



开发者指南

FOXIT PDF SDK

鸿蒙版

TABLE OF CONTENTS

| | | |
|----------|---|-----------|
| 1 | Foxit PDF SDK 简介..... | 1 |
| 1.1 | Foxit PDF SDK..... | 1 |
| 1.2 | Foxit PDF SDK 鸿蒙版 | 1 |
| 1.2.1 | 为什么选择 Foxit PDF SDK 鸿蒙版..... | 1 |
| 1.2.2 | Foxit PDF SDK 鸿蒙版的主要框架 | 2 |
| 1.2.3 | UI Extensions 组件概述 | 3 |
| 1.2.4 | Foxit PDF SDK 鸿蒙版的主要功能 | 5 |
| 1.3 | 评估 | 6 |
| 1.4 | 授权 | 7 |
| 1.5 | 关于此文档 | 7 |
| 2 | 入门指南..... | 8 |
| 2.1 | 系统要求 | 8 |
| 2.2 | 包结构说明 | 8 |
| 2.3 | 运行 demo..... | 10 |
| 3 | 快速构建一个功能丰富的 PDF 阅读器..... | 12 |
| 3.1 | 创建一个新的鸿蒙工程..... | 12 |
| 3.2 | 集成 Foxit PDF SDK 鸿蒙版到您的应用程序 | 14 |
| 3.3 | 初始化 Foxit PDF SDK 鸿蒙版 | 16 |
| 3.4 | 添加 PDF 文件到沙盒目录 | 16 |
| 3.5 | 使用 PDFViewCtrl 显示 PDF 文档 | 18 |
| 3.6 | 使用 UI Extensions 组件构建一个功能丰富的 PDF 阅读器..... | 22 |
| 4 | 自定义 UI..... | 27 |
| 4.1 | 通过配置文件自定义 UI..... | 27 |
| 4.1.1 | JSON 文件介绍 | 27 |
| 4.1.2 | 配置项描述 | 30 |
| 4.1.3 | 使用配置文件实例化一个 UIExtensionsManager 对象..... | 32 |

| | | |
|----------|---|-----------|
| 4.1.4 | 通过配置文件自定义 UI 的示例 | 33 |
| 5 | 使用 SDK API | 35 |
| 5.1 | Render..... | 35 |
| 5.1.1 | 如何将指定的 PDF 页面渲染到 bitmap | 35 |
| 5.2 | Text Page | 37 |
| 5.2.1 | 如何通过选择获取页面上的文本区域..... | 37 |
| 5.3 | Text Search..... | 38 |
| 5.3.1 | 如何在 PDF 文档中搜索指定的文本模型..... | 38 |
| 5.4 | Bookmark (Outline)..... | 39 |
| 5.4.1 | 如何使用深度优先顺序遍历 PDF 文档的 bookmarks | 40 |
| 5.5 | Reading Bookmark | 41 |
| 5.5.1 | 如何添加自定义 reading bookmark 并枚举所有的 reading bookmarks | 41 |
| 5.6 | Attachment..... | 42 |
| 5.6.1 | 如何将指定文件嵌入到 PDF 文档 | 42 |
| 5.6.2 | 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 | 43 |
| 5.7 | Annotation..... | 43 |
| 5.7.1 | 如何向 PDF 页面中添加注释 | 44 |
| 5.7.2 | 如何删除 PDF 页面中的注释 | 46 |
| 5.7.3 | 如何注册监听器接收注释事件 | 46 |
| 5.8 | Form | 47 |
| 5.8.1 | 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..... | 47 |
| 5.9 | Security | 48 |
| 5.9.1 | 如何使用密码加密 PDF 文件 | 48 |
| 5.10 | Signature..... | 49 |
| 5.10.1 | 如何对 PDF 文档进行签名，并验证签名..... | 49 |
| 5.10.2 | 如何为签名设置定制化时间信息 | 51 |
| 6 | FAQ..... | 54 |
| 6.1 | 从指定的 PDF 文件路径打开一个 PDF 文档 | 54 |
| 6.2 | 打开 PDF 文档时显示指定的页面 | 55 |

| | | |
|----------|---|-----------|
| 6.3 | License key 和序列号无法正常工作 | 57 |
| 6.4 | 在 PDF 文档中添加 link 注释..... | 58 |
| 6.5 | 向 PDF 文档中插入图片 | 59 |
| 6.6 | 高亮 PDF 文档中的表单域和设置高亮颜色..... | 60 |
| 6.7 | 支持全文索引搜索 | 61 |
| 6.8 | 夜间模式颜色设置 | 63 |
| 6.9 | 使用 UIExtensions 设置只读模式 | 64 |
| 6.10 | 更新页面绑定 (page binding) 支持 RTL (Right-to-Left)..... | 65 |
| 6.11 | 打开网页 PDF 文件的问题 | 65 |
| 7 | 技术支持..... | 67 |

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 以及鸿蒙) 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/>。

在本指南中，我们将重点介绍 Foxit PDF SDK 鸿蒙版平台。

1.2 Foxit PDF SDK 鸿蒙版

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丰富的 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 鸿蒙版致力于帮助开发者快速构建高性能、跨设备的 HarmonyOS PDF 应用程序。通过 Foxit 开发包，即使是对 PDF 了解有限的开发人员也可以在 HarmonyOS Next 鸿蒙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 鸿蒙版

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 鸿蒙版提供了快速 PDF 阅读和鸿蒙设备的操作控制。选择 Foxit PDF SDK 鸿蒙版的几大理由：

- **易于集成**

开发人员可以通过几行代码将 SDK 无缝集成到他们自己的应用程序中。

- **设计完美**

Foxit PDF SDK 鸿蒙版拥有简单、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 鸿蒙版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 鸿蒙版提供了 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 Foxit 高保真的 PDF 渲染引擎**

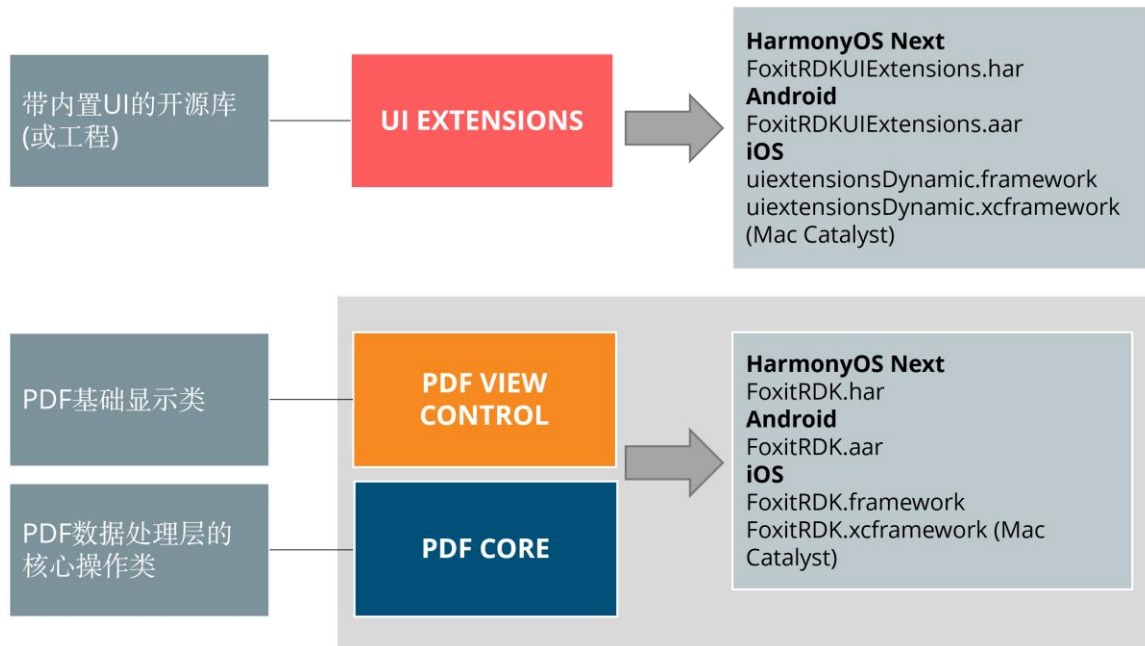
Foxit PDF SDK 的核心技术是基于世界众多知名企业所信赖的 Foxit PDF 引擎。Foxit 强大的 PDF 引擎可快速解析和渲染文档，不受设备环境的约束。

- **优秀的技术支持**

Foxit 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。Foxit 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 鸿蒙版的主要框架

Foxit PDF SDK 鸿蒙版由三种元素组成，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。



Foxit PDF SDK for Android, iOS, 以及 HarmonyOS Next 的三种组成元素

- **PDF Core API**

PDF Core API 是 SDK 的核心部分，建立在 Foxit 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 Foxit 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 是一个自定义组件，用户可以将这个组件集成到自己的业务逻辑中，并且允许进行扩展来满足特定用户的需求。

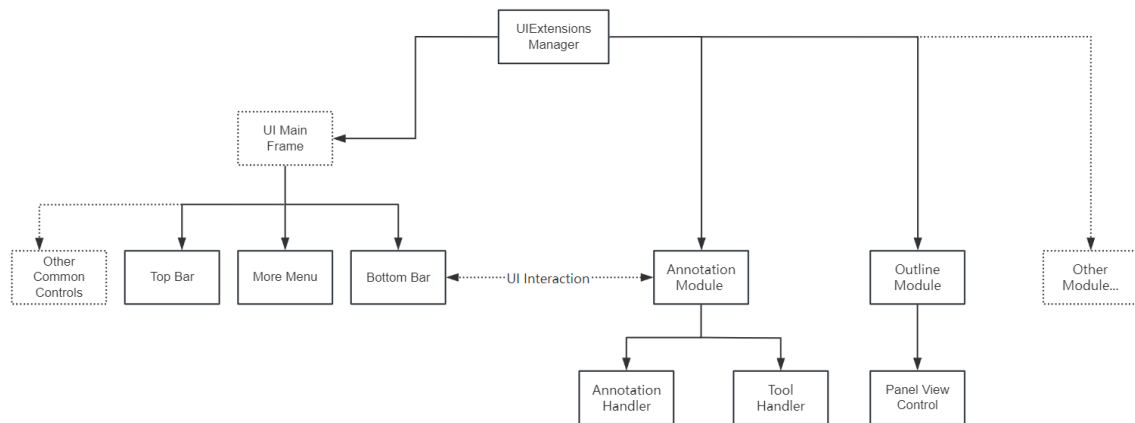
- **UI Extensions 组件**

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。

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 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。

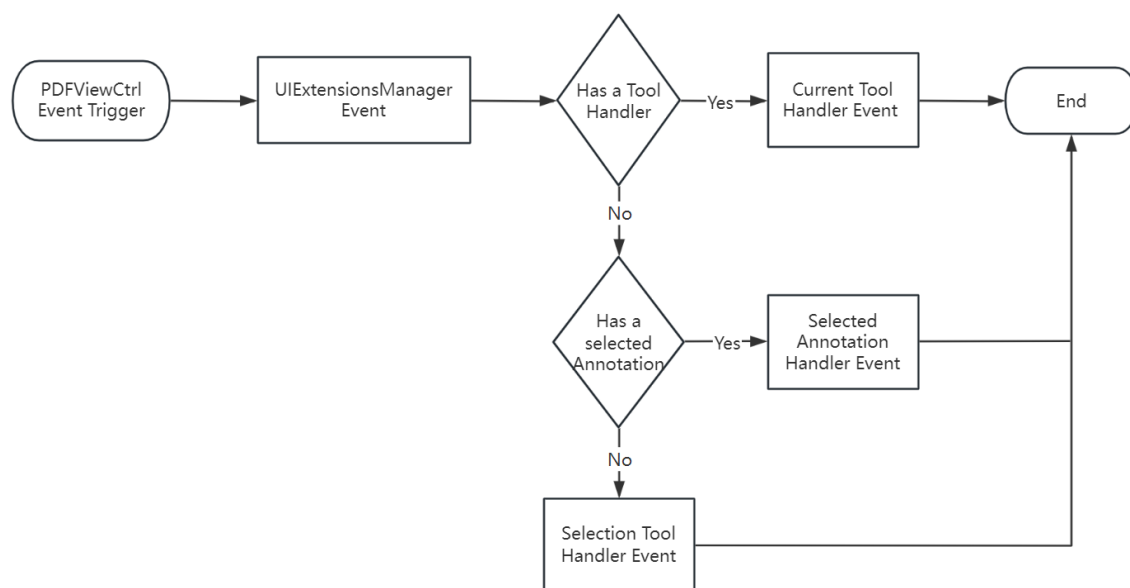


UIExtensionsManager 和 modules 之间的关系

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 Handler 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)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 鸿蒙版的主要功能

Foxit PDF SDK 鸿蒙版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。

| 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 |
| Comparison | 对比两个 PDF 文档，并且标记文档之间的差异 |
| Split Screen | 支持分屏 |
| Right to Left | 支持 RTL (Right to Left) |
| 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 鸿蒙版支持鲁棒性的 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 鸿蒙版提供了一种内存溢出(OOM) 机制。

OOM 是 Foxit PDF SDK 鸿蒙版的一个高级功能，因为其本身的复杂性。OOM 机制的关键点是 Foxit PDF SDK 鸿蒙版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 鸿蒙版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。

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 Foxit 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 Foxit 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 鸿蒙版集成到自己的 HarmonyOS Next 应用程序中的开发人员。它旨在介绍以下章节：

