_demo**" 工程。在 "Source" 文件夹下创建名为 "**ScreenCaptureToolHandler**" 的类，以及创建其对应的头文件。
- b) **ScreenCaptureToolHandler** 类实现 **IToolHandler** 接口，如下所示：

```
@interface ScreenCaptureToolHandler : NSObject<IToolHandler>
```

步骤 2: 处理 **onPageViewLongPress** 和 **onDraw** 事件。

更新 **ScreenCaptureToolHandler.h**，如下所示：

```
#import <Foundation/Foundation.h>
#import <FoxitRDK/FSPDFViewControl.h>
#import <uiextensionsDynamic/UIExtensionsManager.h>
@protocol IToolHandler;
```

```
@class TaskServer;

@interface ScreenCaptureToolHandler : NSObject<IToolHandler>

- (instancetype)initWithUIExtensionsManager:(UIExtensionsManager*)extensionsManager
taskServer:(TaskServer*)taskServer;
@end
```

更新 ScreenCaptureToolHandler.m，如下所示：

```
#import "ScreenCaptureToolHandler.h"
#import <ImageIO/ImageIO.h>
#import <ImageIO/CGImageDestination.h>
#import <MobileCoreServices/UTCoreTypes.h>

@interface ScreenCaptureToolHandler ()  
  
@end  
  
@implementation ScreenCaptureToolHandler {
    UIExtensionsManager* _extensionsManager;
    FSPDFViewCtrl* _pdfViewCtrl;
    TaskServer* _taskServer;  
  
    CGPoint startPoint;
    CGPoint endPoint;  
  
}  
@synthesize type;  
  
- (instancetype)initWithUIExtensionsManager:(UIExtensionsManager*)extensionsManager
taskServer:(TaskServer*)taskServer
{
    self = [super init];
    if (self) {
        _extensionsManager = extensionsManager;
        _pdfViewCtrl = extensionsManager.pdfViewCtrl;
        _taskServer = taskServer;
    }
    return self;
}  
  
-(NSString*)getName
{
    return @"";  
}  
  
-(BOOL)isEnabled
{
    return YES;
}
```

```
- (void)onActivate
{
}

-(void)onDeactivate
{
}

// Save the image to a specified path.
- (void)saveJPGImage:(CGImageRef)imageRef path:(NSString *)path
{
    NSURL *fileURL = [NSURL fileURLWithPath:path];
    CGImageDestinationRef dr = CGImageDestinationCreateWithURL((__bridge CFURLRef)fileURL, kUTTypeJPEG , 1, NULL);

    CGImageDestinationAddImage(dr, imageRef, NULL);
    CGImageDestinationFinalize(dr);

    CFRelease(dr);
}

// Handle the PageView Gesture and Touch event
- (BOOL)onPageViewLongPress:(int)pageIndex recognizer:(UILongPressGestureRecognizer *)recognizer
{
    if (recognizer.state == UIGestureRecognizerStateBegan)
    {
        startPoint = [recognizer locationInView:[_pdfViewCtrl getPageView:pageIndex]];
        endPoint = startPoint;
    }
    else if (recognizer.state == UIGestureRecognizerStateChanged)
    {

        endPoint = [recognizer locationInView:[_pdfViewCtrl getPageView:pageIndex]];

        // Refresh the page view, then the onDraw event will be triggered.
        [_pdfViewCtrl refresh:pageIndex];
    }
    else if (recognizer.state == UIGestureRecognizerStateEnded || recognizer.state == UIGestureRecognizerStateCancelled)
    {
        // Get the size of the Rect.
        CGSize size = {fabs(endPoint.x-startPoint.x), fabs(endPoint.y-startPoint.y)};
        CGPoint origin = {startPoint.x<endPoint.x?startPoint.x:endPoint.x,
        startPoint.y<endPoint.y?startPoint.y:endPoint.y};

        // Get the Rect.
        CGRect rect = {origin, size};

        int newDibWidth = rect.size.width;
```

```
int newDibHeight = rect.size.height;
if (newDibWidth < 1 || newDibHeight < 1)
{
    return YES;
}

UIView* pageView = [_pdfViewCtrl getPageView:pageIndex];
CGRect bound = pageView.bounds;

// Create a bitmap with the size of the selected area.
int imgSize = newDibWidth*newDibHeight*4;
void* pBuff = malloc(imgSize);
NSData* buff = [NSData dataWithBytes:pBuff length:imgSize];

FSBitmap* fsbitmap = [[FSBitmap alloc] initWithWidth:newDibWidth height:newDibHeight
format:FSBitmapDIBArgb buffer: buff pitch:newDibWidth*4];
[fsbitmap fillRect:0xFFFFFFFF rect:nil];
FSRenderer* fsrenderer = [[FSRenderer alloc] initWithBitmap:fsbitmap is_rgb_order:YES];
FSPDFPage* page = [_pdfViewCtrl.currentDoc getPage:pageIndex];

// Calculate the display matrix.
FSMatrix2D* fsmatrix = [page getDisplayMatrix: -rect.origin.x top:-rect.origin.y width:bound.size.width
height:bound.size.height rotate:0];

// Set the render content, then start to render the selected area to the bitmap.
[fsrenderer setRenderContentFlags:FSRendererRenderPage | FSRendererRenderAnnot];
FSProgressive *progressive = [fsrenderer startRender:page matrix:fsmatrix pause:nil];
if (progressive) {
    while (true) {
        if ([progressive resume] != FSProgressiveToBeContinued) {
            break;
        }
    }
}

// Convert FSBitmap to CGImage.
CGDataProviderRef provider = CGDataProviderCreateWithData(NULL, buff.bytes, imgSize, nil);
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
CGBitmapInfo bitmapInfo = kCGBitmapByteOrderDefault | kCGImageAlphaLast;

CGImageRef image = CGImageCreate(newDibWidth,newDibHeight, 8, 32, newDibWidth * 4,
                                colorSpace, bitmapInfo,
                                provider, NULL, YES, kCGRenderingIntentDefault);

// Save the image to a specified path.
NSString* jpgPath = @"/Users/Foxit/Desktop/ScreenCapture.jpg";
[self saveJPGImage:image path:jpgPath];

UIAlertView *alert = [[UIAlertView alloc]initWithTitle:@""
                                         message:@" The selected area was saved as a JPG stored in the
/Users/Foxit/Desktop/ScreenCapture.jpg" delegate:nil cancelButtonTitle: NSLocalizedString(@"OK", @"OK")]
```

```
otherButtonTitles:nil];
[alert show];

    return YES;
}
return YES;
}

// Handle the drawing event.
-(void)onDraw:(int)pageIndex inContext:(CGContextRef)context
{
    if (_extensionsManager.currentToolHandler != self) {
        return;
    }

    CGContextSetLineWidth(context, 2);
    CGContextSetLineCap(context, kCGLineCapSquare);
    UIColor *color = [UIColor redColor];
    CGContextSetStrokeColorWithColor(context, [color CGColor]);
    CGPoint points[] = {startPoint,CGPointMake(endPoint.x, startPoint.y),endPoint,CGPointMake(startPoint.x, endPoint.y)};
    CGContextAddLines(context,points,4);
    CGContextClosePath(context);
    CGContextStrokePath(context);
}

-(BOOL)onPageViewTap:(int)pageIndex recognizer:(UITapGestureRecognizer *)recognizer
{
    return NO;
}

-(BOOL)onPageViewPan:(int)pageIndex recognizer:(UIPanGestureRecognizer *)recognizer
{
    return NO;
}

-(BOOL)onPageViewShouldBegin:(int)pageIndex recognizer:(UIGestureRecognizer *)gestureRecognizer
{
    if (_extensionsManager.currentToolHandler != self) {
        return NO;
    }
    return YES;
}

-(BOOL)onPageViewTouchesBegan:(int)pageIndex touches:(NSSet *)touches withEvent:(UIEvent *)event
{
    return NO;
}

-(BOOL)onPageViewTouchesMoved:(int)pageIndex touches:(NSSet *)touches withEvent:(UIEvent *)event
```

```
{  
    return NO;  
}  
  
- (BOOL)onPageViewTouchesEnded:(int)pageIndex touches:(NSSet *)touches withEvent:(UIEvent *)event  
{  
    return NO;  
}  
  
- (BOOL)onPageViewTouchesCancelled:(int)pageIndex touches:(NSSet *)touches withEvent:(UIEvent *)event  
{  
    return NO;  
}  
  
@end
```

