ComPDFKit PDF SDK

Developer Guides

Comprehensive PDF SDK for Developers

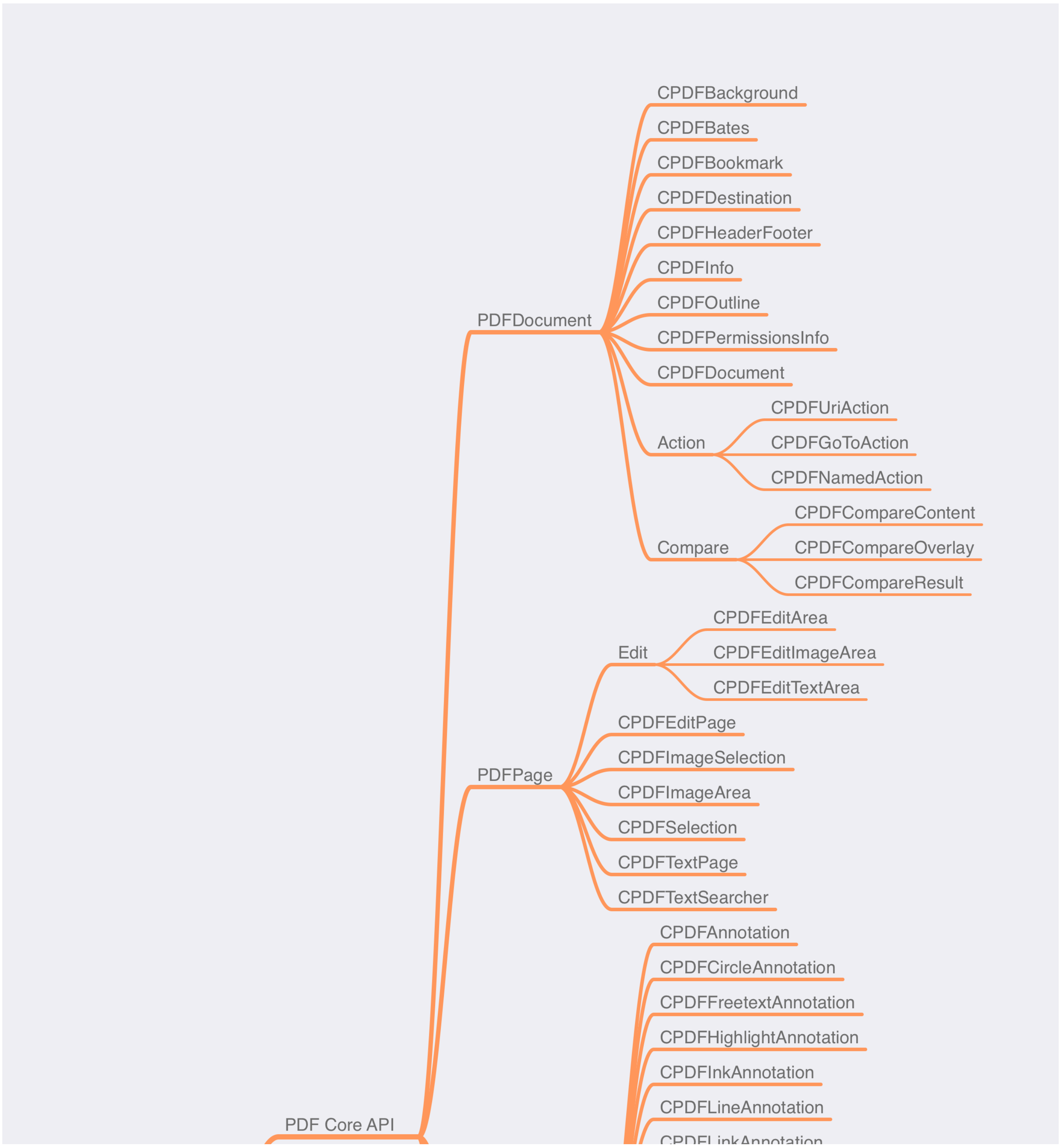
Contents

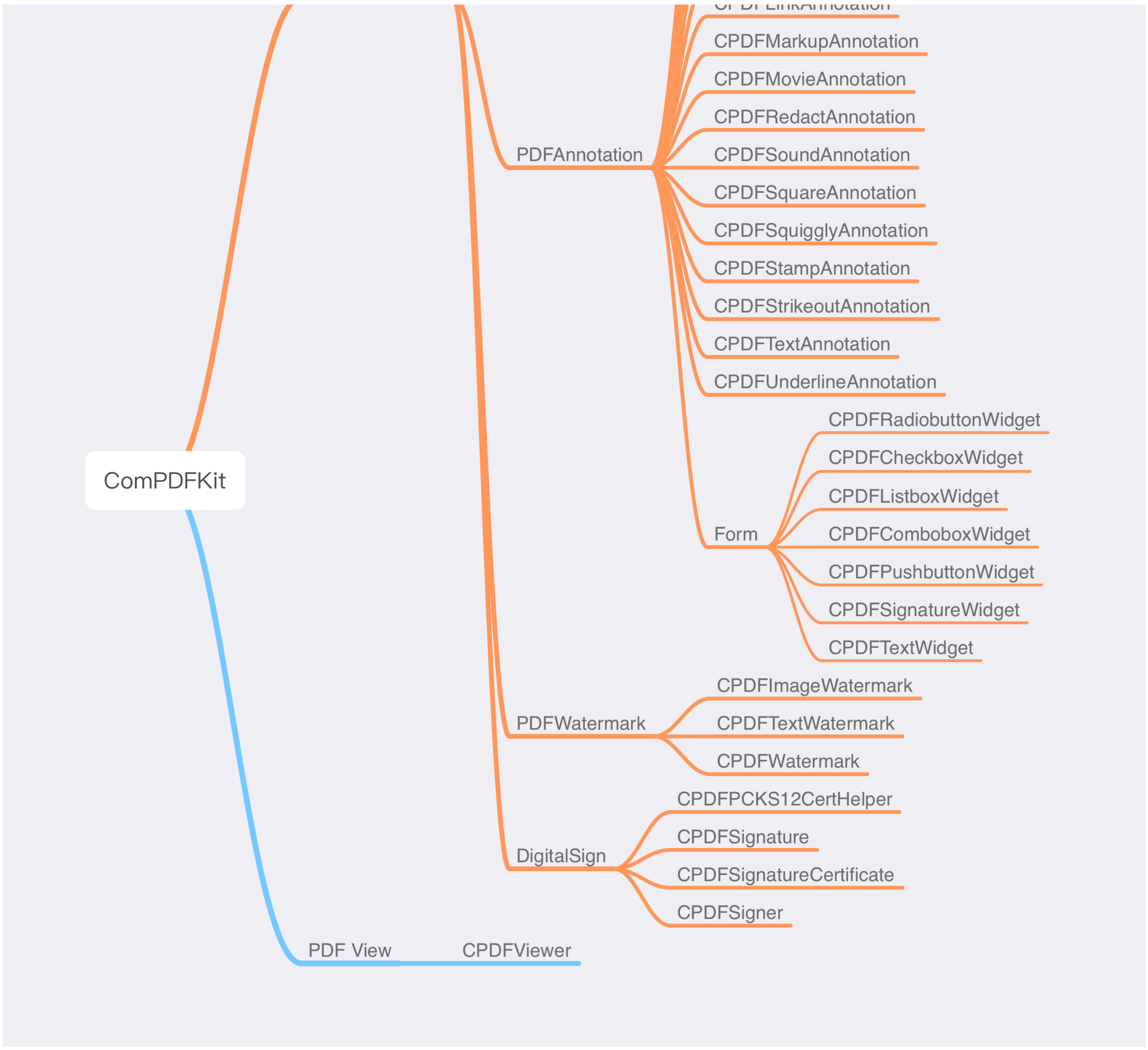
1 Overview

ComPDFKit PDF SDK for Windows is a powerful PDF library that ships with an easy-to-use C# interface. Developers can seamlessly integrate PDF rendering, navigation, creation, searching, annotation, PDF text extraction, form data collection, and editing capabilities into their applications and services.

1.1 ComPDFKit PDF SDK

ComPDFKit PDF SDK consists of two elements as shown in the following picture.





The two elements for ComPDFKit:

- **PDF Core API**

  The ComPDFKit.Desk can be used independently for document rendering, analysis, text extraction, text search, form filling, annotation creation and manipulation, and much more.

- **PDF View**

  The ComPDFKit.Viewer is a utility class that provides the functionality for developers to interact with rendering PDF documents per their requirements. The View Control provides fast and high-quality rendering, zooming, scrolling, and page navigation features.

1.2 Key Features

Viewer component offers:

- Standard page display modes, including Single Page, Double Page, Scrolling, and Cover Mode.