- 章节 1: 介绍 Foxit PDF SDK，特别是 HarmonyOS Next 的鸿蒙版 SDK。
- 章节 2: 说明包的结构，以及运行 demo。
- 章节 3: 介绍如何快速创建功能丰富的 PDF 阅读器。
- 章节 4: 介绍如何自定义用户界面。
- 章节 5: 介绍如何使用 Foxit PDF SDK core API。
- 章节 6: 列出常见问题。
- 章节 7: 提供技术支持信息。

2 入门指南

安装并集成 Foxit PDF SDK 鸿蒙版非常简单。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 HarmonyOS Next 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

鸿蒙设备要求

- 支持 arm64-v8a 和 x86_64 架构
- HarmonyOS Next SDK 5.0.0(12)

DevEco Studio NEXT Developer Beta3

包中 Demos 的运行环境：

- Build #DS-233.14475.28.36.503600
- Build Version: 5.0.3.600, built on August 7, 2024
- Runtime version: 17.0.10+1-b1087.17 aarch64
- VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

2.2 包结构说明

下载 "foxitpdfsdk_1_0_harmonyos_next.zip"包，解压到一个新的目录如 "foxitpdfsdk_1_0_harmonyos_next"，如 Figure 2-1 所示。其中解压包中包括如下的内容：

| | |
|---------------------------|--------------------------------------|
| docs: | API 手册，开发文档，说明手册 |
| libs: | License 文件，HAR 库，UI Extensions 组件源代码 |
| samples: | 鸿蒙示例工程 |
| legal.txt: | 法律和版权信息 |
| release_notes.txt: | 发布信息 |

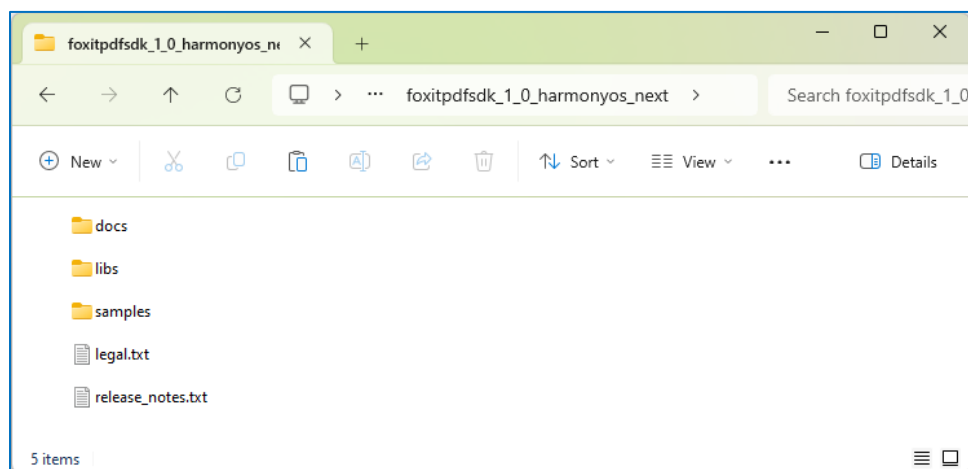


Figure 2-1

如 Figure 2-2 所示的"libs"文件夹下是 Foxit PDF SDK 鸿蒙版的核心组件。

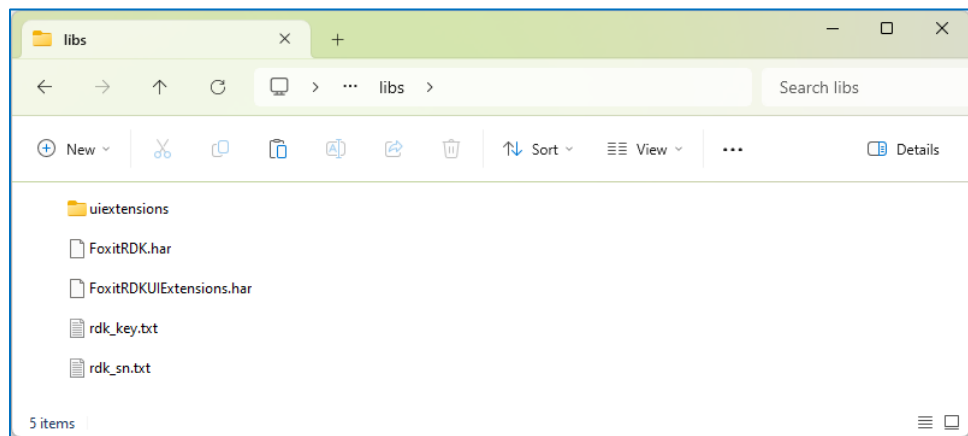


Figure 2-2

其中：

- **uiextensions** 工程 – 在"libs"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，可以帮助开发人员快速将功能丰富的 PDF 阅读器嵌入到他们的鸿蒙应用中。当然，开发人员也可以通过"uiextensions"工程为特定的应用灵活自定义和设计 UI。
- **FoxitRDK.har** – 包含了 Foxit PDF SDK 鸿蒙版所有的 TS APIs，以及".so"库。".so"库是 SDK 的核心，包含了 Foxit PDF SDK 鸿蒙版的核心函数。它针对每种架构单独编译，当期支持 arm64-v8a 和 x86_64 架构。
- **FoxitRDKUIExtensions.har** – 由"libs"目录下的 "uiextensions"工程编译生成。包含内置 UI 实现，以及 UI 所需要的资源文件，如图片，字符串、颜色值、布局文件以及其他 UI 资源。

2.3 运行 demo

下载和安装 DevEco Studio IDE (<https://developer.huawei.com/consumer/cn/download/deveco-studio>)。

Foxit PDF SDK 鸿蒙版为开发人员提供了一个 Complete PDF Viewer demo，位于 "samples" 目录下。该 demo 阐述了如何通过使用 Foxit PDF SDK 鸿蒙版实现一个功能丰富的 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 鸿蒙版所提供的所有功能和内置 UI 实现。

在 DevEco Studio 中运行该 demo，请按如下的步骤：

- a) 在 DevEco Studio 中打开 demo，通过 "File -> Open..."，然后找到 Complete PDF Viewer demo 所在的位置，选择 complete_pdf_viewer。点击 "OK"。
- b) 开启一个鸿蒙设备或者模拟器。在本章中，将使用模拟器来运行 demo。在运行该 demo 时，"**samples\complete_pdf_viewer\entry\src\main\resources\rawfile**" 目录下的 "sample.pdf" 文件将会被自动拷贝到模拟器的沙盒目录下。
- c) 点击 "Run -> Run 'entry'" 来运行 demo。当在模拟器上安装成功后，会显示 Complete PDF Viewer demo 的主页，点击 "所有 PDF"，然后点击 "sample.pdf" 文档，可以看到如 Figure 2-3 所示的功能选项。该 demo 实现了一个功能丰富的 PDF 阅读器，请随意体验。



Figure 2-3

3 快速构建一个功能丰富的 PDF 阅读器

Foxit PDF SDK 鸿蒙版将所有的 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丰富的 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 鸿蒙版在 HarmonyOS Next 鸿蒙平台创建一个功能丰富的 PDF 阅读器。其主要包括以下的步骤：

- [创建一个新的鸿蒙工程](#)
- [集成 Foxit PDF SDK 鸿蒙版到您的应用程序](#)
- [初始化 Foxit PDF SDK 鸿蒙版](#)
- [添加 PDF 文件到沙盒目录中](#)
- [使用 PDFViewCtrl 显示 PDF 文档](#)
- [使用 UI Extensions 组件构建一个功能丰富的 PDF 阅读器](#)

3.1 创建一个新的鸿蒙工程

在本指南中，使用 DevEco Studio NEXT Developer Beta3，以及 OpenHarmony SDK API Version 12 Beta3.

打开 DevEco Studio，选择 **File -> New -> Create Project...**，在 **Choose Your Ability Template** 向导中，选择 "Empty Ability"，如 Figure 3-1 所示。然后点击 **Next**。

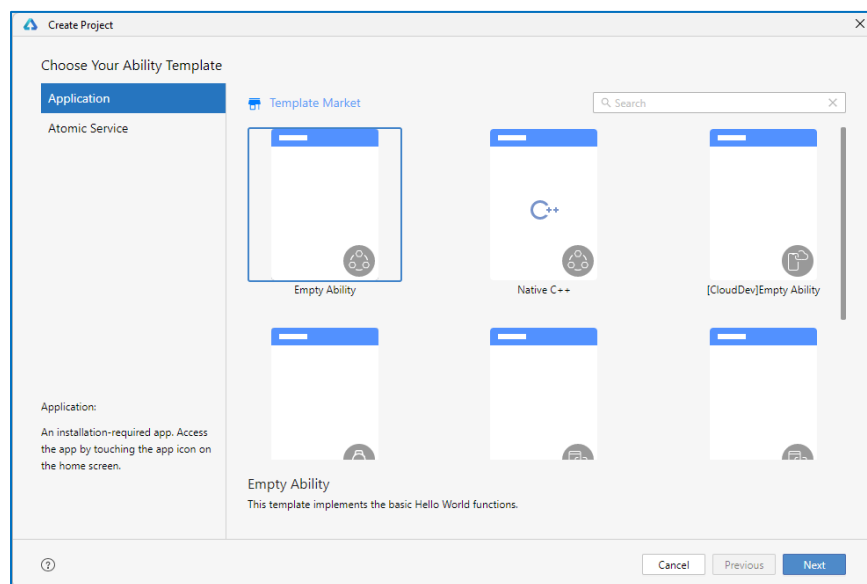


Figure 3-1

然后填写 **Configure Your Project** 对话框，如 Figure 3-2 所示。填写完后，点击 **Finish**。

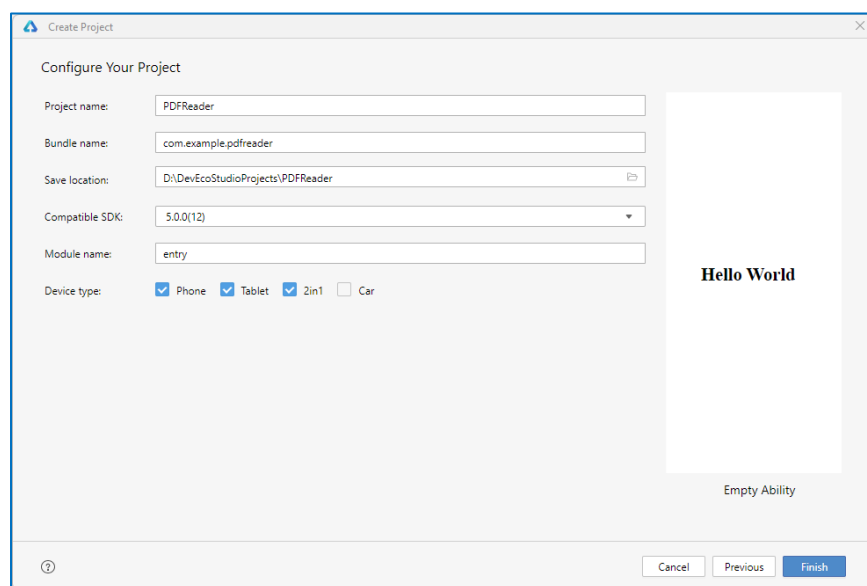


Figure 3-2

3.2 集成 Foxit PDF SDK 鸿蒙版到您的应用程序

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.har** – 包含了 Foxit PDF SDK 鸿蒙版所有的 TS APIs，以及".so"库。".so"库是 SDK 的核心，包含了 Foxit PDF SDK 鸿蒙版的核心函数。它针对每种架构单独编译，当期支持 arm64-v8a 和 x86_64 架构。
- **FoxitRDKUIExtensions.har** – 由"libs"目录下的 "uiextensions"工程编译生成。包含内置 UI 实现，以及 UI 所需要的资源文件，如图片，字符串、颜色值、布局文件以及其他 UI 资源。

在 PDFReader 工程中安装上述的两个库文件，请按照如下的步骤：

- a) 将下载包中的 "libs" 文件夹拷贝到工程根目录，即 "PDFReader" 文件夹下。
- b) 安装 FoxitRDK.har 库。

打开 Terminal，在工程根目录下，运行以下命令来安装 FoxitRDK.har，如 Figure 3-3 所示。

```
ohpm install libs/FoxitRDK.har
```

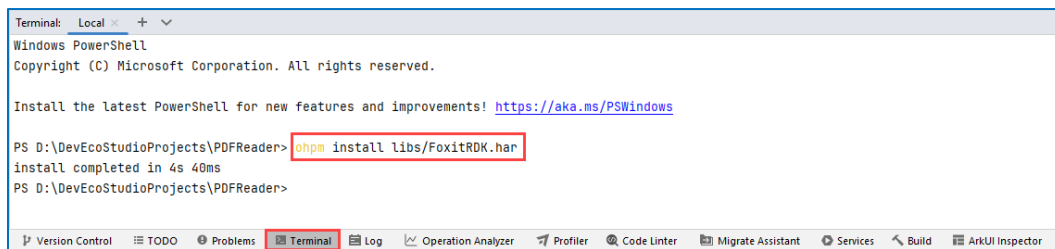


Figure 3-3

安装成功后，在 "PDFReader/oh_modules" 目录下会生成 **foxit_rdk** 文件夹，以及在 "PDFReader/oh-package.json5" 文件中会自动生成引用，如 Figure 3-4 所示。