备注: 在上述的代码中，您需要指定一个存在的路径来保存截图的图片。这里，路径是 "@"/Users/Foxit/Desktop/ScreenCapture.jpg""，请使用一个有效的路径替换它。

步骤 3: 在 ViewController.m 中，实例化一个 **ScreenCaptureToolHandler** 对象，然后将其注册到 UIExtensionsManager。

```
#import "ScreenCaptureToolHandler.h"  
...  
  
@property (nonatomic, strong) ScreenCaptureToolHandler* screenCaptureToolHandler;  
...  
  
self.screenCaptureToolHandler = [[ScreenCaptureToolHandler alloc] initWithUIExtensionsManager:  
self.extensionsManager taskServer:nil];  
[self.extensionsManager registerToolHandler:self.screenCaptureToolHandler];
```

步骤 4: 在 ViewController.m 中，将 **ScreenCaptureToolHandler** 对象设置为当前的 tool handler。

注册 Doc 监听事件：

```
@interface ViewController () <ISearchEventListener, UIExtensionsManagerDelegate, IDocEventListener>  
...  
  
[self.pdfViewCtrl registerDocEventListener:self];
```

在 onDocOpened 函数中设置当前的 tool handler：

```
- (void)onDocOpened:(FSPDFDoc *)document error:(int)error {  
    [self.extensionsManager setCurrentToolHandler:self.screenCaptureToolHandler];  
}
```

现在，我们已经使用 Objective-C 完成了自定义工具的创建。然后，使用 iPhone 11 模拟器来编译和运行该 demo。当成功编译该 demo 后，长按并选择一个矩形区域，则会弹出如 Figure 7-1 所示的消息框。该消息框提示了所选区域被保存的位置。



Figure 7-1

为了验证该工具是否成功截取和保存了所选区域，我们需要找到保存的屏幕截图。在桌面上，可以看到如 Figure 7-2 所示的图片。



Figure 7-2

如您所见，我们已成功使用 Objective-C 创建了区域屏幕截图工具。这只是一个示例，用来说明如何使用 Foxit PDF SDK for iOS 创建自定义工具。您可以参考该示例或者我们的 demos 来开发您需要的工具。

7.2 使用 Swift 语言创建一个区域屏幕截图工具

为方便起见，我们将基于 "samples\swift" 文件夹下的 "**viewer_ctrl_demo_swift**" 工程来构建该工具。实现该工具的步骤如下：

- 创建名为 **ScreenCaptureToolHandler** 的类，该类实现 "**IToolHandler**" 接口。
- 处理 **onPageViewLongPress** 和 **onDraw** 事件。
- 实例化 **ScreenCaptureToolHandler** 对象，然后将其注册到 **UIExtensionsManager**。
- 将 **ScreenCaptureToolHandler** 对象设置为当前的 tool handler。

步骤 1: 创建一个名为 **ScreenCaptureToolHandler** 的类，该类实现 "**IToolHandler**" 接口。

- a) 在 Xcode 中打开 "**viewer_ctrl_demo_swift**" 工程。在 "Source" 文件夹下创建名为 "**ScreenCaptureToolHandler**" 的类。
- b) **ScreenCaptureToolHandler** 类实现 **IToolHandler** 接口，如下所示：

```
class ScreenCaptureToolHandler: NSObject, IToolHandler { }
```