- Navigation with thumbnails, outlines, and bookmarks.
- Text search & selection.
- Zoom in and out & Fit-page.
- Switch between different themes, including Dark Mode, Sepia Mode, Reseda Mode, and Custom Color Mode.
- Text reflow.

Annotations component offers:

- Create, edit, and remove annotations, including Note, Link, Free Text, Line, Square, Circle, Highlight, Underline, Squiggly, Strikeout, Stamp, Ink, and Sound.
- Support for annotation appearances.
- Import and export annotations to/from XFDF.
- Support for annotation flattening.

Forms component offers:

- Create, edit, and remove form fields, including Push Button, Check Box, Radio Button, Text Field, Combo Box, List Box, and Signature.
- Fill PDF Forms.
- Support for PDF form flattening.

Document Editor component offers:

- PDF manipulation, including Split pages, Extract pages, and Merge pages.
- Page edit, include: Delete pages, Insert pages, Move pages, Rotate pages, Replace pages, and Exchange pages.
- Document information setting.
- Extract images.

Content Editor component offers:

- Programmatically add and remove text in PDFs and make it possible to edit PDFs like Word. Allow selecting text to copy, resize, change colors, text alignment, and the position of text boxes.
- Support editing PDF images like moving, rotating, scaling, mirroring, cropping, replacing, copying, and extracting.
- Undo or redo any change.

Security component offers:

- Encrypt and decrypt PDFs, including Permission setting and Password protected.
- Create and remove watermark.
- Redact content including images, text, and vector graphics.
- Create, edit, and remove header & footer, including dates, page numbers, document name, author name, and chapter name.
- Create, edit, and remove bates numbers.
- Create, edit, and remove background that can be a solid color or an image.

Redaction component offers:

- Redact content including images, text, and vector graphics to remove sensitive information or private data, which cannot be restored once applied.
- Create redaction by selecting an area or searching for a specific text.

- Edit and save redaction annotations.

Watermark component offers:

- Add, remove, edit, update, and get the watermarks.
- Support text and image watermarks.

Conversion component offers:

- PDF to PDF/A.

Document Comparison component offers:

- Compare different versions of a document, including overlay comparison and content comparison.

Digital Signatures component offers:

- Sign PDF documents with digital signatures.
- Create and verify digital certificates.
- Create and verify digital digital signatures.
- Create self-sign digital ID and edit signature appearance.
- Support PKCS12 certificates.
- Trust certificates.

1.3 License

ComPDFKit PDF SDK is a commercial SDK, which requires a license to grant developer permission to release their apps. Each license is only valid for one device ID in development mode. Other flexible licensing options are also supported, please contact our marketing team to know more. However, any documents, sample code, or source code distribution from the released package of ComPDFKit PDF SDK to any third party is prohibited.

2 Get Started

It is easy to embed ComPDFKit PDF SDK in your Windows application with a few lines of C# code. Take just a few minutes and get started.

The following sections introduce the requirements, structure of the installation package, and how to make a Windows PDF Reader in C# with ComPDFKit PDF SDK.

2.1 Requirements

- Windows 7, 8, 10, and 11 (32-bit and 64-bit).
- Visual Studio 2017 or higher (Make sure the **.NET Desktop Development** is installed).
- .NET Framework 4.5 or higher.

2.2 Windows Package Structure

You can contact us and access our PDF SDK installation package. The SDK package includes the following files.

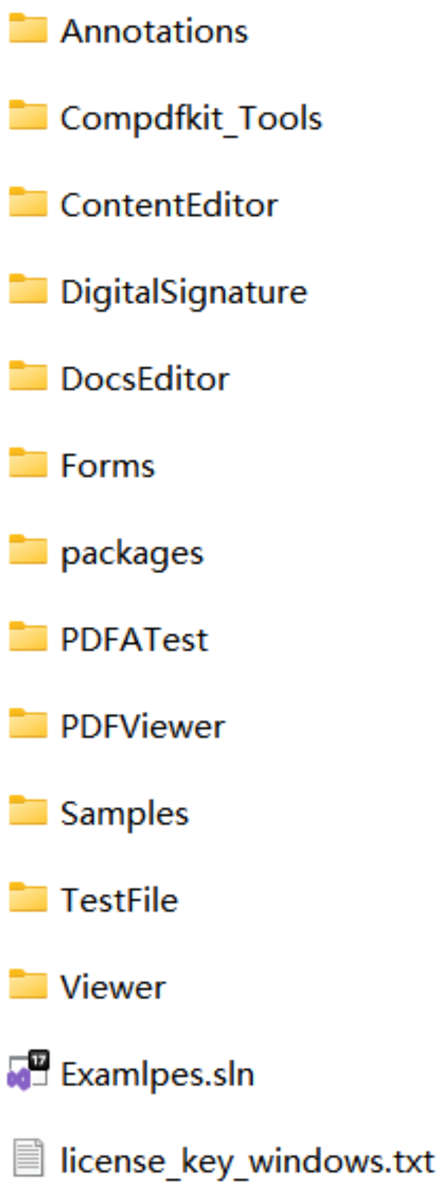
- ***"Examples"*** - A folder containing Windows sample projects.
    - ***"Viewer"*** - A basic PDF viewer, including reading PDFs, changing themes, bookmarks, searching text, etc.
    - ***"Annotations"*** - A PDF viewer with full types of annotation editing, including adding annotations, modifying annotations, annotation lists, etc.
    - ***"ContentEditor"*** - A PDF viewer with text and image editing, including modifying text, replacing images, etc.
    - ***"Forms"*** - A PDF viewer with full types of forms editing, including radio button, combo box, etc.
    - ***"DocsEditor"*** - A PDF viewer with page editing, including inserting/deleting pages, extracting pages, reordering pages, etc.
    - ***"PDFViewer"*** - A multi-functional PDF program that integrates all of the above features.
    - ***"ComPDFKit_Tools"*** - A default control library for quickly building various function modules of PDF viewer.
    - ***"license_key_win.xml"*** - A xml file containing key and secret.
    - ***"TestFile"*** - A folder containing test files.
    - ***"Samples"*** - -A folder containing console application.
- ***"lib"*** - Include the ComPDFKit dynamic library (x86,x64).
- ***"nuget"*** - Include the ComPDFKit.NetFramework nuget package file.
- ***"api_reference_windows.chm"*** - API reference.
- ***"developer_guide_windows.pdf"*** - Developer guide.
- ***"legal.txt"*** - Legal and copyright information.
- ***"release_notes.txt"*** - Release information.

2.3 How to Run a Demo

ComPDFKit PDF SDK for Windows provides multiple demos in C# for developers to learn how to call the SDK on Windows. You can find them in the ***"Examples"*** folder.

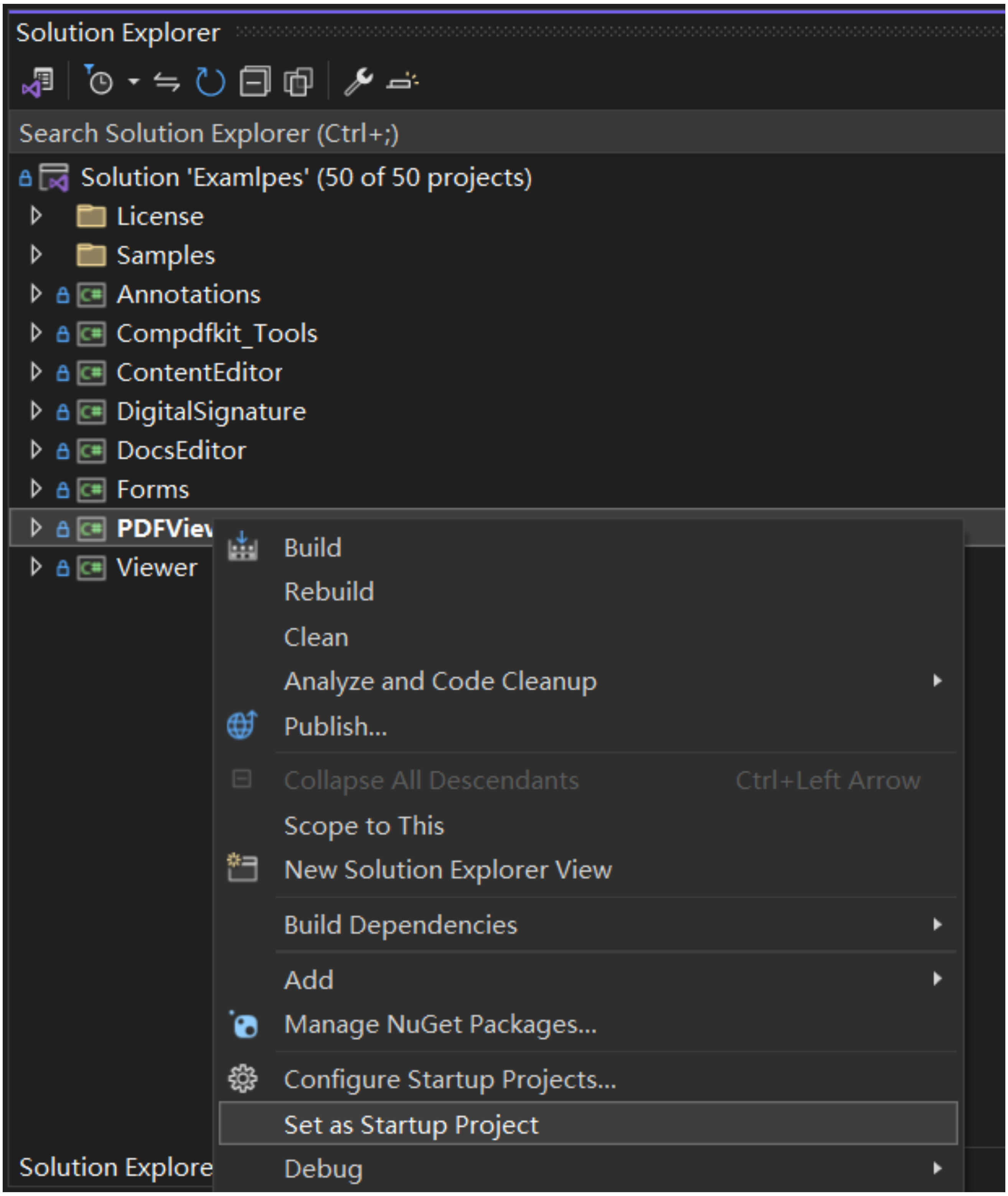
In this guide, we take ***"PDFViewer"*** as an example to show how to run it in Visual Studio 2022.