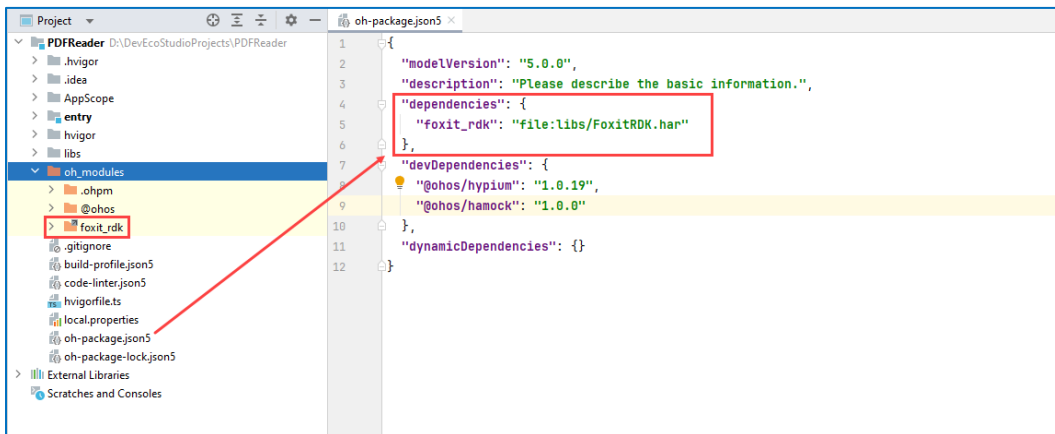


Figure 3-4

c) 安装 FoxitRDKUIExtensions.har 库。

由于 FoxitRDKUIExtensions.har 依赖 FoxitRDK.har，因此当在本地环境安装时，您需要在 "PDFReader/oh-package.json5" 文件中添加如下的命令：

```
"overrides": {  
  "foxit_rdk": "file:libs/FoxitRDK.har"  
},
```

然后，打开 Terminal，在工程根目录下，运行以下命令来安装 FoxitRDKUIExtensions.har：

```
ohpm install libs/FoxitRDKUIExtensions.har
```

安装成功后，在 "PDFReader/oh_modules" 目录下会生成 **uiextensions** 文件夹，以及在 "PDFReader/oh-package.json5" 文件中会自动生成引用，如 Figure 3-5 所示。

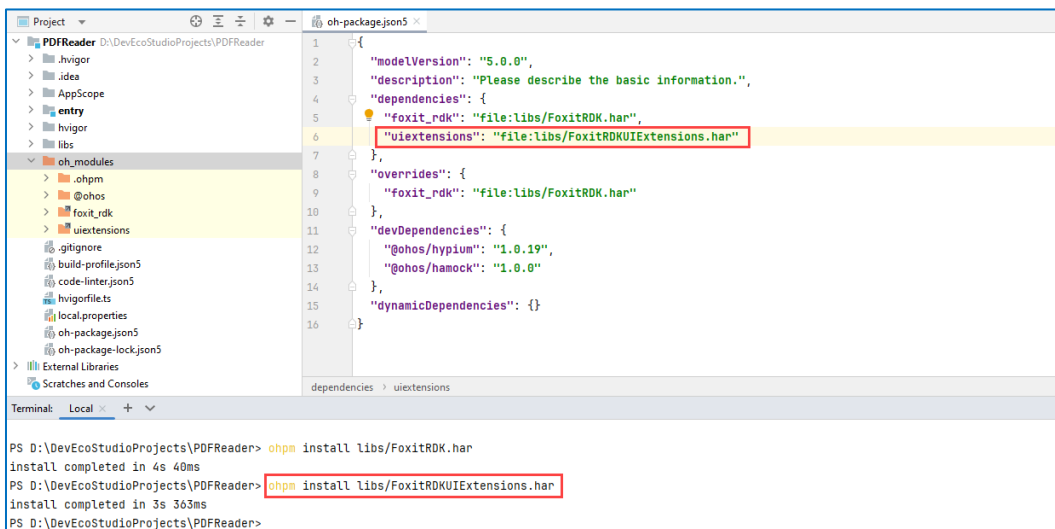


Figure 3-5

3.3 初始化 Foxit PDF SDK 鸿蒙版

在调用任何 API 之前，应用程序必须使用 license 初始化 Foxit PDF SDK 鸿蒙版。

Library.Initialize(sn, 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。以下是初始化 SDK 库的步骤。

在 "PDFReader/entry/src/main/ets/pages/Index.ets" 文件中，添加如下的代码：

- a) 在工程中引入 foxit_rdk 模块和 uiextensions 模块：

```
// 引入 foxit_rdk 模块
import { FoxitRDKNative, PDFViewCtrl, PDFViewCtrlModel, } from 'foxit_rdk';

// 引入 uiextensions 模块
import { UIExtensionsManager, UIExtensionsComponent } from 'uiextensions';
```

- b) 初始化 SDK 库：

```
let sn: string = 'sn'
let key: string = 'key'

FoxitRDKNative.Library.Initialize(sn, key);
```

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)。

3.4 添加 PDF 文件到沙盒目录

备注：PDF 文件的管理是由应用层自己实现的，应用层可以创建一个文件列表来管理文件，具体可参考鸿蒙系统的 API。本示例为了简单起见，将 PDF 文件拷贝到沙盒目录下。

首先，将 PDF 文件，比如 "sample.pdf"，拷贝到 "PDFReader/entry/src/main/resources/rawfile" 文件夹下。然后，在 "PDFReader/entry/src/main/ets/pages/Index.ets" 文件中添加如下的代码：

```
import fs from '@ohos.file.fs';

let context = getContext(this)
let filesDir = context.filesDir;

let fileName = "sample.pdf";
let openDocPath = filesDir + "/" + fileName;

export async function copy_rawfile__to_sandbox(cb: (error: Error) => void) {
  try {

    let sss = fs.createStreamSync(openDocPath, "w");
    sss.closeSync();
```

```
// 获取rawfile 目录下的"sample.pdf"
context.resourceManager.getRawFd('rawfile/' + fileName, async (error, value) => {

  if (error != null) { // getRawFileDescriptor 运行失败
    console.log("[rawfile_copy_to_sandbox] 未能成功将 rawfile 目录下的 sample.pdf 文件拷贝到应用沙盒目录下");
  } else {           // getRawFileDescriptor 运行成功
    let fd = value.fd;
    console.log("LXG fd:" + fd)
    console.log("LXG fileDir:" + filesDir)
    // 打开文件流
    let inputStream = fs.fdopenStreamSync(fd, "r+");
    console.log("LXG inputStream:")
    let outputStream = fs.createStreamSync(openDocPath, "w+");
    // 以流的形式读取源文件内容并写入目标文件
    let bufSize = value.length;
    let readSize = 0;
    let buf = new ArrayBuffer(bufSize);

    class Option {
      public offset: number = 0;
      public length: number = bufSize;
    }

    let option = new Option();
    option.offset = value.offset;
    let readLen = await inputStream.read(buf, option);
    readSize += readLen;
    while (readLen > 0) {
      await outputStream.write(buf);
      option.offset = readSize + value.offset;
      option.length = value.length - readSize >= bufSize ? bufSize : value.length - readSize;
      readLen = await inputStream.read(buf, option);
      readSize += readLen;
    }
    // 关闭文件流
    inputStream.closeSync();
    outputStream.closeSync();
  }
  return cb(error);
});

} catch (error) {
  console.log("[rawfile_copy_to_sandbox] 未能成功将 rawfile 目录下的 sample.pdf 文件拷贝到应用沙盒目录下");
  return cb(error);
}
return cb(new Error());
}
```

3.5 使用 PDFViewCtrl 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 鸿蒙版，并且完成了 SDK 库的初始化，以及添加了 PDF 文档。现在，我们将使用 PDFViewCtrl 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。

在 "PDFReader/entry/src/main/ets/pages/Index.ets" 文件中添加如下的代码：

```
@Entry
@Component
struct Index {
  @State message: string = 'Open Doc';
  @State clickID: number = 0;
  @State pdfViewCtrlModel: PDFViewCtrlModel = new PDFViewCtrlModel(getContext(this))
  public openDoc(){
    this.pdfViewCtrlModel.openDoc(openDocPath, "", (error)=>{
      if (error == FoxitRDKNative.ErrorCode.e_ErrSuccess) {
        this.clickID = 1;
      }
    })
  }
  build() {
    Row() {
      Column() {
        if (this.clickID == 0){
          Button(this.message)
            .fontSize(50)
            .fontWeight(FontWeight.Bold)
            .onClick(() => {
              let res = fs.accessSync(openDocPath);
              if (!res){
                copy_rawfile__to_sandbox((error) =>{
                  if (error == null){
                    try {
                      this.openDoc()
                    } catch (err) {
                      console.info(` fail callback, code: ${err.code}, msg: ${err.msg}`)
                    }
                  }
                })
              }
            })
        } else {
          this.openDoc()
        }
      }
    }
  }
}
```

```
}  
    .width('100%')  
}  
    .height('100%')  
}  
}
```

当前, "PDFReader/entry/src/main/ets/pages/Index.ets" 的全部代码如下所示:

```
import { FoxitRDKNative, PDFViewCtrl, PDFViewCtrlModel, } from 'foxit_rdk';  
import fs from '@ohos.file.fs';  
  
// 初始化 SDK 库  
let sn: string = "  
let key: string = "  
FoxitRDKNative.Library.Initialize(sn, key);  
  
// 将 PDF 文档拷贝到应用沙盒目录下  
let context = getContext(this)  
let filesDir = context.filesDir;  
  
let fileName = "sample.pdf";  
let openDocPath = filesDir + "/" + fileName;  
  
export async function copy_rawfile__to_sandbox(cb: (error: Error) => void) {  
    try {  
  
        let sss = fs.createStreamSync(openDocPath, "w");  
        sss.closeSync();  
  
        // 获取 rawfile 目录下的 "sample.pdf"  
        context.resourceManager.getRawFd('rawfile/' + fileName, async (error, value) => {  
  
            if (error != null) { // getRawFileDescriptor 运行失败  
                console.log("[rawfile_copy_to_sandbox] 未能成功将 rawfile 目录下的 sample.pdf 文件拷贝到应用沙盒目录下  
");  
            } else { // getRawFileDescriptor 运行成功  
                let fd = value.fd;  
                console.log("LXG fd:" + fd)  
                console.log("LXG fileDir:" + filesDir)  
                // 打开文件流  
                let inputStream = fs.fdopenStreamSync(fd, "r+");  
                console.log("LXG inputStream:")  
                let outputStream = fs.createStreamSync(openDocPath, "w+");  
                // 以流的形式读取源文件内容并写入目标文件  
                let bufSize = value.length;  
                let readSize = 0;  
                let buf = new ArrayBuffer(bufSize);  
  
                class Option {  
                    public offset: number = 0;  
                    public length: number = bufSize;
```

```

    }

    let option = new Option();
    option.offset = value.offset;
    let readLen = await inputStream.read(buf, option);
    readSize += readLen;
    while (readLen > 0) {
        await outputStream.write(buf);
        option.offset = readSize + value.offset;
        option.length = value.length - readSize >= bufSize ? bufSize : value.length - readSize;
        readLen = await inputStream.read(buf, option);
        readSize += readLen;
    }
    // 关闭文件流
    inputStream.closeSync();
    outputStream.closeSync();
}
return cb(error);
});

} catch (error) {
    console.log("[rawfile_copy_to_sandbox] 未能成功将 rawfile 目录下的 sample.pdf 文件拷贝到应用沙盒目录下");
    return cb(error);
}
return cb(new Error());
}

// 使用PDFViewCtrl 打开一个PDF 文档
@Entry
@Component
struct Index {
    @State message: string = 'Open Doc';
    @State clickID: number = 0;
    @State pdfViewCtrlModel: PDFViewCtrlModel = new PDFViewCtrlModel(getContext(this))
    public openDoc(){
        this.pdfViewCtrlModel.openDoc(openDocPath, "", (error)=>{
            if (error == FoxitRDKNative.ErrorCode.e_ErrSuccess) {
                this.clickID = 1;
            }
        })
    }
}

build() {
    Row() {
        Column() {
            if (this.clickID == 0){
                Button(this.message)
                    .fontSize(50)
                    .fontWeight(FontWeight.Bold)
                    .onClick(() => {
                        let res = fs.accessSync(openDocPath);
                        if (!res){
                            copy_rawfile__to_sandbox((error) =>{

```



```
        if (error == null){
            try {
                this.openDoc()
            } catch (err) {
                console.info(` fail callback, code: ${err.code}, msg: ${err.msg}`)
            }
        }
    })
    }else {
        this.openDoc()
    }
})
}else {
    PDFViewCtrl({model: this.pdfViewCtrlModel})
    .width('100%')
    .height('100%')
    .backgroundColor(Color.Black)
}
}
.width('100%')
}
.height('100%')
}
```

在本节中，使用鸿蒙模拟器来运行该工程。运行成功后，点击 "Open Doc" 按钮，即可看到 "sample.pdf" 文档显示如 Figure 3-6 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。