步骤 2: 处理 **onPageViewLongPress** 和 **onDraw** 事件。

更新 ScreenCaptureToolHandler.swift，如下所示：

```
import Foundation
import MobileCoreServices
import ImageIO

class ScreenCaptureToolHandler: NSObject, IToolHandler {
    public var type: FSAnnotType

    var extensionManager: UIExtensionsManager!
    var pdfViewCtrl: FSPDFViewCtrl!

    var startPoint = CGPoint()
    var endPoint = CGPoint()

    init(extensionsManager: UIExtensionsManager) {
        self.extensionManager = extensionsManager
        self.pdfViewCtrl = extensionsManager.pdfViewCtrl
        self.type = FSAnnotType.annotUnknownType
        super.init()
    }

    func getName() -> String {
        return ""
    }

    func isEnabled() -> Bool {
        return true
    }

    func onActivate() {
    }

    func onDeactivate() {
    }

    // Save the image to a specified path.
    func saveJPGImage(imageRef: CGImage, path: String) {
        let fileURL: CFURL = NSURL.fileURL(withPath: path) as CFURL
        let dr = CGImageDestinationCreateWithURL(fileURL, kUTTypeJPEG, 1, nil)!
        CGImageDestinationAddImage(dr, imageRef, nil)
        CGImageDestinationFinalize(dr)
    }

    // Handle the PageView Gesture and Touch event
    func onPageViewLongPress(_ pageIndex: Int32, recognizer: UILongPressGestureRecognizer) -> Bool {

        if recognizer.state == UIGestureRecognizerState.began {
            startPoint = recognizer.location(in: pdfViewCtrl.getPageView(pageIndex))
            endPoint = startPoint
        }
    }
}
```

```
}

else if recognizer.state == UIGestureRecognizerState.changed {

    endPoint = recognizer.location(in: pdfViewCtrl.getPageView(pageIndex))

    // Refresh the page view, then the onDraw event will be triggered.
    pdfViewCtrl.refresh(pageIndex)
}

else if recognizer.state == UIGestureRecognizerState.ended || recognizer.state ==
UIGestureRecognizerState.cancelled {

    // Get the size of the Rect.
    let size = CGSize(width: fabs(endPoint.x - startPoint.x), height: fabs(endPoint.y - startPoint.y))
    let origin = CGPoint(x: (startPoint.x < endPoint.x) ? startPoint.x : endPoint.x, y: (startPoint.y < endPoint.y) ?
startPoint.y : endPoint.y)
    // Get the Rect.
    let rect = CGRect(origin: origin, size: size)

    let newDibwidth = rect.size.width
    let newDibHeight = rect.size.height
    if newDibwidth < 1 || newDibHeight < 1 {
        return true
    }

    let pageView = pdfViewCtrl.getPageView(pageIndex)
    let bound = pageView.bounds

    // Create a bitmap with the size of the selected area.
    let imgSize = newDibwidth * newDibHeight * 4
    let capacity: Int = Int(newDibwidth) * Int(newDibHeight) * 4

    let buff = UnsafeMutablePointer<UInt8>.allocate(capacity: capacity)
    let pBuff = NSData.init(bytes: UnsafeRawPointer(buff), length: capacity)

    let pitch: Int = Int(newDibwidth) * 4
    guard let fsbitmap = FSBitmap.init(width: Int32(newDibwidth), height: Int32(newDibHeight),
format:FSBitmapDIBFormat.dibArgb , buffer: pBuff as Data, pitch: Int32(pitch)) else {
        return false
    }
    fsbitmap.fillRect(0xFFFFFFFF, rect: nil)
    let fsrenderer = FSRenderer.init(bitmap: fsbitmap, is_rgb_order: true)
    let page = pdfViewCtrl.currentDoc?.getPage(pageIndex)

    // Calculate the display matrix.
    let fsmatrix = page?.getDisplayMatrix(-Int32(rect.origin.x), top: -Int32(rect.origin.y), width:
Int32(bound.size.width), height: Int32(bound.size.height), rotate: FSRotation.rotation0)

    // Set the render content, then start to render the selected area to the bitmap.
    fsrenderer?.setRenderContentFlags(UInt32(UInt8(FSRendererContentFlag.renderPage.rawValue) |
UInt8(FSRendererContentFlag.renderAnnot.rawValue)))
}
```

```
let progress = fsrenderer?.startRender(page, matrix: fsmatrix, pause: nil)
if ((progress) != nil) {

    while (true) {
        if (progress?.resume() != FSPergressiveState.toBeContinued)
        {
            break
        }
    }
}

// Convert FSBitmap to CGImage.
let releaseData: CGDataProviderReleaseDataCallback = {
    (info: UnsafeMutableRawPointer?, data:UnsafeRawPointer, size:Int) -> Void in
}

let provider: CGDataProvider = CGDataProvider.init(dataInfo: nil, data: pBuff.bytes, size: Int(imgSize),
releaseData: releaseData)!
let colorSpace = CGColorSpaceCreateDeviceRGB()
let bitmapInfo: CGBitmapInfo = .byteOrderMask

let image = CGImage(width: Int(newDibwidth), height: Int(newDibHeight), bitsPerComponent: 8,
bitsPerPixel: 32, bytesPerRow: Int(newDibwidth) * 4, space: colorSpace, bitmapInfo: bitmapInfo, provider:
provider, decode: nil, shouldInterpolate: true, intent: CGColorRenderingIntent.defaultIntent)

// Save the image to a specified path.
let jpgPath = "/Users/Foxit/Desktop/ScreenCapture.jpg"
self.saveJPGImage(image!, path: jpgPath)

let alert = UIAlertView(title: "", message: "The selected area was saved as a JPG stored in the
/Users/Foxit/Desktop/ScreenCapture.jpg", delegate: nil, cancelButtonTitle: NSLocalizedString("OK", comment:
"OK"))
alert.show()
return true
}
return true
}

// Handle the drawing event.
func onDraw(_ pageIndex: Int32, in context: CGContext) {
    context.setLineWidth(CGFloat(2))
    context.setLineCap(.square)
    let color = UIColor.red
    context.setStrokeColor(color.cgColor)
    let points = [startPoint, CGPoint(x: CGFloat(endPoint.x), y: CGFloat(startPoint.y)), endPoint, CGPoint(x:
    CGFloat(startPoint.x), y: CGFloat(endPoint.y))]
    context.addLines(between: points)
    context.closePath()
    context.strokePath()
}
```

```
func onPageViewTap(_ pageIndex: Int32, recognizer: UITapGestureRecognizer?) -> Bool {
    return false
}

func onPageViewPan(_ pageIndex: Int32, recognizer: UIPanGestureRecognizer) -> Bool {
    return false
}

func onPageViewShouldBegin(_ pageIndex: Int32, recognizer gestureRecognizer: UIGestureRecognizer) ->
Bool {
    return true
}

func onPageViewTouchesBegan(_ pageIndex: Int32, touches: Set<AnyHashable>, with event: UIEvent) -> Bool
{
    return false
}

func onPageViewTouchesMoved(_ pageIndex: Int32, touches: Set<AnyHashable>, with event: UIEvent) -> Bool
{
    return false
}

func onPageViewTouchesEnded(_ pageIndex: Int32, touches: Set<AnyHashable>, with event: UIEvent) -> Bool
{
    return false
}

func onPageViewTouchesCancelled(_ pageIndex: Int32, touches: Set<AnyHashable>, with event: UIEvent) ->
Bool {
    return false
}
```