1. Copy your ***"license_key_windows.txt"*** to the ***"Examples"*** folder (The file is the license to make your project run).
2. Find ***"Examples.sln"*** in the ***"Examples"*** folder and open it in Visual Studio 2022.

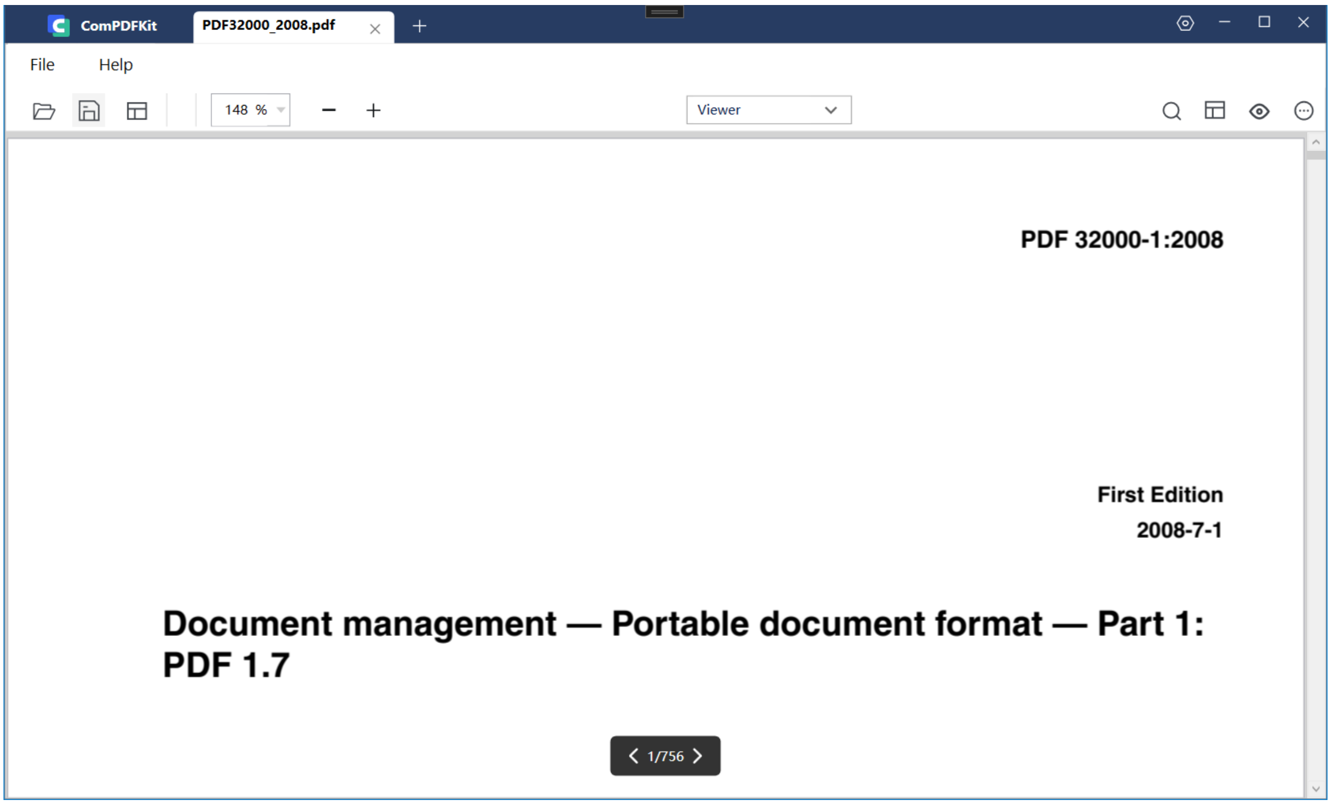


Annotations
Compdfkit_Tools
ContentEditor
DigitalSignature
DocsEditor
Forms
packages
PDFATest
PDFViewer
Samples
TestFile
Viewer
Examlpes.sln
license_key_windows.txt

3. Select ***"PDFViewer"*** and right-click to set it as a startup project.



4. Run the project and then you can open the multifunctional ***"PDFViewer"*** demo.



PDF 32000-1:2008

First Edition
2008-7-1

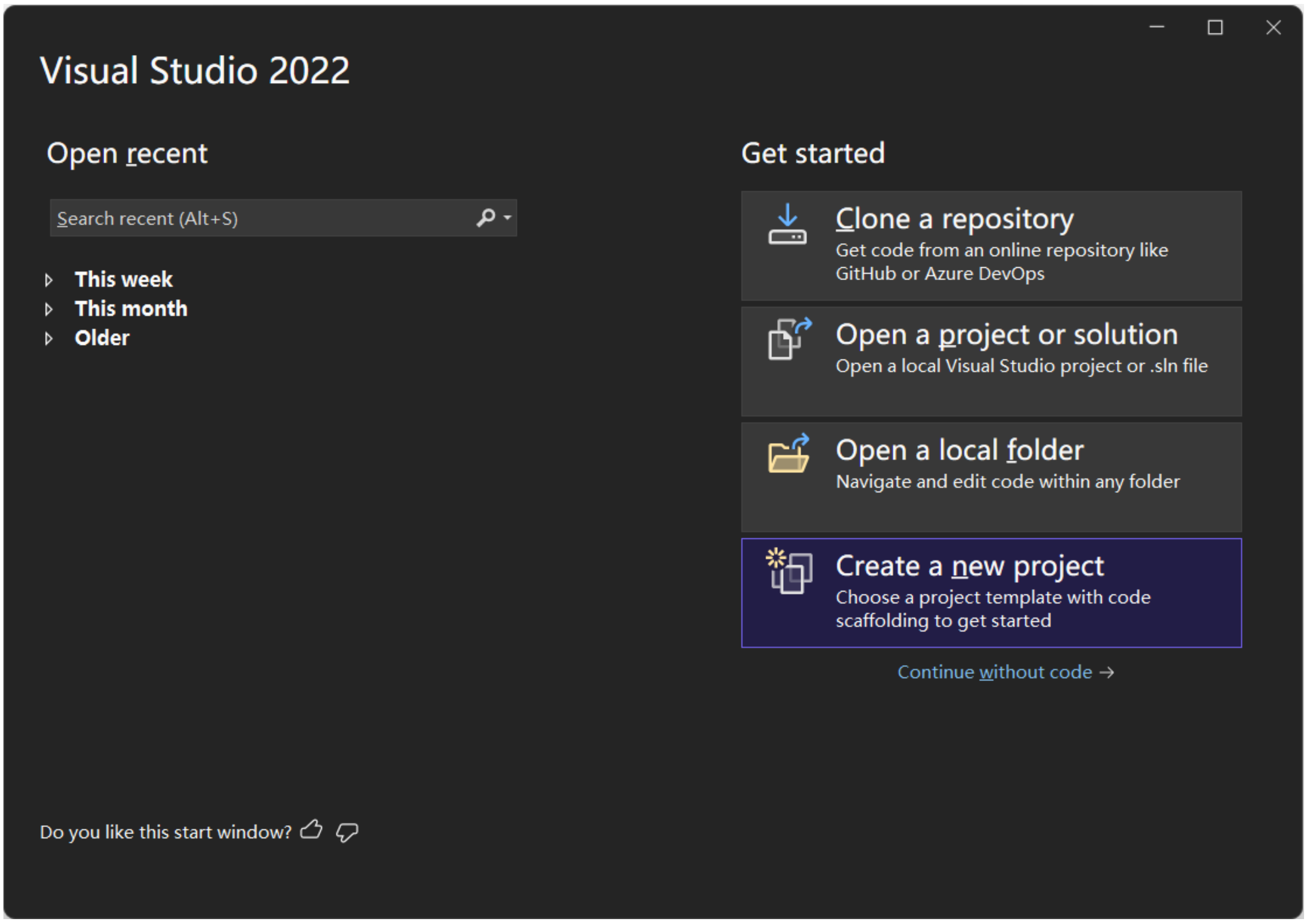
Document management — Portable document format — Part 1: PDF 1.7