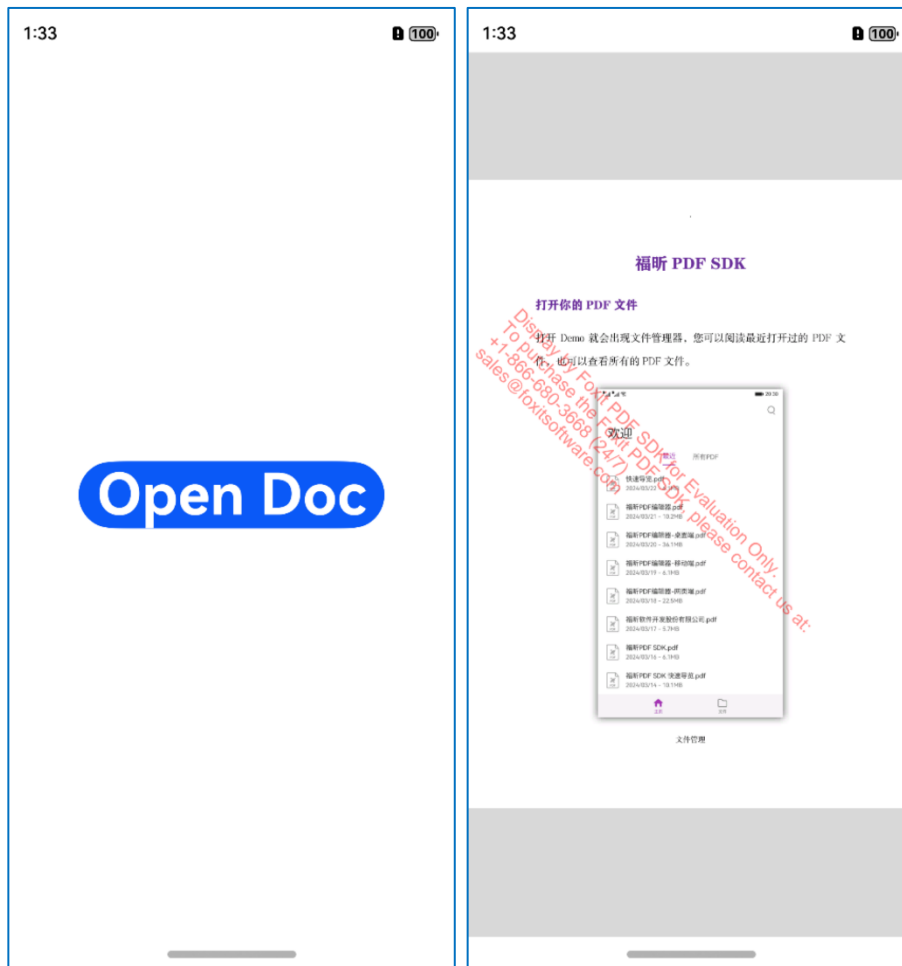


Figure 3-6

3.6 使用 UI Extensions 组件构建一个功能丰富的 PDF 阅读器

Foxit PDF SDK 鸿蒙版带有内置的 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 鸿蒙版实现，并且封装在 UI Extensions 组件中。因此，构建一个功能丰富的 PDF 阅读器变得越来越简单。您只需要实例化一个 UIExtensionsManager 对象。

在 "PDFReader/entry/src/main/ets/pages/Index.ets" 文件中添加如下的代码：

```
@Entry
@Component
struct Index {
  @State message: string = 'Open Doc';
  @State clickID: number = 0;
  @State uiextensionsManager: UIExtensionsManager = new UIExtensionsManager(getContext(this));
  public openDoc(){
    this.uiextensionsManager.openDocument(openDocPath, "", (error: number)=>{
      if (error == FoxitRDKNative.ErrorCode.e_ErrSuccess) {
        this.clickID = 1;
      }
    });
  }
}
```

```
    }  
  })  
}  
  
build() {  
  Row() {  
    Column() {  
      if (this.clickID == 0){  
        Button(this.message)  
          .fontSize(50)  
          .fontWeight(FontWeight.Bold)  
          .onClick(() => {  
            let res = fs.accessSync(openDocPath);  
            if (!res){  
              copy_rawfile__to_sandbox((error) =>{  
                if (error == null){  
                  try {  
                    this.openDoc()  
                  } catch (err) {  
                    console.info(` fail callback, code: ${err.code}, msg: ${err.msg}`)  
                  }  
                }  
              })  
            }  
          })  
        }else {  
          this.openDoc()  
        }  
      }  
    }  
  }  
  .width('100%')  
  .height('100%')  
  .height('100%')  
}
```

当前, "PDFReader/entry/src/main/ets/pages/Index.ets" 的全部代码如下所示:

```
import { FoxitRDKNative, PDFViewCtrl, PDFViewCtrlModel, } from 'foxit_rdk';  
import { UIExtensionsManager, UIExtensionsComponent } from 'uiextensions';  
  
import fs from '@ohos.file.fs';  
  
// 初始化 SDK 库  
let sn: string = "  
let key: string = "
```

```
FoxitRDKNative.Library.Initialize(sn, key);

// 将PDF 文档拷贝到应用沙盒目录下
let context = getContext(this)
let filesDir = context.filesDir;

let fileName = "sample.pdf";
let openDocPath = filesDir + "/" + fileName;

export async function copy_rawfile__to_sandbox(cb: (error: Error) => void) {
  try {

    let sss = fs.createStreamSync(openDocPath, "w");
    sss.closeSync();

    // 获取rawfile 目录下的"sample.pdf"
    context.resourceManager.getRawFd('rawfile/' + fileName, async (error, value) => {

      if (error != null) { // getRawFileDescriptor 运行失败
        console.log("[rawfile_copy_to_sandbox] 未能成功将 rawfile 目录下的 sample.pdf 文件拷贝到应用沙盒目录下");
      } else { // getRawFileDescriptor 运行成功
        let fd = value.fd;
        console.log("LXG fd:" + fd)
        console.log("LXG fileDir:" + filesDir)
        // 打开文件流
        let inputStream = fs.fdopenStreamSync(fd, "r+");
        console.log("LXG inputStream:")
        let outputStream = fs.createStreamSync(openDocPath, "w+");
        // 以流的形式读取源文件内容并写入目标文件
        let bufSize = value.length;
        let readSize = 0;
        let buf = new ArrayBuffer(bufSize);

        class Option {
          public offset: number = 0;
          public length: number = bufSize;
        }

        let option = new Option();
        option.offset = value.offset;
        let readLen = await inputStream.read(buf, option);
        readSize += readLen;
        while (readLen > 0) {
          await outputStream.write(buf);
          option.offset = readSize + value.offset;
          option.length = value.length - readSize >= bufSize ? bufSize : value.length - readSize;
          readLen = await inputStream.read(buf, option);
          readSize += readLen;
        }
      }
    });
  }
}
```

```
// 关闭文件流
inputStream.closeSync();
outputStream.closeSync();
}
return cb(error);
});

} catch (error) {
  console.log("[rawfile_copy_to_sandbox] 未能成功将 rawfile 目录下的 sample.pdf 文件拷贝到应用沙盒目录下");
  return cb(error);
}
return cb(new Error());
}

// 使用 UI Extensions 组件构建一个功能丰富的 PDF 阅读器
@Entry
@Component
struct Index {
  @State message: string = 'Open Doc';
  @State clickID: number = 0;
  @State uiextensionsManager: UIExtensionsManager = new UIExtensionsManager(getContext(this));
  public openDoc(){
    this.uiextensionsManager.openDocument(openDocPath, "", (error: number)=>{
      if (error == FoxitRDKNative.ErrorCode.e_ErrSuccess) {
        this.clickID = 1;
      }
    })
  }

  build() {
    Row() {
      Column() {
        if (this.clickID == 0){
          Button(this.message)
            .fontSize(50)
            .fontWeight(FontWeight.Bold)
            .onClick(() => {
              let res = fs.accessSync(openDocPath);
              if (!res){
                copy_rawfile__to_sandbox((error) =>{
                  if (error == null){
                    try {
                      this.openDoc()
                    } catch (err) {
                      console.info(` fail callback, code: ${err.code}, msg: ${err.msg}`)
                    }
                  }
                })
              }
            })
        } else {
          this.openDoc()
        }
      }
    }
  }
}
```

```
}  
})  
} else {  
  UIExtensionsComponent({uiExtMgr: this.uiextensionsManager})  
    .width('100%')  
    .height('100%')  
    .backgroundColor(Color.Black)  
  }  
}  
.width('100%')  
}  
.height('100%')  
}
```

运行成功后，点击 "Open Doc" 按钮，即可看到 "sample.pdf" 文档显示如 Figure 3-7 所示。至此，该工程是一个功能丰富的 PDF 阅读器，包含 Complete PDF viewer demo 中的所有功能。

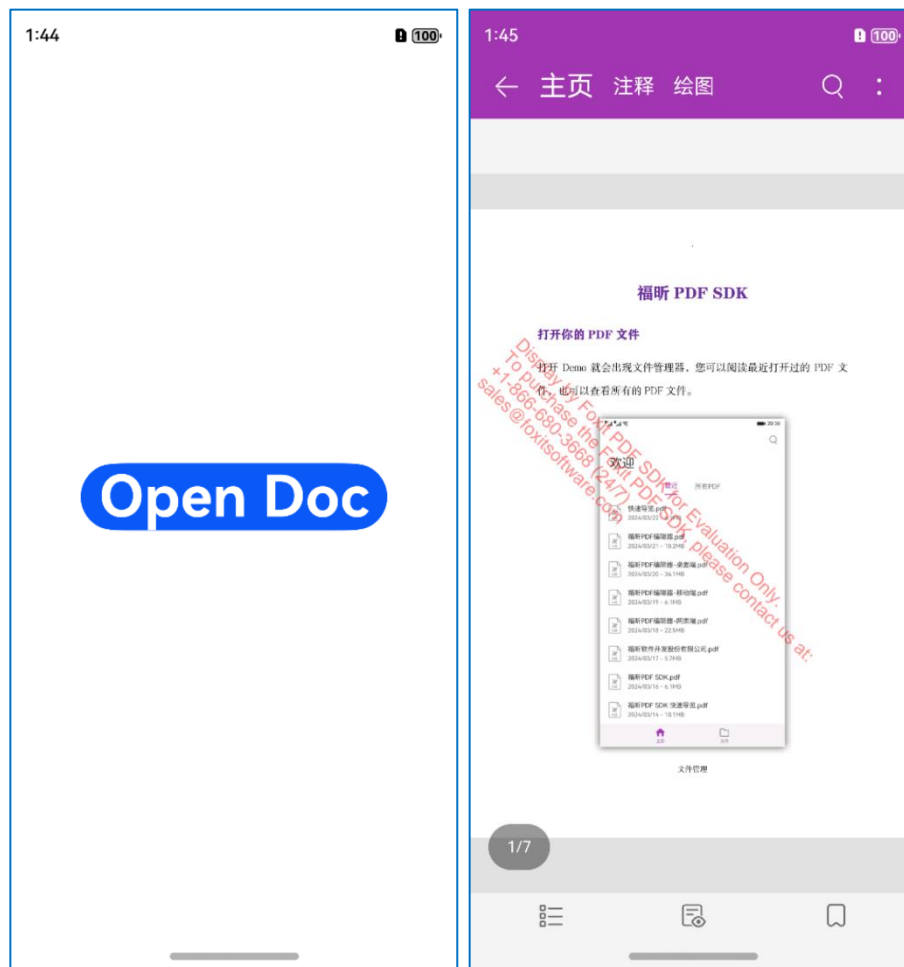


Figure 3-7

4 自定义 UI

Foxit PDF SDK 鸿蒙版为开发人员 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丰富的 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 鸿蒙版提供了 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。此外，1.0 版本提供了一个配置文件 (uiextensions_config.json)，可以帮助开发人员自定义内置 UI 中的任何元素。

以下部分将介绍如何通过配置文件自定义功能模块、权限管理和 UI 元素。

4.1 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。

4.1.1 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 鸿蒙版包中

"samples\complete_pdf_viewer\entry\src\main\resources\rawfile" 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,
      "typewriter": true,
      "textbox": true,
      "callout": true,
    }
  }
}
```

```
"note": true,
"polygon": true,
"cloud": true,
"polyline": true,
"image": true
},
"search": true,
"pagenavigation": true
},
"permissions": {
  "runJavaScript": true
},
"uiSettings": {
  "pageMode": "Single",
  "continuous": false,
  "zoomMode": "FitWidth",
  "colorMode": "Normal",
  "enableFormNavigationBar": true,
  "highlightForm": true,
  "highlightFormColor": "#200066cc",
  "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, "fillColor": null
    },
    "oval": {
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null
    }
  }
}
```