备注: 在上述的代码中，您需要指定一个存在的路径来保存截图的图片。这里，路径是 "/Users/Foxit/Desktop/ScreenCapture.jpg"，请使用一个有效的路径替换它。

步骤 3: 在 ViewController.swift 中，实例化一个 **ScreenCaptureToolHandler** 对象，然后将其注册到 UIExtensionsManager。

```
var screenCaptureToolHandler: ScreenCaptureToolHandler!
...
self.screenCaptureToolHandler = ScreenCaptureToolHandler.init(extensionsManager: self.extensionsManager)
self.extensionsManager.register(self.screenCaptureToolHandler)
```

步骤 4: 在 ViewController.swift 中，将 **ScreenCaptureToolHandler** 对象设置为当前的 tool handler。

注册 Doc 监听事件：

```
class ViewController: UIViewController, UISearchBarDelegate, ISearchEventListener, IDocEventListener  
...  
self.pdfViewCtrl.register(self)
```

在 onDocOpened 函数中设置当前的 tool handler：

```
func onDocOpened(_ document: FSPDFDoc?, error: Int32) {  
    self.extensionsManager.currentToolHandler = self.screenCaptureToolHandler  
}
```

现在，我们已经使用 Swift 完成了自定义工具的创建。然后编译和运行该 demo。当成功编译后，长按并选择一个矩形区域，则会弹出如 Figure 7-1 所示的消息框。在桌面上，可以看到如 Figure 7-2 所示的图片。

这只是一个示例，用来说明如何使用 Foxit PDF SDK for iOS 和 Swift 语言创建自定义工具。您可以参考该示例或者我们的 demos 来开发您需要的工具。

8 使用 Cordova 实现 Foxit PDF SDK for iOS

在开发跨平台移动应用程序时，Apache Cordova 是一个理想的开源框架。'cordova-plugin-foxitpdf' 是我们提供的使用 Foxit PDF SDK for iOS 的移动框架插件之一。该插件帮助您可以使用 Cordova 框架实现强大的 PDF viewing 功能。通过此插件，您可以预览任何 PDF 文件，包括 PDF 2.0 标准的文件，XFA 文档和受 RMS 加密的文档，您还可以注释和编辑 PDF 文档。

对于 'cordova-plugin-foxitpdf' 插件的用法，请参阅网站

<https://github.com/foxitsoftware/cordova-plugin-foxitpdf>。

9 使用 React Native 实现 Foxit PDF SDK for iOS

React Native 是一个使用 JavaScript 和 React 构建 native 应用程序的开源移动开发框架。'react-native-foxitpdf'是我们提供的使用 Foxit PDF SDK for iOS 的移动框架插件之一。该插件帮助您可以使⽤ React Native 框架实现强大的 PDF viewing 功能。通过此插件，您可以预览任何 PDF 文件，包括 PDF 2.0 标准的文件，XFA 文档和受 RMS 加密的文档，您还可以注释和编辑 PDF 文档。

对于 'react-native-foxitpdf' 插件的用法，请参阅网站 <https://github.com/foxitsoftware/react-native-foxitpdf>。

10 使用 Xamarin 实现 Foxit PDF SDK for iOS

Xamarin 是一个使用共享 C# 代码库构建 native 应用程序的跨平台开发框架。我们为 Android 和 iOS 平台分别提供了对应的 bindings for Android and iOS ('foxit_xamarin_android' 和 'foxit_xamarin_ios')，以便开发人员可以将 Foxit PDF SDK 强大的 PDF 功能无缝地集成到他们的 Xamarin 应用程序中。

对于 'foxit_xamarin_ios' 插件的用法，请参阅网站 <https://github.com/foxitsoftware/xamarin-foxitpdf>。

11 FAQ

11.1 Bitcode 支持

什么是 Bitcode? Foxit PDF SDK for iOS 是否支持 Bitcode?

Bitcode 是被编译程序的一种中间形式的代码。Bitcode 允许 Apple 在后期重新优化应用程序的二进制文件，而不需要向 App Store 重新提交一个新的版本。

是的。Foxit PDF SDK for iOS 从 3.0 版本开始支持 Bitcode。

11.2 从指定的 PDF 文件路径打开一个 PDF 文档

如何从指定的 PDF 文件路径打开一个 PDF 文档?