Note: *This is a demo project, presenting completed ComPDFKit PDF SDK functions. The functions might be different based on the license you have purchased. Please check that the functions you choose work fine in this demo project.*

2.4 How to Make a Windows Program in C# with ComPDFKit PDF SDK

2.4.1 Create a New Project

1. Fire up Visual Studio 2022, click **Create a new project**.


Visual Studio 2022

Open recent

Search recent (Alt+S)

- This week
- This month
- Older

Get started

Clone a repository Get code from an online repository like GitHub or Azure DevOps

Open a project or solution Open a local Visual Studio project or .sln file

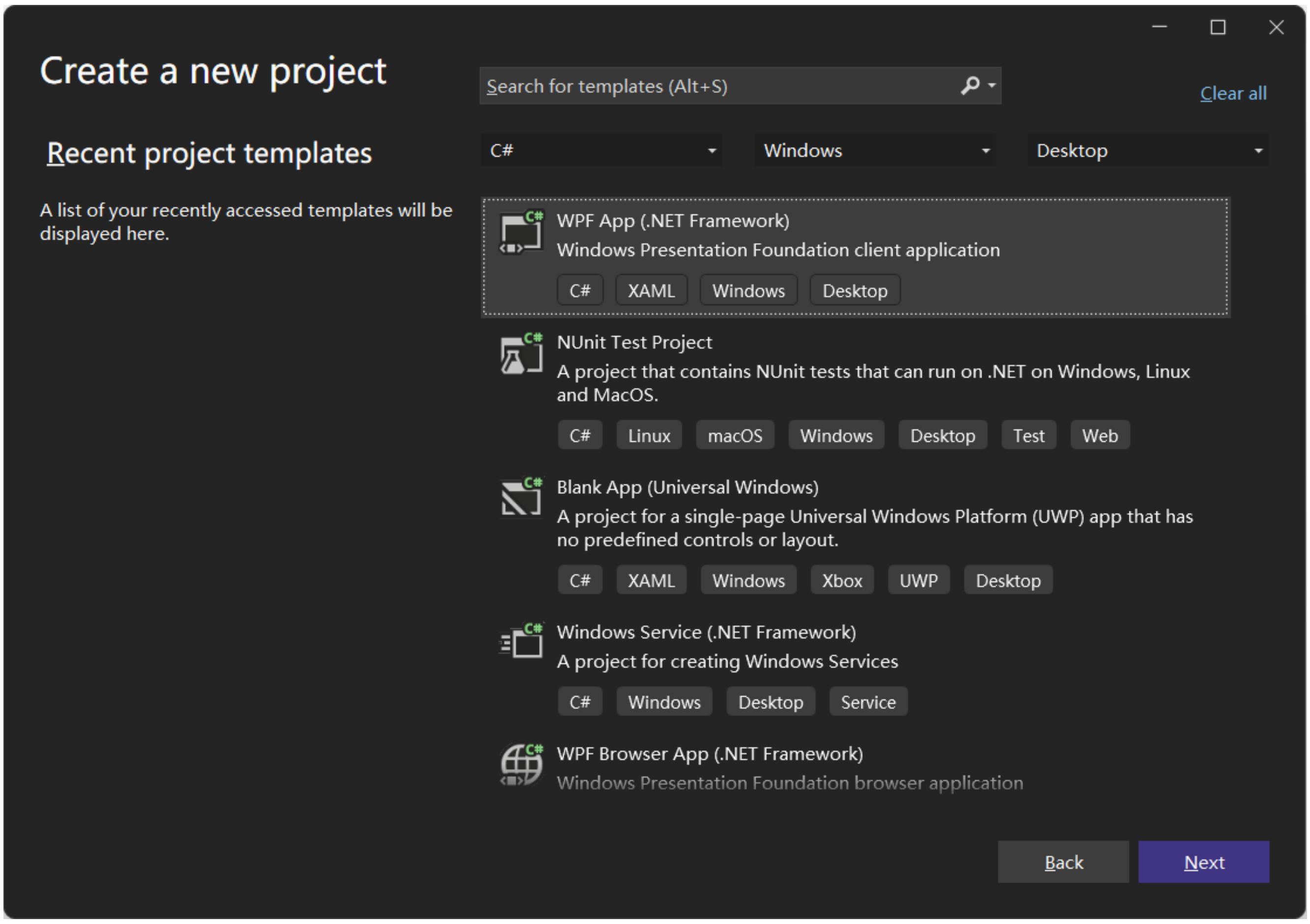
Open a local folder Navigate and edit code within any folder

Create a new project Choose a project template with code scaffolding to get started

Continue without code →

Do you like this start window?

2. Choose **WPF App (.NET Framework)** and click **Next**.


Create a new project

Search for templates (Alt+S)

Clear all

Recent project templates

A list of your recently accessed templates will be displayed here.

C# | Windows | Desktop

WPF App (.NET Framework)
Windows Presentation Foundation client application
C# XAML Windows Desktop

NUnit Test Project
A project that contains NUnit tests that can run on .NET on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop Test Web

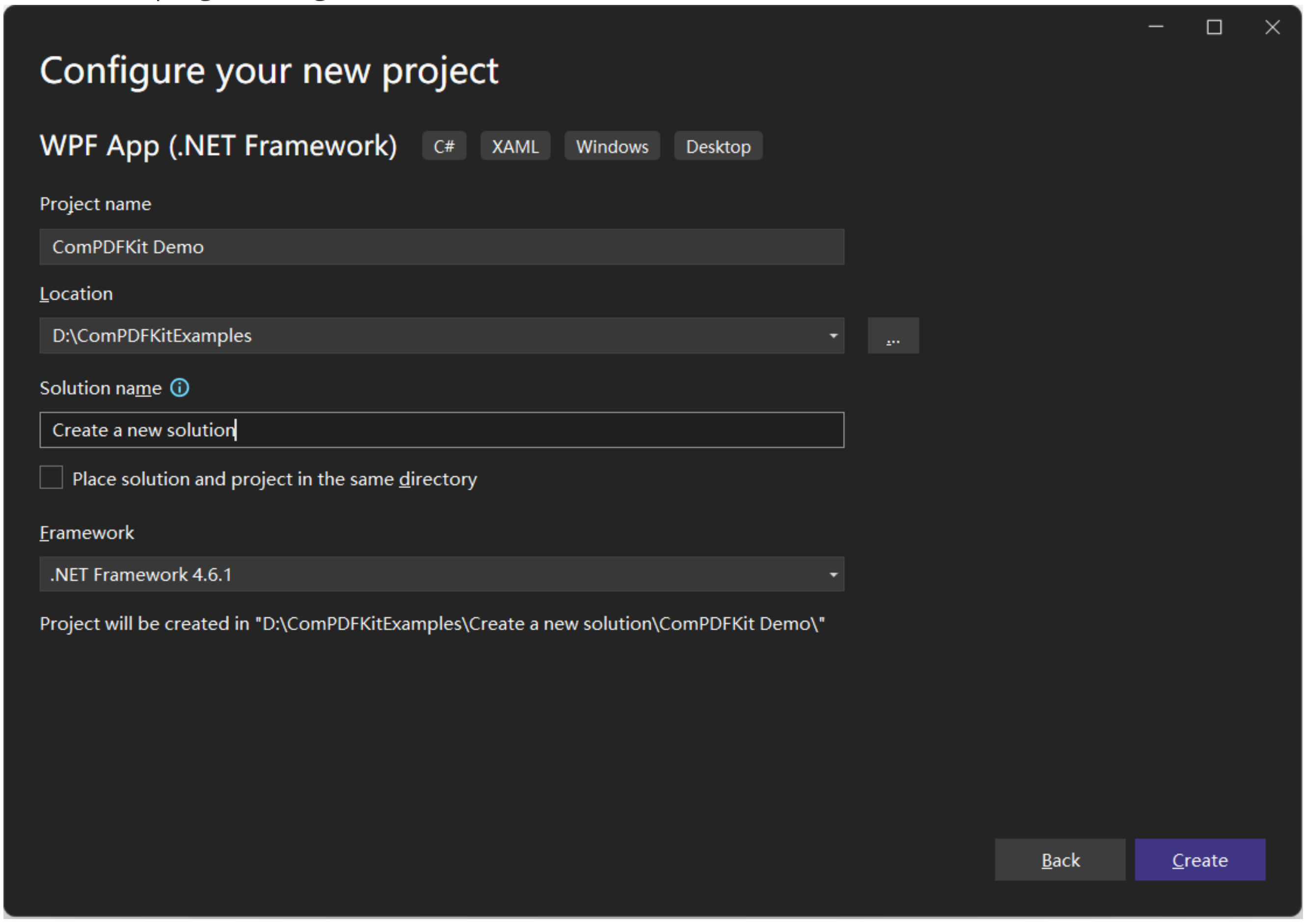
Blank App (Universal Windows)
A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout.
C# XAML Windows Xbox UWP Desktop

Windows Service (.NET Framework)
A project for creating Windows Services
C# Windows Desktop Service

WPF Browser App (.NET Framework)
Windows Presentation Foundation browser application

Back Next

3. Configure your project: Set your project name and choose the location to store your program. The project name is called "ComPDFKit Demo" in this example. This sample project uses .NET Framework 4.6.1 as the programming framework.