```
    },  
    "arrow": {  
      "color": "#ff0000", "opacity": 1.0, "thickness": 2  
    },  
    "polygon": {  
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null  
    },  
    "cloud": {  
      "color": "#ff0000", "opacity": 1.0, "thickness": 2, "fillColor": null  
    },  
    "polyline": {  
      "color": "#ff0000", "opacity": 1.0, "thickness": 2  
    },  
    "typewriter": {  
      "textColor": "#0000ff",  
      "opacity": 1.0,  
      "textFace": "Courier",  
      "textSize": 18  
    },  
    "textbox": {  
      "color": "#ff0000",  
      "textColor": "#0000ff",  
      "opacity": 1.0,  
      "textFace": "Courier",  
      "textSize": 18  
    },  
    "callout": {  
      "color": "#ff0000",  
      "textColor": "#0000ff",  
      "opacity": 1.0,  
      "textFace": "Courier",  
      "textSize": 18  
    },  
    "note": {  
      "color": "#ff0000",  
      "opacity": 1.0,  
      "icon": "Comment"  
    },  
    "image": {  
      "rotation": 0,  
      "opacity": 1.0  
    }  
  }  
}
```

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。

4.1.2 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, typewriter, textbox, callout, note, polygon, cloud, polyline, image) | 注释模块集合 |
| search | 文本搜索 |
| pagenavigation | PDF 页面导航 |

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** 的默认值为 **"true"**。

| 权限管理 | 描述 |
|---------------|-------------------|
| runJavaScript | 是否允许执行 JavaScript |

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 |
|-------------------------|---------------|--------|---|----------|----------------------------|
| pageMode | 页面显示模式 | String | Single/ Facing/ CoverLeft/ CoverMiddle/ CoverRight/ Reflow | Single | |
| 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。 |
| 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。 |
| enableFormNavigationBar | 是否启用 Form 工具类 | Bool | true/false | true | |

| UI 配置子项 | | 描述/属性 | 值类型 | 可选值 | 默认值 | 备注 |
|--------------------|-----------------|-----------|---------|------------|-----------|-----------------------------|
| 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。 |
| annotations | continuouslyAdd | | Bool | true/false | true | 是否连续添加某个注释 |
| | highlight | color | RGB | | #ffff00 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | areaHighlight | color | RGB | | #ffff00 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | 如果区域超出页面，则使用默认的配置。 |
| | 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 | |
| | | 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 | |
| | | fillColor | RGB | | null | |
| | oval | color | RGB | | #ff0000 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | thickness | numeric | [1-12] | 2 | |
| | | fillColor | RGB | | null | |
| | arrow | color | RGB | | #ff0000 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | thickness | numeric | [1-12] | 2 | |
| | polygon | color | RGB | | #ff0000 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | thickness | numeric | [1-12] | 2 | |
| | | fillColor | RGB | | null | |
| | cloud | color | RGB | | #ff0000 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | thickness | numeric | [1-12] | 2 | |
| | | fillColor | RGB | | null | |
| | polyline | color | RGB | | #ff0000 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |

| UI 配置子项 | | 描述/属性 | 值类型 | 可选值 | 默认值 | 备注 |
|---------|------------|-----------|---------|---|---------|--------------------------------------|
| | typewriter | thickness | numeric | [1-12] | 2 | |
| | | textColor | RGB | | #0000ff | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | 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 | 18 | |
| | textbox | color | RGB | | #ff0000 | |
| | | textColor | RGB | | #0000ff | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | 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 | 18 | |
| | callout | color | RGB | | #ff0000 | |
| | | textColor | RGB | | #0000ff | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | 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 | 18 | |
| | note | 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。 |
| | | | | | | |
| | image | rotation | numeric | 0/90/180/270 | 0 | |
| | | opacity | numeric | [0.0-1.0] | 1.0 | |
| | | textSize | Integer | >=1 | 12 | |

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

在 3.6 小节 "使用 UI Extensions 组件构建一个功能丰富的 PDF 阅读器"，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。

备注：在这里，我们假设您已经将名为"uiextensions_config.json"的JSON 文件放到 "PDFReader\entry\src\main\resources\rawfile"文件夹下。

在 "PDFReader/entry/src/main/ets/pages/Index.ets" 中，加入以下代码：

```
import { UIExtensionsManager, UIExtensionsComponent, ConfigModel } from 'uiextensions';
...

// @State uiextensionsManager: UIExtensionsManager = new UIExtensionsManager(getContext(this));
@State uiextensionsManager: UIExtensionsManager = new UIExtensionsManager(getContext(this), new
ConfigModel(getContext(this), 'uiextensions_config.json'));
```

4.1.4 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通过修改配置文件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)。

备注：为了方便起见，我们将在 "samples" 文件夹下的 "**complete_pdf_viewer**" demo 中进行演示。

在 DevEco Studio 中打开 "**complete_pdf_viewer**" demo。在 "samples\complete_pdf_viewer\entry\src\main\resources\rawfile " 文件夹下找到配置文件 "uiextensions_config.json"。

示例 1：禁用 "search" 和 "pagenavigation" 功能模块。

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

```
"search": false,
"pagenavigation": false,
```

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：



"search" 和 "pagenavigation" 功能模块被移除了。

示例 2：不允许执行 JavaScript。

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

```
"permissions": {  
  "runJavaScript": false,  
},
```

然后，重新编译和运行该 demo，当点击带 JS 脚本的按钮时，将没有任何响应。

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：

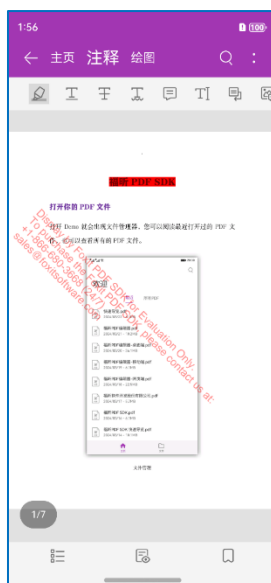
```
"highlight": {  
  "color": "#ff0000", "opacity": 1.0 },
```

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 使用 SDK API

Foxit PDF SDK 鸿蒙版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 鸿蒙版的主要功能，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。

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

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

- 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 `FoxitRDKNative.Filler` 对象来填表，则应使用 `FoxitRDKNative.Render` 接口来渲染当前获取到焦点的表单控件，而不是使用 `Renderer.RenderAnnot` 接口。

5.1.1 如何将指定的 PDF 页面渲染到 bitmap

```
import { image } from '@kit.ImageKit';
import { FoxitRDKNative } from 'foxit_rdk';
import { BusinessError } from '@ohos.base';

class RenderUnit {
  public pixelMap?: image.PixelMap;

  public renderPage(page: FoxitRDKNative.PDFPage, width: number, height: number): void {

    try {
      if (!page.IsParsed()) {
```

```
class PauseCallBackImpl extends FoxitRDKNative.PauseCallback {
  NeedToPauseNow(): boolean {
    return false;
  }
}

const progressive = page.StartParse(FoxitRDKNative.PDFPage.e_ParsePageNormal, new
PauseCallBackImpl(), false)
let state = FoxitRDKNative.Progressive.e_ToBeContinued;
while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
  state = progressive.Continue();
}

const w: number = Math.ceil(width);
const h: number = Math.ceil(height);
const size: number = w * h * 4;

const matrix: FoxitRDKNative.Matrix2D = page.GetDisplayMatrix(0, 0, w, h, FoxitRDKNative.e_Rotation0)
const color: ArrayBuffer = new ArrayBuffer(size)
const bitmap: FoxitRDKNative.Bitmap =
  new FoxitRDKNative.Bitmap(w, h, FoxitRDKNative.Bitmap.e_DIBArgb, color, w * 4)
bitmap.FillRect(0xFFFFFFFF)
const render: FoxitRDKNative.Renderer = new FoxitRDKNative.Renderer(bitmap, false);

class PauseCallBackImpl extends FoxitRDKNative.PauseCallback {
  NeedToPauseNow(): boolean {
    return false;
  }
}

const progressive = render.StartRender(page, matrix, new PauseCallBackImpl());
let state = FoxitRDKNative.Progressive.e_ToBeContinued;
while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
  state = progressive.Continue();
}
let opts: image.InitializationOptions =
  { editable: false, pixelFormat: image.PixelMapFormat.RGBA_8888, size: { height: h, width: w } }
image.createPixelMap(color, opts, (error: BusinessError, pixelMap: image.PixelMap) => {
  if (error) {
    console.error('Failed to create pixelMap.');
```

return;

```
  }

  this.pixelMap = pixelMap; // 使用 Image 或者其他组件将 pixelMap 加载到布局上面
  console.log('Succeeded in creating pixelMap.');
```

```
})
} catch (e) {
  console.error(e);
}
```



```
}  
}
```

5.2 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 `TextPage` 对象中。`TextPage` 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 `TextPage` 对象来构建 `TextSearch` 对象。
- 访问类似超文本链接的文本，使用 `TextPage` 对象来构建 `PageTextLinks` 对象。
- 高亮 PDF 页面上的选中文本，构建一个 `TextPage` 对象来计算选中文本区域。

5.2.1 如何通过选择获取页面上的文本区域

```
import { FoxitRDKNative } from 'foxit_rdk';  
  
class TextPageUnit {  
    public getTextRectsBySelection(page: FoxitRDKNative.PDFPage, startPos: FoxitRDKNative.PointF,  
        endPos: FoxitRDKNative.PointF): Array<FoxitRDKNative.RectF> | null {  
        try {  
            if (!page.IsParsed()) {  
                class PauseCallbackImpl extends FoxitRDKNative.PauseCallback {  
                    NeedToPauseNow(): boolean {  
                        return false;  
                    }  
                }  
  
                const progressive = page.StartParse(FoxitRDKNative.PDFPage.e_ParsePageNormal, new  
                    PauseCallbackImpl(), false)  
                let state = FoxitRDKNative.Progressive.e_ToBeContinued;  
                while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {  
                    state = progressive.Continue();  
                }  
            }  
  
            // 从解析的 PDF 页面创建一个 text page  
            const textPage = new FoxitRDKNative.TextPage(page, FoxitRDKNative.TextPage.e_ParseTextNormal);  
            if (textPage == null || textPage.IsEmpty()) {  
                return null;  
            }  
            let startCharIndex = textPage.GetIndexAtPos(startPos.x, startPos.y, 5);  
            let endCharIndex = textPage.GetIndexAtPos(endPos.x, endPos.y, 5);  
            // getTextRectCount API 要求开始字符索引必须小于或等于结束字符索引  
            startCharIndex = startCharIndex < endCharIndex ? startCharIndex : endCharIndex;  
            endCharIndex = endCharIndex > startCharIndex ? endCharIndex : startCharIndex;  
            const count = textPage.GetTextRectCount(startCharIndex, endCharIndex - startCharIndex);
```

```
if (count > 0) {
    const array: Array<FoxitRDKNative.RectF> = new Array();
    for (let i = 0; i < count; i++) {
        const rectF = textPage.GetTextRect(i);
        if (rectF == null || rectF.IsEmpty()) {
            continue;
        }
        array.push(rectF);
    }
    // 返回的矩形是以 PDF 单位表示的，如果调用者需要在屏幕上高亮显示这些文本矩形，那么首先应该将这些矩形
    // 转换为设备单位
    return array;
}
} catch (e) {
    console.error(e);
}
return null;
}
```

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

5.3.1 如何在 PDF 文档中搜索指定的文本模型

```
import { FoxitRDKNative } from 'foxit_rdk';

class TextSearchUnit {
    private searchText() {
        try {
            const pdfpath = "XXX/Sample.pdf";
            const doc = new FoxitRDKNative.PDFDoc(pdfpath);
            doc.Load("");

            class SearchCancelCallbackImpl extends FoxitRDKNative.SearchCancelCallback {
                NeedToCancelNow(): boolean {
                    return false;
                }
            }

            // 创建一个 text search handler
            const textSearch =
```

```
new FoxitRDKNative.TextSearch(doc, new SearchCancelCallbackImpl(),
FoxitRDKNative.TextPage.e_ParseTextNormal);
// 设置搜索开始的起始页索引。默认情况下，结束页将是最后一页
textSearch.SetStartPage(0);
// 设置要搜索的文本
textSearch.SetPattern("foxit");
// 设置搜索标志以匹配大小写和整个单词。
textSearch.SetSearchFlags(FoxitRDKNative.TextSearch.e_SearchMatchCase |
FoxitRDKNative.TextSearch.e_SearchMatchWholeWord);
while (textSearch.FindNext()) {
    // 如果为true，则找到了一个匹配项
    // 获取找到的匹配项的页面索引
    const pageIndex = textSearch.GetMatchPageIndex();
    // 获取找到的匹配项文本的起始字符索引
    const startCharIndex = textSearch.GetMatchStartCharIndex();
    // 获取找到的匹配项文本的结束字符索引
    const endCharIndex = textSearch.GetMatchEndCharIndex();
    // 获取找到的匹配项文本的矩形区域
    const matchRects: FoxitRDKNative.RectFArray = textSearch.GetMatchRects();
}
} catch (e) {
    console.error(e);
}
}
```

5.4 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

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

5.4.1 如何使用深度优先顺序遍历 PDF 文档的 bookmarks

```
import { FoxitRDKNative } from 'foxit_rdk';

class BookmarkUnit {
  private depthFistTravelBookmarkTree(bookmark: FoxitRDKNative.Bookmark, doc: FoxitRDKNative.PDFDoc):
void {
  if (bookmark == null || bookmark.IsEmpty()) {
    return;
  }
  try {
    this.depthFistTravelBookmarkTree(bookmark.GetFirstChild(), doc);
    while (true) {
      // 获取书签标题
      const title = bookmark.GetTitle();
      const dest = bookmark.GetDestination();
      if (dest != null && !dest.IsEmpty()) {
        let left: number, right: number, top: number, bottom: number;
        let zoom: number;
        const pageIndex = dest.GetPageIndex(doc);
        // left,right,top,bottom,zoom 只在一些特殊的缩放模式下有意义
        const mode = dest.GetZoomMode();
        switch (mode) {
          case FoxitRDKNative.Destination.e_ZoomXYZ:
            left = dest.GetLeft();
            top = dest.GetTop();
            zoom = dest.GetZoomFactor();
            break;
          case FoxitRDKNative.Destination.e_ZoomFitPage:
            break;
          case FoxitRDKNative.Destination.e_ZoomFitHorz:
            top = dest.GetTop();
            break;
          case FoxitRDKNative.Destination.e_ZoomFitVert:
            left = dest.GetLeft();
            break;
          case FoxitRDKNative.Destination.e_ZoomFitRect:
            left = dest.GetLeft();
            bottom = dest.GetBottom();
            right = dest.GetRight();
            top = dest.GetTop();
            break;
          case FoxitRDKNative.Destination.e_ZoomFitBBox:
            break;
          case FoxitRDKNative.Destination.e_ZoomFitBHorz:
            top = dest.GetTop();
            break;
          case FoxitRDKNative.Destination.e_ZoomFitBVert:
            left = dest.GetLeft();
            break;
          default:

```