Foxit PDF SDK for iOS 提供了多个接口用来打开 PDF 文档。您可以从指定的 PDF 文件路径或从内存缓冲区打开一个 PDF 文档。对于指定的 PDF 文件路径，有两种方法可以使用。

第一种是使用 **openDoc** 接口，该接口包括以下的操作：创建 PDF 文档对象(**initWithPath**)，加载文档内容(**load**)，以及将 PDF 文档对象设置给视图控件(**setDoc**)。以下是示例代码：

备注：**openDoc** 接口仅可用于从文件路径打开 PDF 文档。如果需要自定义加载 PDF 文档，可以在回调函数(**FSFileReadCallback**) 中实现，然后使用带有回调函数 **FireRead** 的 **initWithHandler** 接口创建文档对象。接下来，使用 **load** 加载文档内容，并使用 **setDoc** 将 PDF 文档对象设置给视图控件。

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>

@interface ViewController : UIViewController

@end

@implementation ViewController
{
    FSPDFViewCtrl* pdfViewCtrl;
}

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF.
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];
}
```

```
// Initialize a FSPDFViewCtrl object with the size of the entire screen.  
pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame: [self.view bounds]];  
  
// Open an unencrypted PDF document from a specified PDF file path.  
[pdfViewCtrl openDoc:pdfPath password:nil completion:nil];  
  
// Add the pdfView to the root view.  
[self.view addSubview:pdfViewCtrl];  
}  
@end
```

第二种是使用 **initWithPath** 接口创建 PDF 文档对象，使用 **load** 接口加载文档内容，然后使用 **setDoc** 将 PDF 文档对象设置给视图控件。以下是示例代码：

```
#import "ViewController.h"  
#import <FoxitRDK/FSPDFViewControl.h>  
  
@interface ViewController : UIViewController  
  
@end  
  
@implementation ViewController  
{  
  
    FSPDFViewCtrl* pdfViewCtrl;  
}  
  
- (void)viewDidLoad {  
    [super viewDidLoad];  
  
    // Get the path of a PDF.  
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];  
  
    // Initialize a PDFDoc object with the path to the PDF file.  
    FSPDFDoc* pdfdoc = [[FSPDFDoc alloc] initWithPath: pdfPath];  
  
    // Load the unencrypted document content.  
    if(FSErrSuccess != [pdfdoc load:nil]) {  
        return;  
    }  
  
    // Initialize a FSPDFViewCtrl object with the size of the entire screen.  
    pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame: [self.view bounds]];  
  
    // Set the document to view control.  
    [pdfViewCtrl setDoc:pdfdoc];  
  
    // Add the pdfView to the root view.  
    [self.view addSubview:pdfViewCtrl];
```

```
}
```

11.3 打开 PDF 文档时显示指定的页面

如何在打开 PDF 文档时，显示指定的页面？

为了在打开 PDF 文档时显示指定的页面，您需要使用接口 [**pdfViewCtrl gotoPage: (int) animated: (BOOL)**]。Foxit PDF SDK for iOS 使用多线程来提高渲染速度，因此您需要确保在使用 **gotoPage** 接口之前，文档已经被成功加载。有两种方法可以实现该功能。

第一种是在 **openDoc** 接口中创建条件语句，以确保仅在文档加载完成时，再调用 **gotoPage**。如果文档没有加载完成，**gotoPage** 接口是不会起作用的，那么打开文档时将显示第一页。这是因为 **gotoPage** 接口启动了一个新的线程来执行操作。以下是示例代码：

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>

@interface ViewController : UIViewController

@end

@implementation ViewController
{

    FSPDFViewCtrl* pdfViewCtrl;
}

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF.
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];

    // Initialize a FSPDFViewCtrl object with the size of the entire screen.
    pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame: [self.view bounds]];

    // Open an unencrypted PDF document from a specified PDF file path, and go to the third page when
    // completing the document loading.
    [pdfViewCtrl openDoc:pdfPath password:nil completion:^(enum FSErrorCode error){

        if(error == FSErrSuccess)

            // Display the third page.
            [pdfViewCtrl gotoPage:2 animated:NO];
    }];

    // Add the pdfView to the root view.
    [self.view addSubview:pdfViewCtrl];
}
```

```
}
```

```
@end
```

第二种是实现<IDocEventListener> 协议，然后在 **onDocOpened** 事件中调用 **gotoPage** 接口。以下是示例代码：

```
#import "ViewController.h"
#import <FoxitRDK/FSPDFViewControl.h>

@interface ViewController : UIViewController <IDocEventListener>

@end

@implementation ViewController
{
    FSPDFViewCtrl* pdfViewCtrl;
}

- (void)viewDidLoad {
    [super viewDidLoad];

    // Get the path of a PDF
    NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];

    // Initialize a FSPDFViewCtrl object with the size of the entire screen.
    pdfViewCtrl = [[FSPDFViewCtrl alloc] initWithFrame:[self.view bounds]];

    // Register the PDF document event listener.
    [pdfViewCtrl registerDocEventListener:self];

    // Open an unencrypted PDF document from a specified PDF file path.
    [pdfViewCtrl openDoc:pdfPath password:nil completion:nil];

    // Add the pdfView to the root view.
    [self.view addSubview:pdfViewCtrl];
}

#pragma IDocEventListener

-(void)onDocOpened:(FSPDFDoc *)document error:(int)error
{
    // display the third page.
    [pdfViewCtrl gotoPage:2 animated:NO];
}
@end
```

11.4 License key 和序列号无法正常工作

从网站下载的 SDK 包，未进行任何更改，为什么 license key 和序列号无法正常工作？

通常，上传到网站的包，里面的 license key 和序列号是可以正常工作的。在上传到网站之前是经过测试的。因此，如果您发现 license key 和序列号无法使用，则可能是由设备的日期引起的。如果您设备的时间在下载包 "libs" 文件夹下 **rdk_key.txt** 文件中的 StartDate 之前，则 "librdk.so" 库将无法解锁。请检查您设备的日期。

11.5 在 PDF 文档中添加 link 注释

如何在 PDF 文档中添加 link 注释？

为了将 link 注释添加到 PDF 文档，首先需要调用 **FSPDFPage::addAnnot** 将一个 link 注释添加到指定页面，然后调用 **FSAction::Create** 创建一个 action，并将该 action 设置给刚添加的 link 注释。以下是在 PDF 首页添加一个 URI link 注释的示例代码：

```
#define DOCUMENT_PATH [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,  
NSUserDomainMask, YES) objectAtIndex:0]  
  
...  
  
// Get the path of a PDF  
NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];  
  
// Initialize a PDFDoc object with the path to the PDF file.  
FSPDFDoc *document = [[FSPDFDoc alloc] initWithPath: pdfPath];  
  
// load the unencrypted document content.  
[document load:nil];  
  
// Get the first page of the PDF file.  
FSPDFPage *page = [document getPage:0];  
  
// Add a link annotation to the first page.  
FSRectF *rect = [[FSRectF alloc] initWithLeft1:250 bottom1:650 right1:450 top1:750];  
FSLink *linkAnnot = [[FSLink alloc] initWithAnnot:[page addAnnot: FSAnnotLink rect:rect]];  
  
// Create a URI action and set the URI.  
FSURIAction *uriAction = [[FSURIAction alloc] initWithAction:[FSAction create:document  
action_type:FSActionTypeURI]];  
[uriAction setURI:@"https://www.foxitsoftware.com"];  
  
// Set the action to link annotation.  
[linkAnnot setAction:uriAction];  
  
// Reset appearance stream.  
[linkAnnot resetAppearanceStream];
```

```
// Save the document that has added the link annotation.  
[document saveAs:[DOCUMENT_PATH stringByAppendingPathComponent:@"Sample_annot.pdf"]  
save_flags:FSPDFDocSaveFlagNormal];
```