Configure your new project

WPF App (.NET Framework) C# XAML Windows Desktop

Project name

ComPDFKit Demo

Location

D:\ComPDFKitExamples

Solution name

Create a new solution

Place solution and project in the same directory

Framework

.NET Framework 4.6.1

Project will be created in "D:\ComPDFKitExamples\Create a new solution\ComPDFKit Demo\"

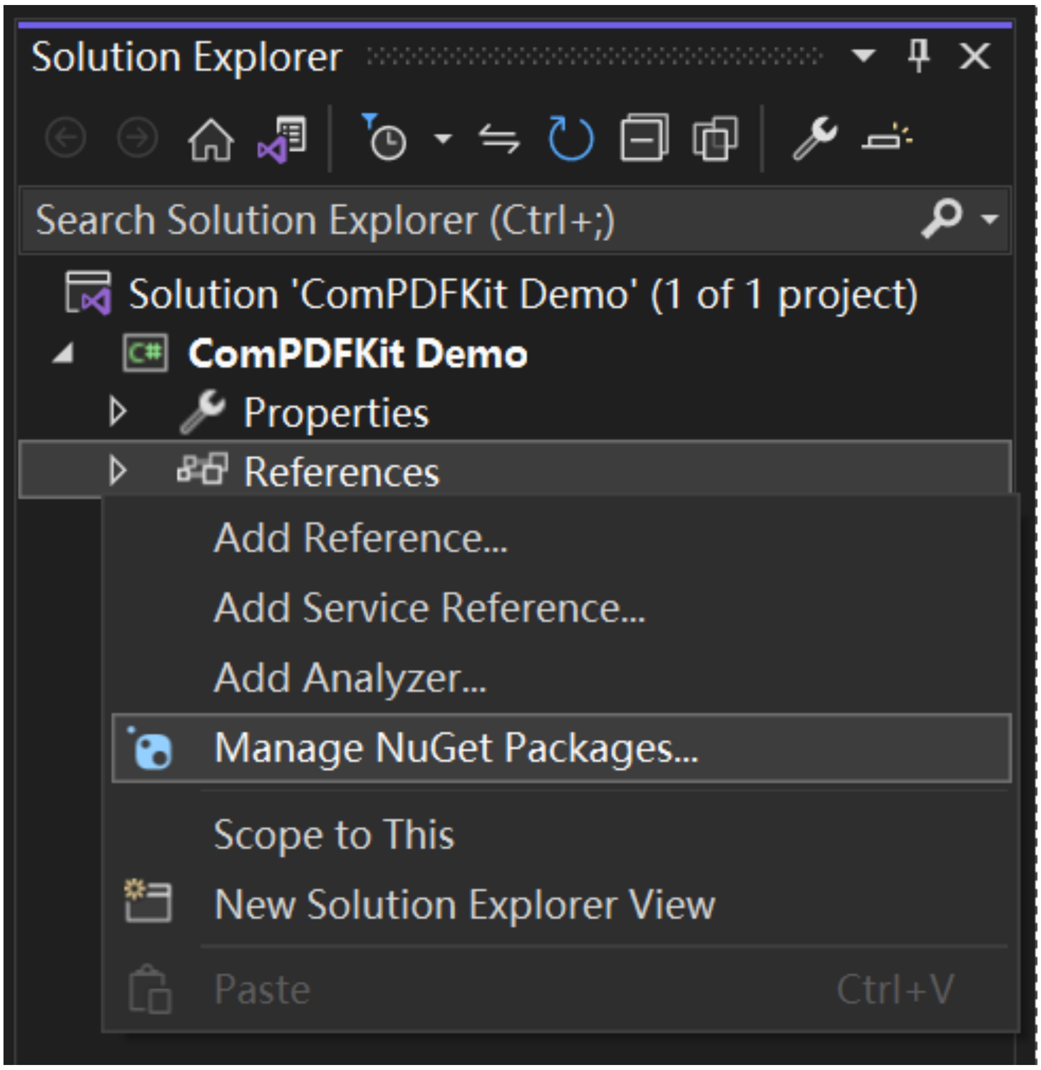
4. Click the **Create** button. Then, the new project will be created.

2.4.2 Add ComPDFKit to Your Project

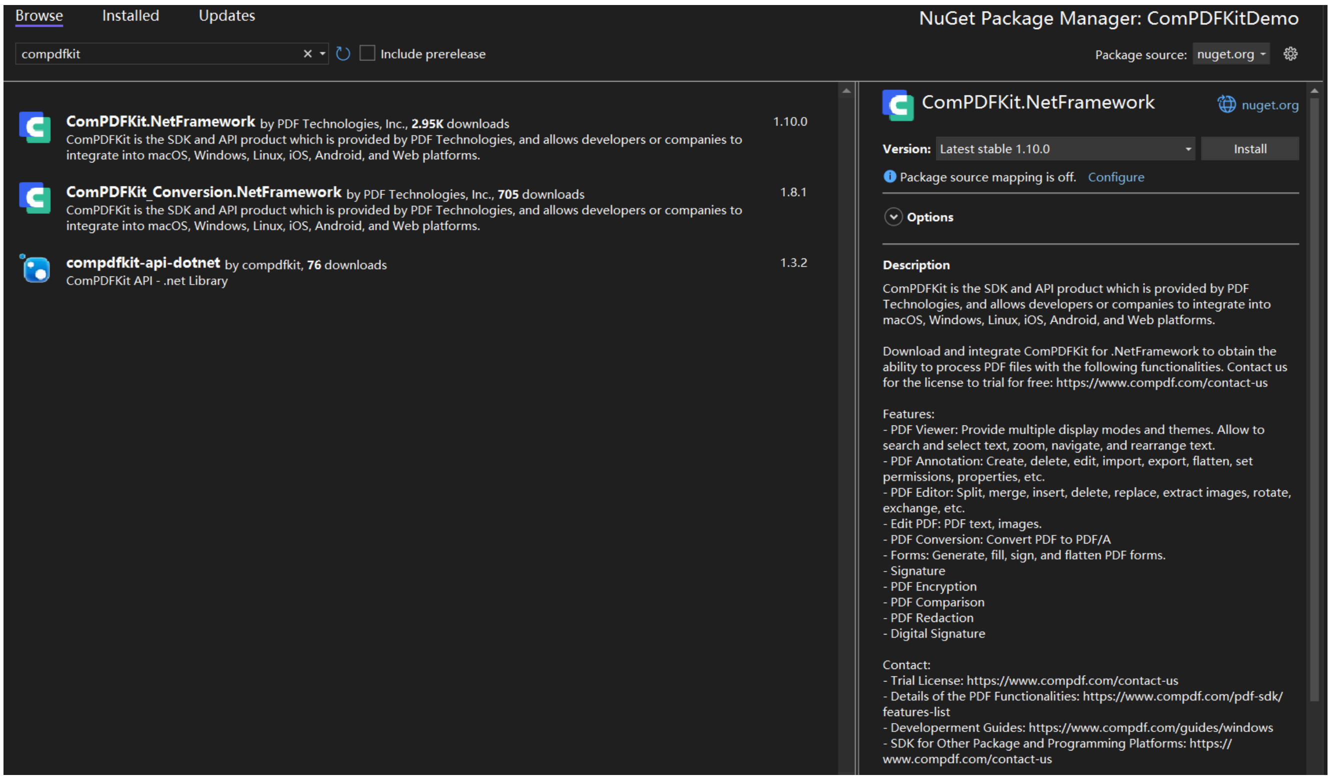
There are two ways to add ComPDFKit to your Project: `Nuget Repository` and `Local Package`, you can choose one or the other according to your needs.

Nuget Repository

1. Open your project's solution, and in the Solution Explorer, right-click on **References** and click on the menu item **Manage NuGet Packages...**. This will open the NuGet Package Manager for your solution.