```

        break;
    }
}
bookmark = bookmark.GetNextSibling();
if (bookmark == null || bookmark.IsEmpty()) {
    break;
}
this.depthFistTravelBookmarkTree(bookmark.GetFirstChild(), doc);
}
} catch (e) {
    console.error(e);
}
}
}
}

```

5.5 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 `PDFDoc.GetReadingBookmarkCount` 接口来计算其个数，并且可以调用 `PDFDoc.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。

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

```

import { FoxitRDKNative } from 'foxit_rdk';

class ReadingBookmarkUnit {
    private addReadingBookmark(pdfDoc: FoxitRDKNative.PDFDoc, title: string,
        pageIndex: number): FoxitRDKNative.ReadingBookmark {
        const count = pdfDoc.GetReadingBookmarkCount();
        return pdfDoc.InsertReadingBookmark(count, title, pageIndex);
    }

    // 枚举 PDF 文档中所有的 reading bookmarks
    private getReadingBookmark(pdfDoc: FoxitRDKNative.PDFDoc): void {
        try {
            const count = pdfDoc.GetReadingBookmarkCount();
            for (let i = 0; i < count; i++) {
                const readingBookmark = pdfDoc.GetReadingBookmark(i);
                if (readingBookmark.IsEmpty()) {
                    continue;
                }
                // 获取 reading bookmark 的标题
                const title = readingBookmark.GetTitle();
            }
        }
    }
}

```

```
// 获取与 reading bookmark 关联的页面索引
const pageIndex = readingBookmark.GetPageIndex();
// 获取 reading bookmark 的创建日期
const creationTime = readingBookmark.GetDateTime(true);
// 获取 reading bookmark 的修改日期
const modificationTime = readingBookmark.GetDateTime(false);
}
} catch (e) {
console.error(e);
}
}
}
```

5.6 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。

5.6.1 如何将指定文件嵌入到 PDF 文档

```
import { FoxitRDKNative } from 'foxit_rdk';

class AttachmentUnit {
private addFileAttachment(): void {
try {
const pdfpath = "XXX/Sample.pdf";
const doc = new FoxitRDKNative.PDFDoc(pdfpath);
doc.Load("");

// 将指定文件嵌入到 PDF 文档中
const filePath = "/xxx/fileToBeEmbedded.xxx";
const nameTree = new FoxitRDKNative.PDFNameTree(doc,
FoxitRDKNative.PDFNameTree.e_EmbeddedFiles);
const attachments = new FoxitRDKNative.Attachments(doc, nameTree);
const fileSpec = new FoxitRDKNative.FileSpec(doc);
fileSpec.SetFileName(filePath);
if (!fileSpec.Embed(filePath)) {
return;
}
attachments.AddEmbeddedFile(filePath, fileSpec);
} catch (e) {
console.error(e);
}
}
}
```

5.6.2 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 { FoxitRDKNative } from 'foxit_rdk';

class AttachmentUnit2 {
  private exportAttachment(): void {
    try {
      const pdfpath = "XXX/Sample.pdf";
      const doc = new FoxitRDKNative.PDFDoc(pdfpath);
      doc.Load("");

      const nameTree = new FoxitRDKNative.PDFNameTree(doc,
FoxitRDKNative.PDFNameTree.e_EmbeddedFiles);
      const attachments = new FoxitRDKNative.Attachments(doc, nameTree);
      // 提取嵌入的附件文件
      const count = attachments.GetCount();
      for (let i = 0; i < count; i++) {
        const key: string = attachments.GetKey(i);
        if (key != null && key.length > 0) {
          const fileSpec1 = attachments.GetEmbeddedFile(key);
          const exportedFile = "/somewhere/" + fileSpec1.GetFileName();
          if (fileSpec1 != null && !fileSpec1.IsEmpty()) {
            fileSpec1.ExportToFile(exportedFile);
          }
        }
      }
    } catch (e) {
      console.error(e);
    }
  }
}
```

5.7 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 5-1 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 5-1 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 5-1

| 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 |

| Annotation type | Description | Markup | Supported by SDK |
|-----------------|---------------------------|--------|------------------|
| 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 PDF 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。

5.7.1 如何向 PDF 页面中添加注释

```
import { FoxitRDKNative } from 'foxit_rdk';

class AnnotationUnit {
    private addAnnot(): void {
        try {
            const pdfpath = "xxx/Sample.pdf";
            const doc = new FoxitRDKNative.PDFDoc(pdfpath);
            doc.Load("");
            const pdfPage = doc.GetPage(1);
            const rect = new FoxitRDKNative.RectF(100, 100, 120, 120);
            const note = new FoxitRDKNative.Note(pdfPage.AddAnnot(FoxitRDKNative.Annot.e_Note, rect));
        }
    }
}
```



```
if (note == null || note.IsEmpty()) {
    return;
}
note.SetIconName("Comment");
// 将边框颜色设置为蓝色
note.SetBorderColor(0xff0000ff);
note.SetContent("This is the note comment, write any content here.");
note.ResetAppearanceStream();

class SearchCancelCallbackImpl extends FoxitRDKNative.SearchCancelCallback {
    NeedToCancelNow(): boolean {
        return false;
    }
}
// 以下代码展示如何在搜索到的文本上添加高亮注释
const textSearch = new FoxitRDKNative.TextSearch(pdfPage.GetDocument(), new
SearchCancelCallbackImpl(),
    FoxitRDKNative.TextPage.e_ParseTextNormal);
if (textSearch == null || textSearch.IsEmpty()) {
    return;
}
// 假设需要高亮的文本是 "foxit"
textSearch.SetPattern("foxit");
const bMatched = textSearch.FindNext();
if (bMatched) {
    const rects = textSearch.GetMatchRects();
    const rectCount = rects.GetSize();
    // 根据搜索到的文本矩形，填充 quadpoints 数组。
    const arrayOfQuadPoints = new FoxitRDKNative.QuadPointsArray();
    let matchRect: FoxitRDKNative.RectF;
    for (let i = 0; i < rectCount; i++) {
        matchRect = rects.GetAt(i);
        const quadPoints = new FoxitRDKNative.QuadPoints();
        quadPoints.first = new FoxitRDKNative.PointF(matchRect.left, matchRect.top);
        quadPoints.second = new FoxitRDKNative.PointF(matchRect.right, matchRect.top);
        quadPoints.third = new FoxitRDKNative.PointF(matchRect.left, matchRect.bottom);
        quadPoints.fourth = new FoxitRDKNative.PointF(matchRect.right, matchRect.bottom);
        arrayOfQuadPoints.Add(quadPoints);
    }
    // 只需给 markup annotation 设置一个空矩形，然后根据 quadPoints 的坐标值来计算 annotation 矩形
    const rect2 = new FoxitRDKNative.RectF(0, 0, 0, 0);
    const textMarkup = new
FoxitRDKNative.TextMarkup(pdfPage.AddAnnot(FoxitRDKNative.Annot.e_Highlight, rect2));
    // 将 quadpoints 设置给 markup annotation
    textMarkup.SetQuadPoints(arrayOfQuadPoints);
    // 将边框颜色设置为红色
    textMarkup.SetBorderColor(0xffff0000);
    // 设置为 30% 的透明度
    textMarkup.SetOpacity(0.3);
    // 生成外观
    textMarkup.ResetAppearanceStream();
}
```

```
    }  
  } catch (e) {  
    console.error(e);  
  }  
}  
}
```

5.7.2 如何删除 PDF 页面中的注释

```
import { FoxitRDKNative } from 'foxit_rdk';  
  
class AnnotationUnit2 {  
  private removeAnnot(): void {  
    try {  
      const pdfpath = "xxx/Sample.pdf";  
      const doc = new FoxitRDKNative.PDFDoc(pdfpath);  
      doc.Load("");  
      const pdfPage = doc.GetPage(1);  
      const annot = pdfPage.GetAnnot(0);  
      if (annot == null || annot.IsEmpty()) {  
        return;  
      }  
      // 删除第一个 annot, 使第二个 annot 变为第一个  
      pdfPage.RemoveAnnot(annot);  
    } catch (e) {  
      console.error(e)  
    }  
  }  
}
```

5.7.3 如何注册监听器接收注释事件

在接收注释事件之前需要提前注册注释监听器。具体参考下述代码。

```
import { FoxitRDKNative } from 'foxit_rdk';  
import { UIExtensionsManager } from 'uiextensions';  
import { IAnnotEventListener } from 'uiextensions/src/main/ets/event/IAnnotEventListener';  
  
class AnnotationUnit3 {  
  private annotEventListener: IAnnotEventListener = {  
  
    onAnnotAdded: (page: FoxitRDKNative.PDFPage, annot: FoxitRDKNative.Annot): void => {  
    },  
  
    onAnnotWillDelete: (page: FoxitRDKNative.PDFPage, annot: FoxitRDKNative.Annot): void => {  
    },  
  
    onAnnotDeleted: (page: FoxitRDKNative.PDFPage, annot: FoxitRDKNative.Annot): void => {  
    },  
  
    onAnnotModified: (page: FoxitRDKNative.PDFPage, annot: FoxitRDKNative.Annot): void => {  
    }  
  }  
}
```

```

    },

    onAnnotChanged: (lastAnnot: FoxitRDKNative.Annot | null, currentAnnot: FoxitRDKNative.Annot | null): void
=> {
    }

}

registerYourAnnotEventListener(extensionsManager: UIExtensionsManager): void {
    // 调用registerAnnotEventListener 来注册 annot 事件监听器以接收 annot 事件
    extensionsManager.getDocumentManager().registerAnnotEventListener(this.annotEventListener);
}

unregisterYourAnnotEventListener(extensionsManager: UIExtensionsManager): void {
    // 调用unregisterAnnotEventListener 来取消注册 annot 事件监听器
    extensionsManager.getDocumentManager().unregisterAnnotEventListener(this.annotEventListener);
}
}

```

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

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

5.8.1 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 { FoxitRDKNative } from 'foxit_rdk';

class FormUnit {
    private formTest(): void {
        try {
            const pdfpath = "xxx/Sample.pdf";
            const doc = new FoxitRDKNative.PDFDoc(pdfpath);
            doc.Load("");

            const hasForm = doc.HasForm();
            if (hasForm) {
                // 从文档中创建一个form 对象
                const form = new FoxitRDKNative.Form(doc);
            }
        }
    }
}

```

```
// 将表单数据导出到一个XML 文件
form.ExportToXML("/somewhere/export.xml");
// 或者从XML 文件中导入表单数据
form.ImportFromXML("/somewhere/export.xml");
}
} catch (e) {
    console.error(e);
}
}
}
```

5.9 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。

5.9.1 如何使用密码加密 PDF 文件

```
import { FoxitRDKNative } from 'foxit_rdk';

class SecurityUnit {
    public encryptPDF(pdfDoc: FoxitRDKNative.PDFDoc, ownerPassword: string, userPassword: string,
        savePth: string): boolean {

        // 设置加密数据。
        const encryptData = new FoxitRDKNative.StdEncryptData(true,
            FoxitRDKNative.PDFDoc.e_PermModify | FoxitRDKNative.PDFDoc.e_PermAssemble |
            FoxitRDKNative.PDFDoc.e_PermFillForm,
            FoxitRDKNative.SecurityHandler.e_CipherAES, 16);

        const securityHandler = new FoxitRDKNative.StdSecurityHandler();
        try {
            if (!securityHandler.Initialize(encryptData, userPassword, ownerPassword)) {
                return false;
            }
            pdfDoc.SetSecurityHandler(securityHandler);
            if (!pdfDoc.SaveAs(savePth, FoxitRDKNative.PDFDoc.e_SaveFlagNormal)) {
                return false;
            }
            return true;
        } catch (e) {
            console.error(e);
        }
        return false;
    }
}
```

5.10 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。

5.10.1 如何对 PDF 文档进行签名，并验证签名

```
import { FoxitRDKNative } from 'foxit_rdk';

class SignatureUnit {
    // 示例代码展示了如何对 PDF 文档进行签名，并验证签名
    public addNewSignatureAndSign(page: FoxitRDKNative.PDFPage, rect: FoxitRDKNative.RectF): void {
        try {
            // 在指定的页面矩形上添加一个新的签名
            const signature = page.AddSignature(rect);
            // 设置 appearance flags
            signature.SetAppearanceFlags(FoxitRDKNative.Signature.e_APFlagLabel |
FoxitRDKNative.Signature.e_APFlagDN |
FoxitRDKNative.Signature.e_APFlagText | FoxitRDKNative.Signature.e_APFlagLocation |
FoxitRDKNative.Signature.e_APFlagReason | FoxitRDKNative.Signature.e_APFlagSigner);
            // 设置 signer
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameSigner, "Foxit");
            // 设置 location
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameLocation, "Anywhere");
            // 设置 reason
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameReason, "AnyReason");
            // 设置 contact info
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameContactInfo, "AnyInfo");
            // 设置 domain name
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameDN, "AnyDN");
            // 设置 description
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameText, "AnyContent");
            // 默认支持 "Adobe.PPKLite" Filter
            signature.SetFilter("Adobe.PPKLite");
            // 默认支持 "adbe.pkcs7.sha1" 或 "adbe.pkcs7.detached" SubFilter
            signature.SetSubFilter("adbe.pkcs7.detached");