11.6 向 PDF 文档中插入图片

如何向 PDF 文档中插入图片？

向 PDF 文档中插入图片，可以调用 **FSPDFPage::addImageFromFilePath** 接口。以下是将图片插入到 PDF 文档首页的示例代码：

备注：在调用**FSPDFPage::addImageFromFilePath** 接口之前，您需要获取并解析将要添加图片的页面。

```
#define DOCUMENT_PATH [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,  
NSUserDomainMask, YES) objectAtIndex:0]  
  
...  
  
// Get the path of a PDF.  
NSString* pdfPath = [[NSBundle mainBundle] pathForResource:@"Sample" ofType:@"pdf"];  
  
// Initialize a PDFDoc object with the path to the PDF file.  
FSPDFDoc *document = [[FSPDFDoc alloc] initWithPath: pdfPath];  
  
// load the unencrypted document content.  
[document load:nil];  
  
// Get the first page of the PDF file.  
FSPDFPage *page = [document getPage:0];  
  
// Parse the page.  
if (!([page isParsed])) {  
    FSProgressive* progressive = [page startParse: FSPDFPageParsePageNormal pause:nil is_reparse:NO];  
    while ([progressive resume] == FSProgressiveToBeContinued) {  
        continue;  
    }  
}  
  
// Get the image path.  
NSString* imagePath = @"/Users/xiaole/Desktop/1.png";  
  
// Add an image to the first page.  
FSPointF* point = [[FSPointF alloc] init];  
[point set:100 y:300];  
[page addImageFromFilePath:imagePath position:point width:100 height:120 auto_generate_content:YES];
```

```
// Save the document that has added the link annotation.  
[document saveAs:[DOCUMENT_PATH stringByAppendingPathComponent:@"Sample_image.pdf"]  
save_flags:FSPDFDocSaveFlagNormal];
```

11.7 高亮 PDF 文档中的 links 和设置高亮颜色

如何设置是否高亮 PDF 文档中的 links? 以及如何设置高亮的颜色?

默认情况下，高亮 PDF 文档中的 links 功能是启用的。如果您想要禁用它或者设置高亮颜色，可以通过 JSON 文件(仅支持 6.3 及以上版本)或调用 API 进行设置。

备注: 如果您需要设置高亮的颜色，请确保 links 高亮的功能是启用的。

通过 JSON 文件

设置 `"highlightLink": false`, 在 PDF 文档中禁用 links 高亮功能。

设置 `"highlightLinkColor": "#16007000"`, 设置高亮颜色(输入您需要的颜色值)。

通过调用 API

UIExtensionsManager.enableHighlightLinks 属性用来设置是否在 PDF 文档中启用 links 高亮的功能。如果您不需要启用该功能，请将其参数设置为 "false"，如下所示：

```
// Assume you have already Initialized a UIExtensionsManager object  
extensionsManager.enableHighlightLinks = false;
```

UIExtensionsManager.linksHighlightColor 属性用来设置高亮的颜色。以下是设置该属性的示例代码：

```
// Assume you have already Initialized a UIExtensionsManager object  
extensionsManager.linksHighlightColor = [UIColor colorWithRed:0 green:0 blue:1 alpha:0.3];
```

11.8 高亮 PDF 文档中的表单域和设置高亮颜色

如何设置是否高亮 PDF 文档中的表单域? 以及如何设置高亮的颜色?

默认情况下，高亮 PDF 文档中的表单域功能是启用的。如果您需要设置高亮的颜色，请确保表单域高亮的功能是启用的。

从 6.3 版本开始，您可以通过 JSON 配置文件轻松地启用/禁用表单域高亮功能，以及设置高亮的颜色，只需要设置如下的两个配置项：

```
"highlightForm": true,  
"highlightFormColor": "#200033cc",
```

对于 6.3 之前的版本，如果您需要启用/禁用表单域高亮功能，或者设置高亮的颜色，您需要使用 **FSFormFiller::highlightFormFields** 和 **FSFormFiller::setHighlightColor** 接口修改 UI Extensions 组件的源代码。

FSFormFiller::highlightFormFields 接口用来设置是否启用 PDF 文档中的表单域高亮功能。请参考 4.3 小节 "[通过源代码自定义 UI 实现](#)" 将 "libs/uiextensions_src" 文件夹下的 "uiextensions" 工程添加到您的工程中。然后，在 "UIExtensions/Form/FormAnnotHandler.m" 文件中找到 **onDocOpened** 函数，将 **FSFormFiller::highlightFormFields** 的参数设置为 "false"，如下所示：

```
- (void)onDocOpened:(FSPDFDoc*)document error:(int)error
{
    ...
    ...
    [_formFiller highlightFormFields:false];
}
```

FSFormFiller::setHighlightColor 接口用来设置高亮的颜色。以下是调用此 API 的示例代码：

```
- (void)onDocOpened:(FSPDFDoc*)document error:(int)error
{
    ...
    ...
    [_formFiller highlightFormFields:true];
    [_formFiller setHighlightColor:0x4b00ff00];
}
```