2. Search for `ComPDFKit.NetFramework`, and you'll find the package on nuget.org.
3. On the right side, in the panel describing the package, click on the **Install** button to install the package.


Browse
Installed
Updates
NuGet Package Manager: ComPDFKitDemo
compdfkit
Include prerelease
Package source: nuget.org

ComPDFKit.NetFramework by PDF Technologies, Inc., **2.95K** downloads 1.10.0
ComPDFKit is the SDK and API product which is provided by PDF Technologies, and allows developers or companies to integrate into macOS, Windows, Linux, iOS, Android, and Web platforms.

ComPDFKit_Conversion.NetFramework by PDF Technologies, Inc., **705** downloads 1.8.1
ComPDFKit is the SDK and API product which is provided by PDF Technologies, and allows developers or companies to integrate into macOS, Windows, Linux, iOS, Android, and Web platforms.

compdfkit-api-dotnet by compdfkit, **76** downloads 1.3.2
ComPDFKit API - .net Library

ComPDFKit.NetFramework nuget.org

Version: Latest stable 1.10.0 Install

Package source mapping is off. Configure

Options

Description

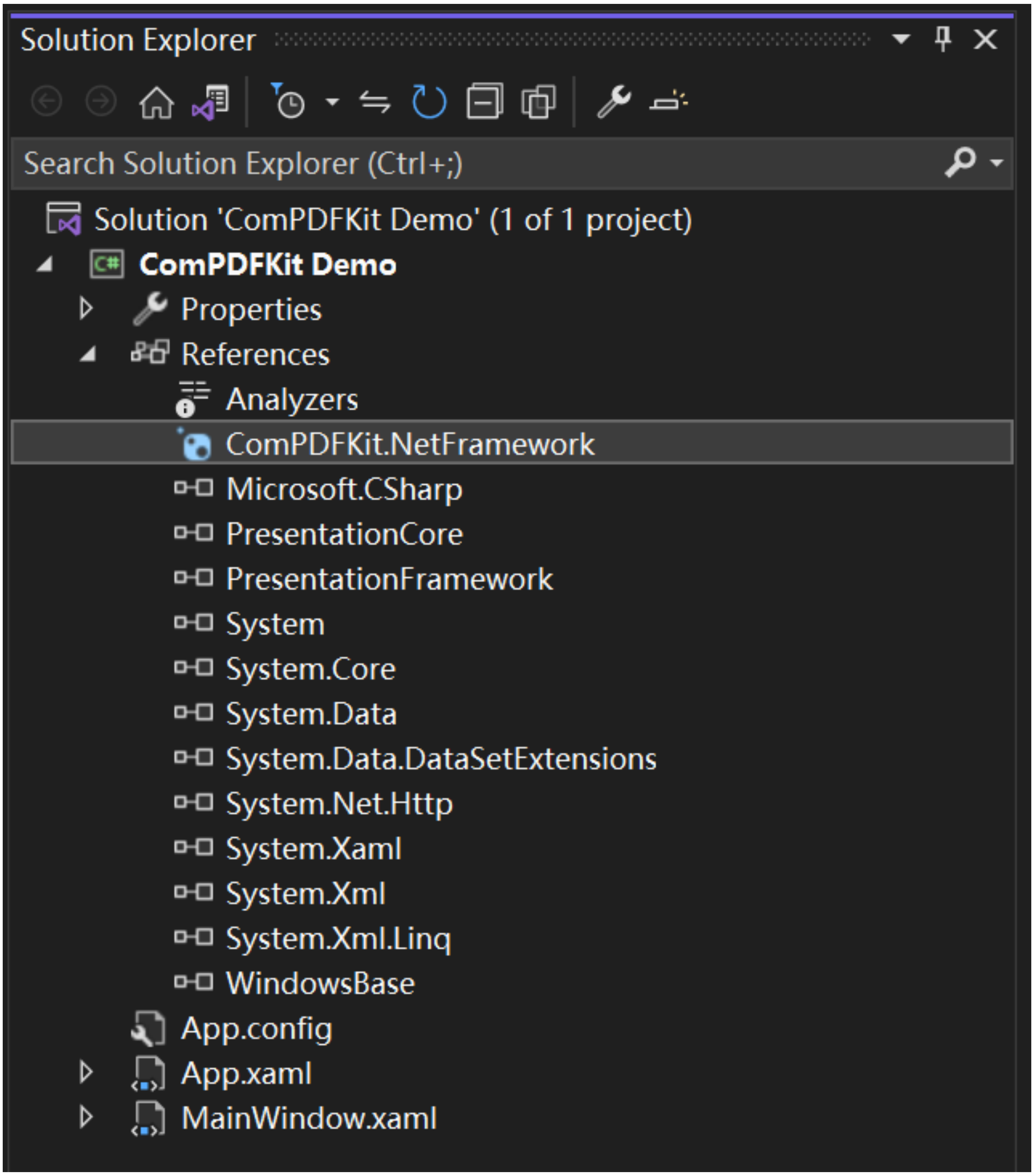
ComPDFKit is the SDK and API product which is provided by PDF Technologies, and allows developers or companies to integrate into macOS, Windows, Linux, iOS, Android, and Web platforms.

Download and integrate ComPDFKit for .NetFramework to obtain the ability to process PDF files with the following functionalities. Contact us for the license to trial for free: https://www.compdf.com/contact-us

Features:
- PDF Viewer: Provide multiple display modes and themes. Allow to search and select text, zoom, navigate, and rearrange text.
- PDF Annotation: Create, delete, edit, import, export, flatten, set permissions, properties, etc.
- PDF Editor: Split, merge, insert, delete, replace, extract images, rotate, exchange, etc.
- Edit PDF: PDF text, images.
- PDF Conversion: Convert PDF to PDF/A
- Forms: Generate, fill, sign, and flatten PDF forms.
- Signature
- PDF Encryption
- PDF Comparison
- PDF Redaction
- Digital Signature

Contact:
- Trial License: https://www.compdf.com/contact-us
- Details of the PDF Functionalities: https://www.compdf.com/pdf-sdk/features-list
- Developerment Guides: https://www.compdf.com/guides/windows
- SDK for Other Package and Programming Platforms: https://www.compdf.com/contact-us

4. Once that is complete, you'll see a reference to the package in the Solution Explorer under **References**.


Solution Explorer

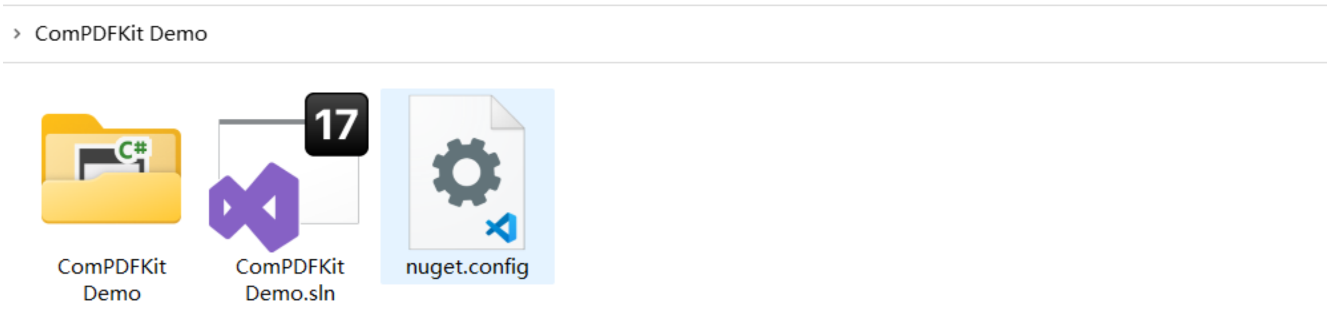
Search Solution Explorer (Ctrl+;)

Solution 'ComPDFKit Demo' (1 of 1 project)
- ComPDFKit Demo
  - Properties
  - References
    - Analyzers
    - ComPDFKit.NetFramework
    - Microsoft.CSharp
    - PresentationCore
    - PresentationFramework
    - System
    - System.Core
    - System.Data
    - System.Data.DataSetExtensions
    - System.Net.Http
    - System.Xaml
    - System.Xml
    - System.Xml.Linq
    - WindowsBase
  - App.config
  - App.xaml
  - MainWindow.xaml

Local Package

Rather than targeting a package held at nuget.org, you may set up a configuration to point to a local package. This can be useful for some situations, for example, your build machines don't have access to the internet.

1. You can find ***"ComPDFKit.NetFramework....nupkg"*** file in the SDK Package
2. Create or edit a ***"nuget.config"*** file in the same directory as your solution file (e.g. ***"ComPDFKit Demo.sln"***).