            // 输入的 PKCS #12 格式证书，其包含公钥和私钥
```

```
const certPath = "/somewhere/cert.pfx";
// 该证书的密码
const certPassword = "123";
const signedPDFPath = "/somewhere/signed.pdf";

// 开始签名，如果成功，签名后的 PDF 将保存到 "save_path" 指定的路径
class PauseCallBackImpl extends FoxitRDKNative.PauseCallback {
    NeedToPauseNow(): boolean {
        return false;
    }
}

const progressive =
    signature.StartSign(certPath, certPassword, FoxitRDKNative.Signature.e_DigestSHA1, signedPDFPath,
        new ArrayBuffer(0), new PauseCallBackImpl());
if (progressive != null) {
    let state = FoxitRDKNative.Progressive.e_ToBeContinued;
    while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
        state = progressive.Continue();
    }
    if (state != FoxitRDKNative.Progressive.e_Finished) {
        return;
    }
}

// 从已签名的 PDF 文档中获取签名，然后进行验证
const pdfDoc = new FoxitRDKNative.PDFDoc(signedPDFPath);
const err = pdfDoc.Load(null);
if (err != FoxitRDKNative.ErrorCode.e_ErrSuccess) {
    return;
}
const count = pdfDoc.GetSignatureCount();
for (let i = 0; i < count; i++) {
    const sign = pdfDoc.GetSignature(i);
    if (sign != null && !sign.IsEmpty()) {
        const progressive_1 = sign.StartVerify(new ArrayBuffer(0), null);
        if (progressive_1 != null) {
            let state = FoxitRDKNative.Progressive.e_ToBeContinued;
            while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
                state = progressive_1.Continue();
            }
            if (state != FoxitRDKNative.Progressive.e_Finished) {
                continue;
            }
        }
        const verifiedState = sign.GetState();
        if ((verifiedState & FoxitRDKNative.Signature.e_StateVerifyValid) ==
            FoxitRDKNative.Signature.e_StateVerifyValid) {
            console.info("Signature", "addNewSignatureAndSign: Signature" + i + "is valid.");
        }
    }
}
```

```
}  
} catch (e) {  
    console.error(e);  
}  
}  
}
```

5.10.2 如何为签名设置定制化时间信息

通过 Signature 对象的 SetSignTime 方法不能改变时间格式。我们可以通过传递参数（时间字符串）给签名词典来解决这个问题。具体参考下述代码。

```
import { FoxitRDKNative } from 'foxit_rdk';  
  
class SignatureUnit2 {  
    public addNewSignatureAndSign(page: FoxitRDKNative.PDFPage, rect: FoxitRDKNative.RectF): void {  
        try {  
            // 在指定的页面矩形上添加一个新的签名  
            const signature = page.AddSignature(rect);  
            // 设置 appearance flags  
            signature.SetAppearanceFlags(FoxitRDKNative.Signature.e_APFlagLabel |  
FoxitRDKNative.Signature.e_APFlagDN |  
FoxitRDKNative.Signature.e_APFlagText | FoxitRDKNative.Signature.e_APFlagLocation |  
FoxitRDKNative.Signature.e_APFlagReason | FoxitRDKNative.Signature.e_APFlagSigner |  
FoxitRDKNative.Signature.e_APFlagSigningTime);  
            // 设置 signer  
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameSigner, "Foxit");  
            // 设置 location  
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameLocation, "Anywhere");  
            // 设置 reason  
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameReason, "AnyReason");  
            // 设置 contact info  
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameContactInfo, "AnyInfo");  
            // 设置 domain name  
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameDN, "AnyDN");  
            // 设置 description  
            signature.SetKeyValue(FoxitRDKNative.Signature.e_KeyNameText, "AnyContent");  
            // DateTime dateTime = new DateTime();  
            // ...  
            // signature.SetSignTime(dateTime);  
            // 签名日期的默认格式是 yyMMddhhmmss-时区  
            // 如果您需要设置自定义格式的时间，请参考以下代码  
            const dictionary = signature.GetSignatureDict();  
            dictionary.SetAtString("M", "2022/02/13 11:00:00" /* formatted time string */);  
            // 默认支持 "Adobe.PPKLite" Filter  
            signature.SetFilter("Adobe.PPKLite");  
            // 默认支持 "adbe.pkcs7.sha1" 或 "adbe.pkcs7.detached" SubFilter  
            signature.SetSubFilter("adbe.pkcs7.detached");  
  
            // 输入的 PKCS #12 格式证书，其包含公钥和私钥
```

```
const certPath = "/somewhere/cert.pfx";
// 该证书的密码
const certPassword = "123";
const signedPDFPath = "/somewhere/signed.pdf";

// 开始签名，如果成功，签名后的 PDF 将保存到 "save_path" 指定的路径
class PauseCallBackImpl extends FoxitRDKNative.PauseCallback {
    NeedToPauseNow(): boolean {
        return false;
    }
}

const progressive =
    signature.StartSign(certPath, certPassword, FoxitRDKNative.Signature.e_DigestSHA1, signedPDFPath,
        new ArrayBuffer(0), new PauseCallBackImpl());
if (progressive != null) {
    let state = FoxitRDKNative.Progressive.e_ToBeContinued;
    while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
        state = progressive.Continue();
    }
    if (state != FoxitRDKNative.Progressive.e_Finished) {
        return;
    }
}

// 从已签名的 PDF 文档中获取签名，然后进行验证
const pdfDoc = new FoxitRDKNative.PDFDoc(signedPDFPath);
const err = pdfDoc.Load("");
if (err != FoxitRDKNative.ErrorCode.e_ErrSuccess) {
    return;
}
const count = pdfDoc.GetSignatureCount();
for (let i = 0; i < count; i++) {
    const sign = pdfDoc.GetSignature(i);
    if (sign != null && !sign.IsEmpty()) {
        const progressive_1 = sign.StartVerify(null, null);
        if (progressive_1 != null) {
            let state = FoxitRDKNative.Progressive.e_ToBeContinued;
            while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
                state = progressive_1.Continue();
            }
            if (state != FoxitRDKNative.Progressive.e_Finished) {
                continue;
            }
        }
        const verifiedState = sign.GetState();
        if ((verifiedState & FoxitRDKNative.Signature.e_StateVerifyValid) ==
            FoxitRDKNative.Signature.e_StateVerifyValid) {
            console.info("Signature", "addNewSignatureAndSign: Signature" + i + "is valid.");
        }
    }
}
```



```
    }  
  } catch (e) {  
    console.error(e);  
  }  
}  
}
```

6 FAQ

6.1 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？

在构造 PDFDoc 时，只能使用绝对路径。由于 HarmonyOS Next 的沙盒机制，您无法访问应用外文件的真实路径。因此，您需要先将文件拷贝到应用内部，然后再打开。请参考如下的代码：

```
@Entry
@Component
struct Index {
  @State isShowDoc: boolean = false;
  @State pdfViewModel: PDFViewCtrlModel = new PDFViewCtrlModel(getContext());
  private appFilesDir = getContext().filesDir;

  build() {
    Column() {
      if (!this.isShowDoc) {
        Button('Open Doc')
          .onClick(async ()=>{
            const filePath = await this.selectFile();
            if(filePath != null) {
              this.isShowDoc = true;
              this.pdfViewModel.openDoc(filePath);
            }
          })
      } else {
        PDFViewCtrl({ model: this.pdfViewModel })
      }
    }
    .width('100%')
    .height('100%')
    .justifyContent(FlexAlign.Center)
  }

  async selectFile(): Promise<string | null> {
    try {
      let documentSelectOptions = new picker.DocumentSelectOptions();
      // documentSelectOptions.fileSuffixFilters = suffix;
      documentSelectOptions.maxSelectNumber = 10;
      if (canIUse('SystemCapability.FileManagement.UserFileService.FolderSelection')) {
        documentSelectOptions.selectMode = picker.DocumentSelectMode.FILE;
      }

      const documentViewPicker = new picker.DocumentViewPicker(getContext());
```

```
const documentSelectResult: Array<string> = await documentPicker.select(documentSelectOptions);
if (documentSelectResult.length === 0) {
    return null;
}

const importFiles: Array<string> = new Array();
for (let i = 0; i < documentSelectResult.length; i++) {
    let uri = documentSelectResult[i];

    let src_file = fs.openSync(uri, fs.OpenMode.READ_ONLY);
    src_file.path

    const dest_filepath = this.appFilesDir + '/' + src_file.name;
    fs.copyFileSync(src_file.fd, dest_filepath);
    fs.closeSync(src_file.fd);

    importFiles.push(dest_filepath);
}
return importFiles[0];
} catch (error) {
    console.error(error);
}
return null;
}
}
```

6.2 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 **pdfViewModel.gotoPage**。Foxit PDF SDK 鸿蒙版使用多线程来提高渲染速度，因此您需要确保在使用 **pdfViewModel.gotoPage** 接口之前，文档已经被成功加载。

请在 **IDocEventListener** 中实现回调接口，然后在 **onDocOpened** 事件中调用 **gotoPage** 接口。以下是示例代码：

```
import { FoxitRDKNative, IDocEventListener, PDFViewCtrl, PDFViewCtrlModel } from 'foxit_rdk';
import picker from '@ohos.file.picker';
import fs, { Filter, Options } from '@ohos.file.fs';

@Entry
@Component
struct Index {
    @State isShowDoc: boolean = false;
    @State pdfViewModel: PDFViewCtrlModel = new PDFViewCtrlModel(getContext());
    private appFilesDir = getContext().filesDir;

    build() {
```

```
Column() {
  if (!this.isShowDoc) {
    Button('Open Doc')
      .onClick(async () => {
        const filePath = await this.selectFile();
        if (filePath != null) {
          this.isShowDoc = true;
          this.pdfViewModel.openDoc(filePath);
        }
      })
  } else {
    RelativeContainer() {
      PDFViewCtrl({ model: this.pdfViewModel })
        .id('pdfview')
        .alignRules({
          top: { anchor: "__container__", align: VerticalAlign.Top },
          left: { anchor: "__container__", align: HorizontalAlign.Start }
        })
    }

    Column() {
      Button('Goto Last Page')
        .onClick(() => {
          const pageCount = this.pdfViewModel.getPageCount();
          this.pdfViewModel.gotoPage(pageCount - 1);
        })
    }
    .width('100%')
    .margin({ top: 100 })
    .id('ToolAction')
  }
  .width('100%')
  .height('100%')
}

.width('100%')
.height('100%')
.justifyContent(FlexAlign.Center)
}

aboutToAppear(): void {
  this.pdfViewModel.registerDocEventListener(this.docEventListener);
}

aboutToDisappear(): void {
  this.pdfViewModel.unregisterDocEventListener(this.docEventListener);
}

async selectFile(): Promise<string | null> {
  try {
    let documentSelectOptions = new picker.DocumentSelectOptions();
    // documentSelectOptions.fileSuffixFilters = suffix;
```

```
documentSelectOptions.maxSelectNumber = 10;
if (canIUse('SystemCapability.FileManagement.UserFileService.FolderSelection')) {
    documentSelectOptions.selectMode = picker.DocumentSelectMode.FILE;
}

const documentViewPicker = new picker.DocumentViewPicker(getContext());
// await select complete
const documentSelectResult: Array<string> = await documentViewPicker.select(documentSelectOptions);
if (documentSelectResult.length === 0) {
    return null;
}

const importFiles: Array<string> = new Array();
for (let i = 0; i < documentSelectResult.length; i++) {
    let uri = documentSelectResult[i];

    let src_file = fs.openSync(uri, fs.OpenMode.READ_ONLY);
    src_file.path

    const dest_filepath = this.appFilesDir + '/' + src_file.name;
    fs.copyFileSync(src_file.fd, dest_filepath);
    fs.closeSync(src_file.fd);

    importFiles.push(dest_filepath);
}
return importFiles[0];
} catch (error) {
    console.error(error);
}
return null;
}

private docEventListener: IDocEventListener = {
    onDocOpened: (document: FoxitRDKNative.PDFDoc, errCode: number): void => {
        if (errCode == FoxitRDKNative.ErrorCode.e_ErrSuccess) {
            const pageCount = this.pdfViewModel.getPageCount();
            this.pdfViewModel.gotoPage(pageCount - 2);
        }
    }
}
```

6.3 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。

6.4 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 **PDFPage.AddAnnot** 将一个 link 注释添加到指定页面，然后调用 **Action.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：

```
import { FoxitRDKNative } from 'foxit_rdk';

class LinkUnit {
  private addLinkAnnot() {
    try {
      const path = 'xxx/Sample.pdf'
      // 初始化一个 PDFDoc 对象
      const document = new FoxitRDKNative.PDFDoc(path);
      // 加载未加密的文档内容
      document.Load();
      // 获取 PDF 文件的第一页
      const page = document.GetPage(0);

      // 在第一页添加一个 link annotation
      const linkAnnot = new FoxitRDKNative.Link(page.AddAnnot(FoxitRDKNative.Annot.e_Link,
        new FoxitRDKNative.RectF(250, 650, 400, 750)));