11.9 支持全文索引搜索

Foxit PDF SDK for iOS 是否支持全文索引搜索？如果支持，如何搜索在我的移动设备上离线存储的 PDF 文件？

是的。Foxit PDF SDK for iOS 支持全文索引搜索。

要使用此功能，请按照如下的步骤：

a) 根据目录来创建一个文档来源，该目录为文档的搜索目录。

```
-(id)initWithDirectory: (NSString *)directory;
```

b) 创建一个全文文本搜索对象，以及设置用于存储索引数据的数据库路径。

```
-(id)init;
-(void)setDataBasePath:(NSString *)path_of_data_base;
```

c) 开始索引文档来源中的 PDF 文档。

```
- (FSProgressive*)startUpdateIndex: (FSDocumentsSource*)source pause: (id<FSPauseCallback>)pause
reUpdate:(BOOL)reUpdate;
```

备注：您可以索引指定的PDF文件。例如，如果某个PDF文档的内容发生了更改，您可以使用以下的API对其进行重新进行索引：

```
- (BOOL)updateIndexWithFilePath: (NSString *)file_path;
```

- d) 从索引数据源中搜索指定的内容。搜索的结果将通过指定的回调函数来返回给外部，每找到一个匹配结果，则调用一次回调函数。

```
- (BOOL)searchOf: (NSString *)match_string rank_mode:(FSFullTextSearchRankMode)rank_mode callback:
(id<FSSearchCallback>)Callback;
```

如下是使用全文索引搜索的示例代码：

```
- (void)FullTextSearch {
    NSString *directory = @"INPUT_DIRECTORY";
    FSDocumentsSource* docs = [[FSDocumentsSource alloc] initWithDirectory:directory];
    FSFullTextSearch* fulltextSearch = [FSFullTextSearch init];

    NSString* dbPath = @"The path of data base to store the indexed data...";
    [fulltextSearch set DataBasePath:dbPath];

    FSProgressive* progressive = [fulltextSearch startUpdateIndex:docs pause:nil reUpdate:NO];
    if (progressive) {
        while (true) {
            if ([progressive resume] != FSProgressiveToBeContinued) {
                break;
            }
        }
    }

    [fulltextSearch searchOf:@[@"Foxit" RankMode:FSFullTextSearchRankNone callback:[[FSSearchCallbackImp
alloc] init]]];
}

@end
```

回调函数的示例代码如下所示：

```
@interface FSSearchCallbackImp: NSObject<FSSearchCallback>

@end

@implementation FSSearchCallbackImp

-(int)retrieveSearchResult:(NSString*)file_path page_index:(int)page_index
match_result:(NSString*)match_result match_start_text_index:(int)match_start_text_index
```

```

match_end_text_index:(int)match_end_text_index
{
    NSLog(@"file_path: %@", file_path);
    NSLog(@"page_index: %i, match_start_text_index: %i, match_end_text_index: %i\n", page_index,
match_start_text_index, match_end_text_index);
    NSLog(@"match_result: %@\n\n", match_result);
    return 0;
}

@end

```

Note:

- Foxit PDF SDK for Android 提供的全文索引搜索将以递归的方式遍历整个目录，以便搜索目录下的文件和文件夹都会被索引。
- 如果需要中止索引进程，可以将 `pause` 回调参数传递给 `startUpdateIndex` 接口。其回调函数 `NeedPauseNow` 都会被调用，一旦完成一个 PDF 文档的索引。因此，调用者可以在回调函数 `NeedPauseNow` 返回 "true" 时中止索引进程。
- 索引数据库的位置由 `setDataBasePath` 接口设置。如果要清除索引数据库，请手动清除。目前，不支持从索引函数中删除指定文件相关的索引内容。
- `searchOf` 接口的每个搜索结果都由指定的回调返回到外部。一旦 `searchOf` 接口返回"true" 或 "false"，则表示搜索已完成。

11.10 打印 PDF 文档

Foxit PDF SDK for iOS 是否支持打印 PDF 文档？如果支持，如何使用？

是的。Foxit PDF SDK for iOS 从 5.1 版本开始支持打印 PDF 文档的功能。您可以在 Complete PDF viewer demo 的 More Menu View 菜单中点击 Wireless Print 按钮来打印 PDF 文档。此外，您可以调用以下 API 来打印 PDF 文档：

```

// for iPhone and iTouch
(void)printDoc:(FSPDFDoc *)doc animated:(BOOL)animated jobName:(nullable NSString *)jobName
delegate:(nullable id<UIPrintInteractionControllerDelegate>)delegate completionHandler:(nullable
UIPrintInteractionCompletionHandler)completion;

// for iPad
(void)printDoc:(FSPDFDoc *)doc fromRect:(CGRect)rect inView:(UIView *)view animated:(BOOL)animated
jobName:(nullable NSString *)jobName delegate:(nullable id<UIPrintInteractionControllerDelegate>)delegate
completionHandler:(nullable UIPrintInteractionCompletionHandler)completion;

```

使用 PDF 打印功能的示例代码：

```
UIPrintInteractionCompletionHandler completion = ^(UIPrintInteractionController *_Nonnull
```

```

printInteractionController, BOOL completed, NSError *_Nullable error) {
    if (error) {
        UIAlertAction* action = [UIAlertAction actionWithTitle:@"Warning:"
stringByAppendingString:error.localizedDescription] style:UIAlertActionStyleDefault handler:nil];
        UIAlertController* controller = [[UIAlertController alloc] init];
        [controller addAction:action];

        [self showViewController:controller sender:nil];
    }
};

NSString *fileName = @"xxx.pdf";
FSPDFDoc* doc = [[FSPDFDoc alloc] initWithPath:fileName];
[UIExtensionsManager printDoc: doc animated:YES jobName:fileName delegate:nil
completionHandler:completion];

```

11.11 夜间模式颜色设置

如何设置夜间模式颜色？

设置夜间模式颜色，需要首先设置 `FSPDFViewCtrl.mappingModeBackgroundColor` 和 `FSPDFViewCtrl.mappingModeForegroundColor` 属性，然后将 `FSPDFViewCtrl.colorMode` 设置为 `FSRendererColorModeMapping`。