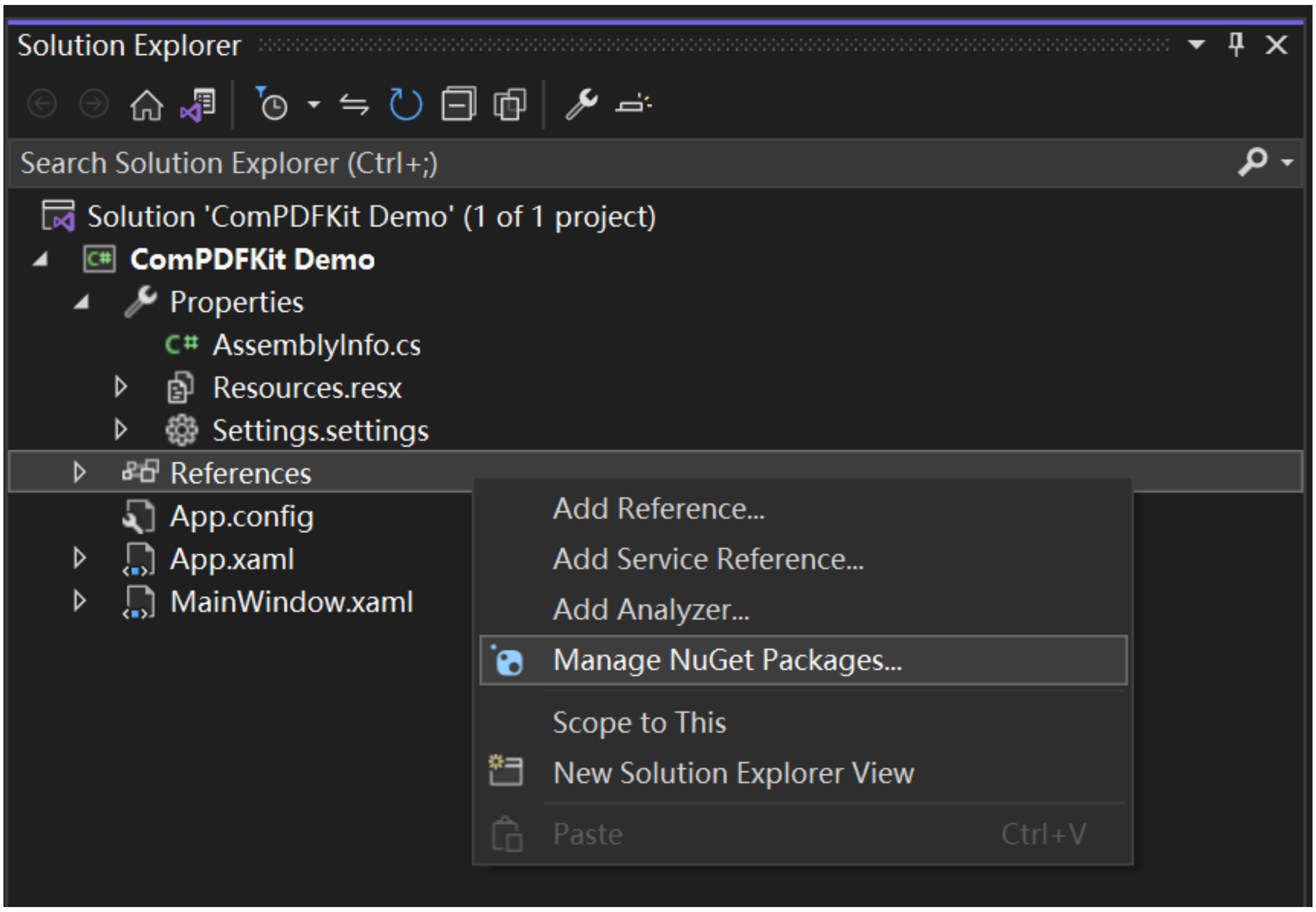


The contents of the file should contain an XML element, `packageSources` — which describes where to find NuGet packages — as a child of a root node named `configuration`. If the file already exists, add the extra `packageSources` entry shown below. If the file is blank, copy and paste the entirety of the following contents:

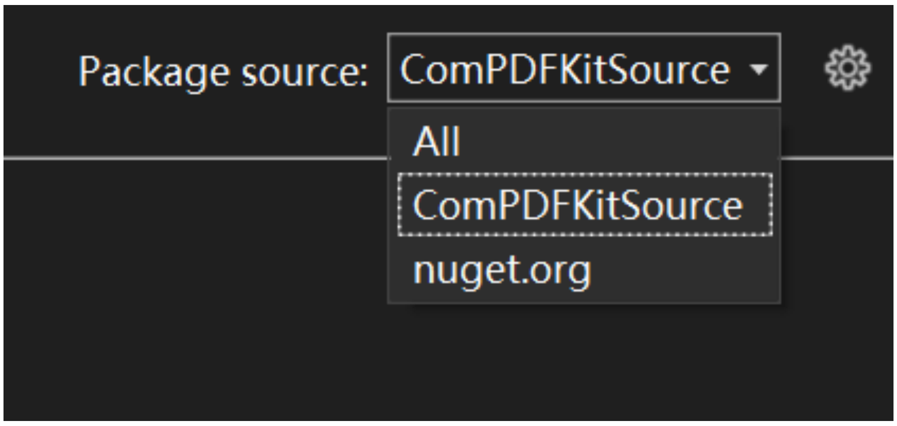
```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="ComPDFKitSource" value="path\to\directoryContainingNupkg" />
  </packageSources>
</configuration>
```

Edit the `value` of the contents to correctly refer to the location of the directory containing the ***"ComPDFKit.NetFramework....nupkg"*** package — for example, `C:\Users\me\nugetPackages\`. Now save the file, and close and reopen your solution for Visual Studio to force a read of the NuGet configuration.

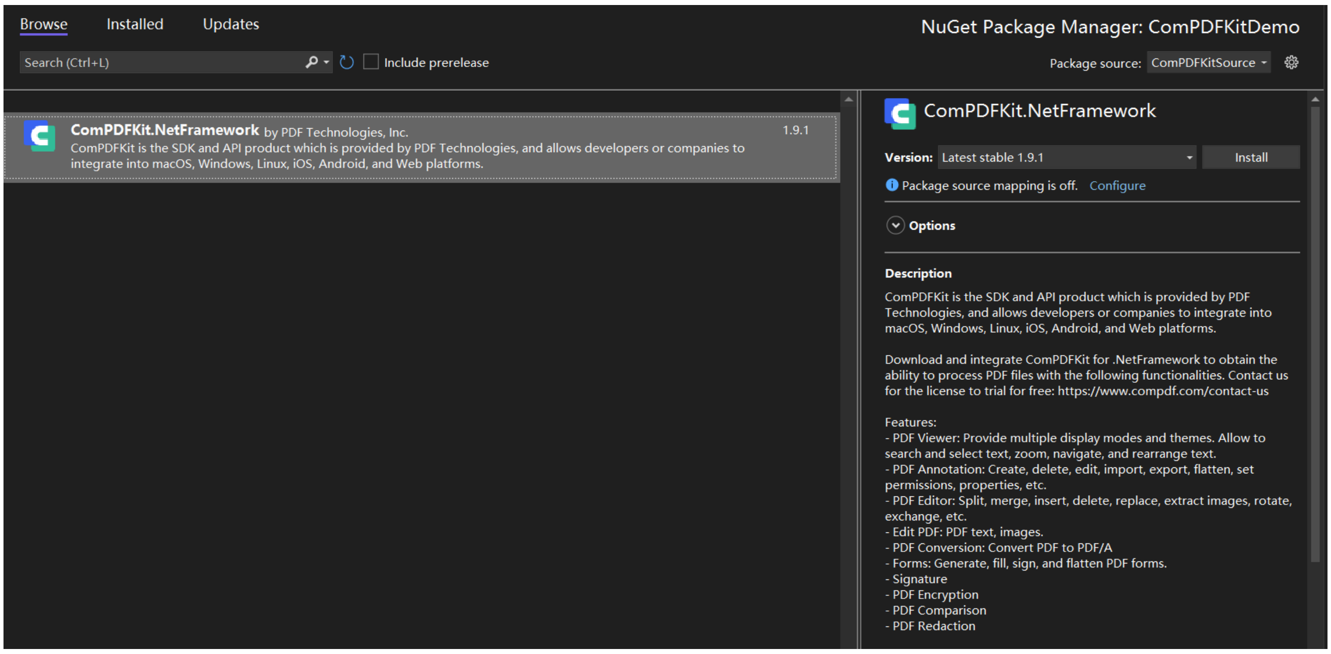
3. Open your project's solution, and in the Solution Explorer, right-click on **References** and click on the menu item **Manage NuGet Packages...**. This will open the NuGet Package Manager for your solution.



4. On the right-hand side of the manager in the Package source dropdown window, choose the entry `ComPDFKitSource` (or whatever you decided to name it). You should then see the entry for ***"ComPDFKit.NetFramework"***.