      // 创建一个 URI action 并设置 URI
      const uriAction =
        new FoxitRDKNative.URIAction(FoxitRDKNative.Action.Create(document,
        FoxitRDKNative.Action.e_TypeURI));
      uriAction.SetURI("www.foxitsoftware.com");

      // 将 action 设置给 link annotation
      linkAnnot.SetAction(uriAction);
      linkAnnot.ResetAppearanceStream();

      // 保存已添加 link annotation 的文档
      document.SaveAs("xxx/sample_link.pdf", FoxitRDKNative.PDFDoc.e_SaveFlagNormal);
    } catch (e) {
      console.error(e);
    }
  }
}
```

6.5 向 PDF 文档中插入图片

如何向 PDF 文档中插入图片？

有两种方法可以帮助您将图片插入到 PDF 文档中。第一种是调用 **PDFPage.AddImageFromFilePath** 接口。您可以参考如下示例代码，该代码将图片插入到 PDF 文档的首页：

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

```
import { FoxitRDKNative } from 'foxit_rdk';

class ImageUnit {
  //1
  private addImage(): void {
    try {
      const path = "xxx/Sample.pdf";
      // 初始化一个 PDFDoc 对象
      const document = new FoxitRDKNative.PDFDoc(path);
      // 加载未加密的文档内容
      document.Load();
      // 获取 PDF 文件的第一页
      const page = document.GetPage(0);
      // 解析页面
      if (!page.IsParsed()) {
        const parse = page.StartParse(FoxitRDKNative.PDFPage.e_ParsePageNormal, null, false);
        let state = FoxitRDKNative.Progressive.e_ToBeContinued;
        while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
          state = parse.Continue();
        }
      }
      // 在第一页添加一张图片
      page.AddImageFromFilePath("XXX/2.png", new FoxitRDKNative.PointF(20, 30), 60, 50, true)
      // 保存已添加图片的文档
      document.SaveAs("XXX/sample_image.pdf", FoxitRDKNative.PDFDoc.e_SaveFlagNormal);
    } catch (e) {
      console.error(e);
    }
  }
}
```

第二种是使用 **PDFPage.AddAnnot** 接口在指定的页面添加一个 screen 注释，然后将图片设置给刚添加的 screen 注释。您可以参考以下的示例代码，该代码将图片作为 screen 注释插入到 PDF 文件的首页：

```
import { FoxitRDKNative } from 'foxit_rdk';

class ImageUnit {
```

```
private addScreenAnnot():void{
    try {
        const path = "xxx/Sample.pdf";
        // 初始化一个 PDFDoc 对象
        const document = new FoxitRDKNative.PDFDoc(path);
        // 加载未加密的文档内容
        document.Load();
        // 获取 PDF 文件的第一页
        const page = document.GetPage(0);

        // 在第一页添加一个 screen annotation
        const screen = new FoxitRDKNative.Screen(page.AddAnnot(FoxitRDKNative.Annot.e_Screen, new
FoxitRDKNative.RectF(100, 350, 250, 150)));

        // 加载一个本地图片
        const image = new FoxitRDKNative.Image('xxx/test.png');
        screen.SetImage(image, 0, 0);
        screen.ResetAppearanceStream();

        // 保存已添加 screen annotation 的文档
        document.SaveAs("XXX/sample_image.pdf", FoxitRDKNative.PDFDoc.e_SaveFlagNormal);
    } catch (e) {
        console.error(e);
    }
}
```

6.6 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。如果您想要禁用它或者设置高亮颜色，可以通过 JSON 文件或调用 API 进行设置。

备注：如果您需要设置高亮的颜色，请确保表单域高亮的功能是启用的。

通过 JSON 文件

设置 `"highlightForm": false,` 在 PDF 文档中禁用表单域高亮功能。

设置 `"highlightFormColor": "#2000ffcc",` 设置高亮颜色 (输入您需要的颜色值)。

通过调用 API

UIExtensionsManager.enableFormHighlight 接口用来设置是否在 PDF 文档中启用表单域高亮的功能。如果您不需要启用该功能，请将其参数设置为 `"false"`，如下所示：

```
// 假设已经初始化一个 UIExtensionsManager 对象: uiextensionsManager
this.uiextensionsManager.enableFormHighlight (false);
```


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

```
// 假设已经初始化一个 UIExtensionsManager 对象: uiextensionsManager  
this.uiextensionsManager.setFormHighlightColor(0x4b0000ff);
```

6.7 支持全文索引搜索

Foxit PDF SDK 鸿蒙版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 鸿蒙版支持全文索引搜索。

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。

```
DocumentsSource#constructor(directory: string);
```

- 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。

```
FullTextSearch#constructor();  
FullTextSearch#SetDataBasePath(path_of_data_base: string): void;
```

- c) 开始索引文档来源中的 PDF 文档。

```
FullTextSearch#StartUpdateIndex(source: DocumentsSource, pause: PauseCallback, reupdate: boolean):  
Progressive;
```

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对其重新进行索引：

```
FullTextSearch#UpdateIndexWithFilePath(file_path: string): boolean
```

- 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。

```
FullTextSearch#SearchOf(match_string: string, rank_mode: number, callback: SearchCallback): boolean
```

如下是使用全文索引搜索的示例代码：

```
class DocumentsSourceUnit {  
    private testSearch(): void {  
        try {  
            const directory = "A search directory...";  
            const search = new FoxitRDKNative.FullTextSearch();  
            const dbPath = "The path of data base to store the indexed data...";  
            search.SetDataBasePath(dbPath);  
            // 获取文档源信息  
            const source = new FoxitRDKNative.DocumentsSource(directory);
```

```
// 创建一个由用户实现的暂停回调对象，以暂停更新过程
const pauseCallback = new PauseCallbackImpl(30);

// 开始索引文档来源中的 PDF 文档
const progressive = search.StartUpdateIndex(source, pauseCallback, false);
let state = FoxitRDKNative.Progressive.e_ToBeContinued;
while (state == FoxitRDKNative.Progressive.e_ToBeContinued) {
    state = progressive.Continue();
}

// 创建一个回调对象，当找到匹配项时将被调用
const searchCallback = new MySearchCallback();

// 从索引的数据源中搜索指定的关键词
search.SearchOf("looking for this text", FoxitRDKNative.FullTextSearch.e_RankHitCountASC,
searchCallback);
} catch (e) {
    console.error(e);
}
}
```

PauseCallbackImpl 回调的示例代码如下所示：

```
class PauseCallbackImpl extends FoxitRDKNative.PauseCallback {
    private m_milliseconds = 0;
    private m_startTime = 0;

    public constructor(milliseconds: number) {
        super()
        this.m_milliseconds = milliseconds;
        this.m_startTime = systemDateTime.getTime();
    }

    NeedToPauseNow(): boolean {
        if (this.m_milliseconds < 1) {
            return false;
        }

        const diff = systemDateTime.getTime() - this.m_startTime;
        if (diff > this.m_milliseconds) {
            this.m_startTime = systemDateTime.getTime();
            return true;
        } else {
            return false;
        }
    }
}
```

MySearchCallback 回调的示例如下所示：

```
class MySearchCallback extends FoxitRDKNative.SearchCallback {  
  
    public release(): void {  
    }  
  
    public retrieveSearchResult(filePath: string, pageIndex: number, matchResult: string,  
matchStartTextIndex: number,  
matchEndTextIndex: number): number {  
        const s =  
            `Found file is :${filePath} \n Page index is :${pageIndex} Start text index :${matchStartTextIndex} End  
text index :${matchEndTextIndex} \n Match is :${matchResult} \n\n`;  
        console.info('MySearchCallback', 'retrieveSearchResult: ' + s);  
        return 0;  
    }  
}
```

备注:

- Foxit PDF SDK 鸿蒙版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 `NeedToPauseNow` 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 `NeedToPauseNow` 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。

6.8 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，需要首先调用

`PDFViewCtrlModel#setMappingModeBackgroundColor(mappingModeBackgroundColor: number)` 和

`PDFViewCtrlModel#setMappingModeForegroundColor(mappingModeForegroundColor: number)` 接口根据您的需要设置颜色，然后使用 `PDFViewCtrlModel#setColorMode(colorMode: number)` 设置颜色模式。

备注: 如果颜色模式已经设置为

`Renderer.e_ColorModeMapping/Renderer.e_ColorModeMappingGray`，在调用 `PDFViewCtrlModel#setMappingModeBackgroundColor(mappingModeBackgroundColor: number)` 和

PDFViewCtrlModel#setMappingModeForegroundColor(mappingModeForegroundColor: number)接口之后，您仍然需要再次设置。否则，颜色设置可能不会起作用。

上述接口需要在 UI Extensions 组件的源代码中调用，请将 "libs" 文件夹下的 "uiextensions" 工程添加到您的工程中。然后在 "uiextensions/src/main/ets/frame/impl/MainFrame.ets" 文件中定位到 `onValueChanged` 函数，根据您的需要设置颜色。

6.9 使用 UIExtensions 设置只读模式

如何使用 UIExtensions 设置只读模式？

1: 接口定义

在 UIExtensionsManager 提供了以下接口：

```
/**
 * whether the pdf document can be modified
 *
 * @return whether the pdf document can be modified
 */
public isEnabledModification(): boolean

/**
 * Set whether the pdf document can be modified. The default setting allows modification
 *
 * @param isEnabled whether the pdf document can be modified
 */
public enableModification(isEnabled: boolean): void
```

2: 接口使用

默认情况下没有权限控制的文档是可以编辑的。如果需要禁用编辑相关的交互功能，可以通过以下方法设置：

```
import { UIExtensionsManager } from 'uiextensions';

class EnableModificationUnit{
    private uiextensionsManager: UIExtensionsManager;
    constructor(uiextMgr: UIExtensionsManager) {
        this.uiextensionsManager = uiextMgr;
    }

    private test(): void {
        this.uiextensionsManager.enableModification(false);
    }
}
```

6.10 更新页面绑定 (page binding) 支持 RTL (Right-to-Left)

如何自动更新页面绑定来支持 RTL (Right-to-Left) ?

对于大多数语言，我们使用的阅读习惯都是从左往右，这需要在左边缘进行页面绑定。但是，也有一些语言的阅读习惯是从右往左，比如阿拉伯语和希伯来语以及几种东亚文字。在这种情况下，用户最好在右边缘进行页面绑定，页面将从右到左排列（第一页在最右边）。为此，我们做了从右到左的页面布局的适配。

页面绑定与水平滚动一起使用。对于垂直滚动，它仅在启用双页模式时才有效。

以编程方式更新页面绑定

Foxit PDF SDK 鸿蒙版在 `PDF_PAGE_BINDING_EDGE` 枚举类中定义了常量 **LEFT** 和 **RIGHT**:

- **LEFT**: 表示文档顺序从左往右排列
- **RIGHT**: 表示文档顺序从右往左排列

然后，通过如下的方法来更新页面绑定来切换页面布局：

```
import { PDFViewCtrlModel, PDF_PAGE_BINDING_EDGE } from 'foxit_rdk';
import { UIExtensionsManager } from 'uiextensions';

class PageBindingUnit {
  private uiExtensionsManager: UIExtensionsManager;
  private pdfviewCtrl: PDFViewCtrlModel;

  constructor(uiExtMgr: UIExtensionsManager) {
    this.uiExtensionsManager = uiExtMgr;
    this.pdfviewCtrl = this.uiExtensionsManager.getPDFViewCtrlModel();
  }

  private test(): void {
    this.pdfviewCtrl.setPageBinding(PDF_PAGE_BINDING_EDGE.RIGHT);
  }
}
```

6.11 打开网页 PDF 文件的问题

如何解决使用 `PDFViewCtrlModel#(url: string, password?: string, cacheOption?: FoxitRDKNative.CacheOption, requestOptions?: Record<string, string>, cb?: (errorCode: FoxitRDKNative.ErrorCode) => void)` 接口打开网页 PDF 文件时，一些文档可能无法正确显示或根本无法显示的问题？

这个问题发生在一些带有大量对象的文档在加载完成之前就被关闭，导致缓存的数据不完整。在尝试加载数据时，当前没有方法来判定数据的有效性，只能判定该数据是否已经被缓存了。

为了解决这个问题，建议用户清除缓存数据，并使用以下方法重新加载文档。这种方法不会影响文档的打开速度，因为 SDK 内部是根据页码来加载网页的。

```
/**
 * Clear the cache file by the specified url.
 *
 * @param url The url of the document that used in {@link PDFViewCtrlModel.openDocFromUrl}
 * @see openDocFromUrl(string, string, CacheOption, Record<string, string>, (errorCode: FoxitRDKNative.ErrorCode) =>
void)
 */
public clearCacheFile(url: string): void

/**
 * Clear all the cache files.
 *
 * @see openDocFromUrl(string, string, CacheOption, Record<string, string>, (errorCode: FoxitRDKNative.ErrorCode) =>
void)
 */
public clearAllCacheFiles(): void
```

7 技术支持

Foxit 支持

为了提供更具个性化的技术支持，您可以登陆您的 [Foxit 帐户](#)来提交问题报告，以便我们收集问题的详细信息。我们的技术支持团队在收到问题报告后，会第一时间来帮助您解决问题。

您也可以查看我们的[支持中心](#)，选择 Foxit PDF SDK 产品，里面有很多有用的文章可能有助于您解决问题。

联系电话

电话: 1-866-MYFoxit 或者 1-866-693-6948