备注：如果 `FSPDFViewCtrl.colorMode` 已经设置为 `FSRendererColorModeMapping`，在更新 `FSPDFViewCtrl.mappingModeBackgroundColor` 和 `FSPDFViewCtrl.mappingModeForegroundColor` 属性之后，您仍然需要再次设置。否则，颜色设置可能不会起作用。

上述的属性需要在 UI Extensions 组件的源代码中修改，请参考 5.3 小节 "[通过源代码自定义 UI 实现](#)" 将 "libs/uiextensions_src" 文件夹下的 "uiextensions" 工程添加到您的工程中。然后在 "UIExtensions/UIExtensionsManager.m" 文件中定位到 `settingBar` 函数，根据您的需要修改属性的颜色。

设置夜间模式颜色的示例代码：

```

- (void)settingBar:(FSSettingBar *)settingBar isNightMode:(BOOL)isNightMode {

    self.settingBar.viewSettingManager.isDayOrNightForLast = YES;
    if ([self.pdfViewCtrl getPageLayoutMode] == PDF_LAYOUT_MODE_REFLOW) {
        UIColor* color = isNightMode ? UIColor.blackColor : UIColor.whiteColor;
        [self.pdfViewCtrl setReflowBackgroundColor:color];
        [self.pdfViewCtrl setPageLayoutMode:PDF_LAYOUT_MODE_REFLOW];
        self.pdfViewCtrl.isNightMode = isNightMode;
    }else{
        if (!(self.pdfViewCtrl.isNightMode == isNightMode && self.pdfViewCtrl.colorMode != FSRendererColorModeMappingGray)) {
            self.pdfViewCtrl.isNightMode = isNightMode;
    }
}

```

```
if (self.pdfViewCtrl.nightColorMode == FSNightColorModeMappingGray && isNightMode) {
    self.pdfViewCtrl.colorMode = FSRendererColorModeMappingGray;
}

if (isNightMode && self.pdfViewCtrl.nightColorMode != FSNightColorModeMappingGray)
{
    // Set background color.
    self.pdfViewCtrl.mappingModeBackgroundColor = [UIColor redColor];
    // Set foreground color.
    self.pdfViewCtrl.mappingModeForegroundColor = [UIColor greenColor];
    // Set color mode.
    self.pdfViewCtrl.colorMode = FSRendererColorModeMapping;
    // Set the background color for the areas that are out of the pdfviewctrl.
    self.pdfViewCtrl.backgroundColor = [UIColor colorWithRed:20.0/255.0 green:20.0/255.0
blue:20.0/255.0 alpha:1.0];
}
else {
    self.pdfViewCtrl.colorMode = FSRendererColorModeNormal;
    self.pdfViewCtrl.backgroundColor = [UIColor colorWithRed:230.0/255.0 green:230.0/255.0
blue:230.0/255.0 alpha:1.0];
}
}
}
```

11.12 Foxit SDK Framework 上传到 Apple App Store

为什么当我在上传嵌入 Foxit PDF SDK framework 的应用程序到 Apple App Store 时，会出现 "Invalid Binary Architecture" 警告和 "Code signing "FoxitRDK.framework" failed" 错误？

Foxit SDK Framework 包括 arm64, armv7, i386 和 x86_64 架构，但是 i386 和 x86_64 架构的 framework 不允许上传到 Apple App Store，因此，如果您直接使用这两种架构的 framework 就会遇到 "Invalid Binary Architecture" 的警告和 "Code signing "FoxitRDK.framework" failed" 错误。

为了解决这个问题，Foxit SDK 在 "libs" 文件夹下提供了一个名为 "prepare-framework-to-publish-to-appstore.sh" 的脚本来剥离 arm 架构，然后将剥离出的库保存在 "./device/FoxitRDK.framework" 目录下。您应该在工程中使用此库，这样您的应用程序才能被上传到 Apple App Store。

11.13 输出 exception/crash 日志信息

如何在应用程序抛出异常或者 crash 时，输出 exception/crash 日志信息？

如果您需要输出 exception/crash 日志信息，您只需要实现<**IExceptionLogger**>协议，然后调用 **FSPDFViewControl::setExceptionLogger** 接口即可。

11.14 本地化设置

如何使用 Foxit PDF SDK for iOS 更改本地化设置？

默认情况下，Foxit PDF SDK for iOS 会根据您系统的当前语言自动切换 UI 语言，前提是 Foxit PDF SDK for iOS 支持该语言。

当前，Foxit PDF SDK for iOS 支持如下的语言：英语(English)，韩语(Korean) 和 中文(Chinese，简体中文和繁体中文)。这些语言的资源文件位于

"libs\uiextensions_src\UIExtensions\Resource\FoxitLocalizable"文件夹下。

如果您需要使用自己本地的语言(而该语言是 Foxit PDF SDK for iOS 当前不支持的语言)，请参考如下的说明：

- 对于 SDK 6.4 版本，您可以参阅文章
<https://developers.foxitsoftware.com/kb/article/change-localization-settings-pdf-sdk-ios/>
来进行本地化设置，需要在 UIExtensions 组件中进行相关的设置。
- 对于 SDK 7.0 及以上版本，您可以直接在您的工程中进行设置。首先，在工程中添加您需要的本地语言，然后将生成的语言文件中的所有字段都翻译成您需要的本地语言。其次，调用 **FSLocalization::addLanguage** 接口添加新加的语言。最后，修改您系统的当前语言为您本地的语言，或者调用 **FSLocalization::setCurrentLanguage** 接口使其生效。

12 技术支持

问题报告

Foxit 为其产品提供全天候 24 小时支持，并拥有 PDF 行业优秀的技术支持工程师开发团队。如果您在使用 Foxit PDF SDK for iOS 时遇到任何技术问题或 bug，请在 <http://tickets.foxitsoftware.com/create.php> 网页上将问题报告提交给 Foxit 技术支持团队。为了更好地帮助您解决问题，请提供以下信息：

- 联系方式
- Foxit PDF SDK 产品和版本
- 您使用的操作系统和 IDE 版本
- 问题的详细说明
- 任何其他相关信息，例如日志文件或错误信息截图

联系方式

您可以直接联系 Foxit，请使用以下的联系方式：

线上支持：

- <http://www.foxitsoftware.com/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