5. On the right side, in the panel describing the package, click on the **Install** button to install the package.


Browse Installed Updates

NuGet Package Manager: ComPDFKitDemo

Search (Ctrl+L) Include prerelease

Package source: ComPDFKitSource

ComPDFKit.NetFramework by PDF Technologies, Inc. 1.9.1
ComPDFKit is the SDK and API product which is provided by PDF Technologies, and allows developers or companies to integrate into macOS, Windows, Linux, iOS, Android, and Web platforms.

ComPDFKit.NetFramework

Version: Latest stable 1.9.1 Install

Package source mapping is off. Configure

Options

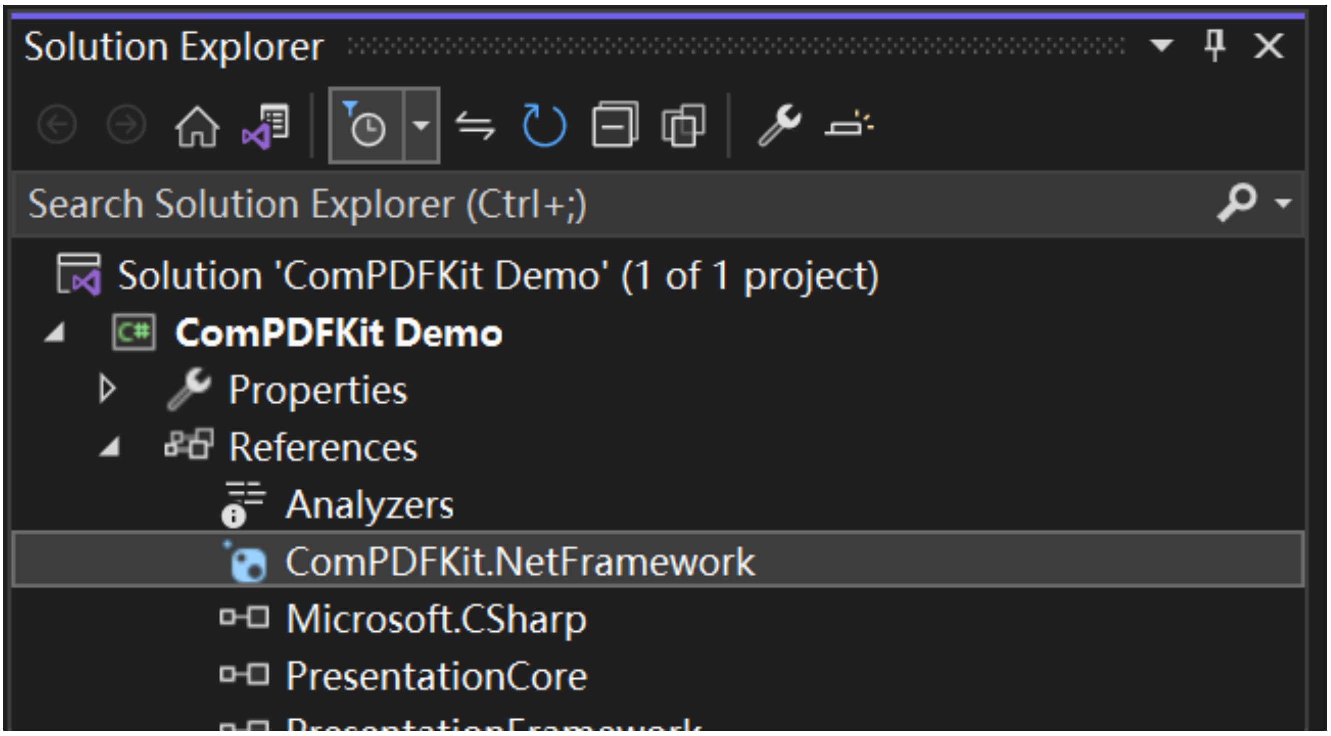
Description

ComPDFKit is the SDK and API product which is provided by PDF Technologies, and allows developers or companies to integrate into macOS, Windows, Linux, iOS, Android, and Web platforms.

Download and integrate ComPDFKit for .NetFramework to obtain the ability to process PDF files with the following functionalities. Contact us for the license to trial for free: https://www.compdf.com/contact-us

Features:
- PDF Viewer: Provide multiple display modes and themes. Allow to search and select text, zoom, navigate, and rearrange text.
- PDF Annotation: Create, delete, edit, import, export, flatten, set permissions, properties, etc.
- PDF Editor: Split, merge, insert, delete, replace, extract images, rotate, exchange, etc.
- Edit PDF: PDF text, images.
- PDF Conversion: Convert PDF to PDF/A
- Forms: Generate, fill, sign, and flatten PDF forms.
- Signature
- PDF Encryption
- PDF Comparison
- PDF Redaction

6. Once that's complete, you'll see a reference to the package in the Solution Explorer under **References**.

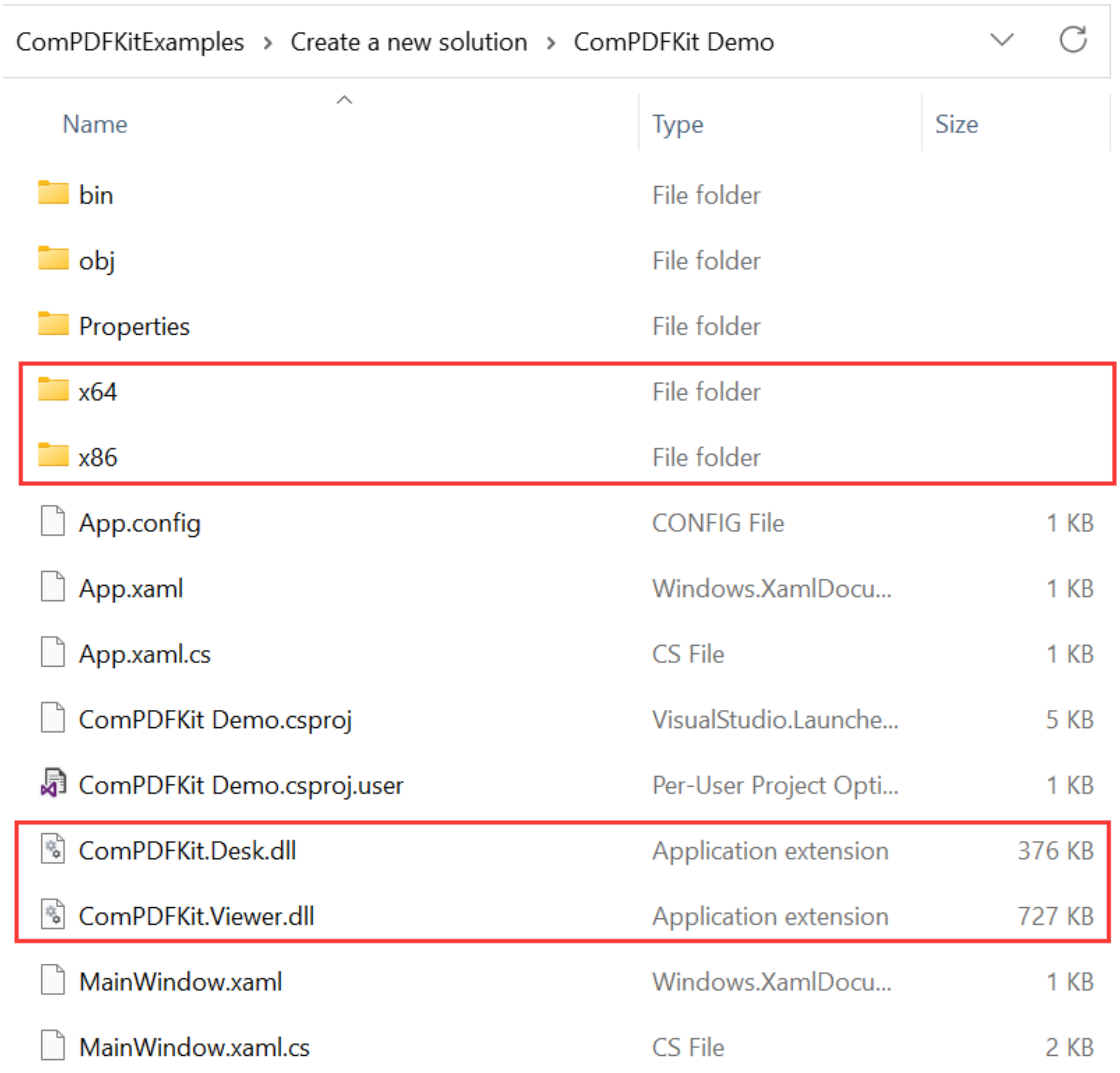

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'ComPDFKit Demo' (1 of 1 project)
- **ComPDFKit Demo**
  - Properties
  - References
    - Analyzers
    - ComPDFKit.NetFramework
    - Microsoft.CSharp
    - PresentationCore

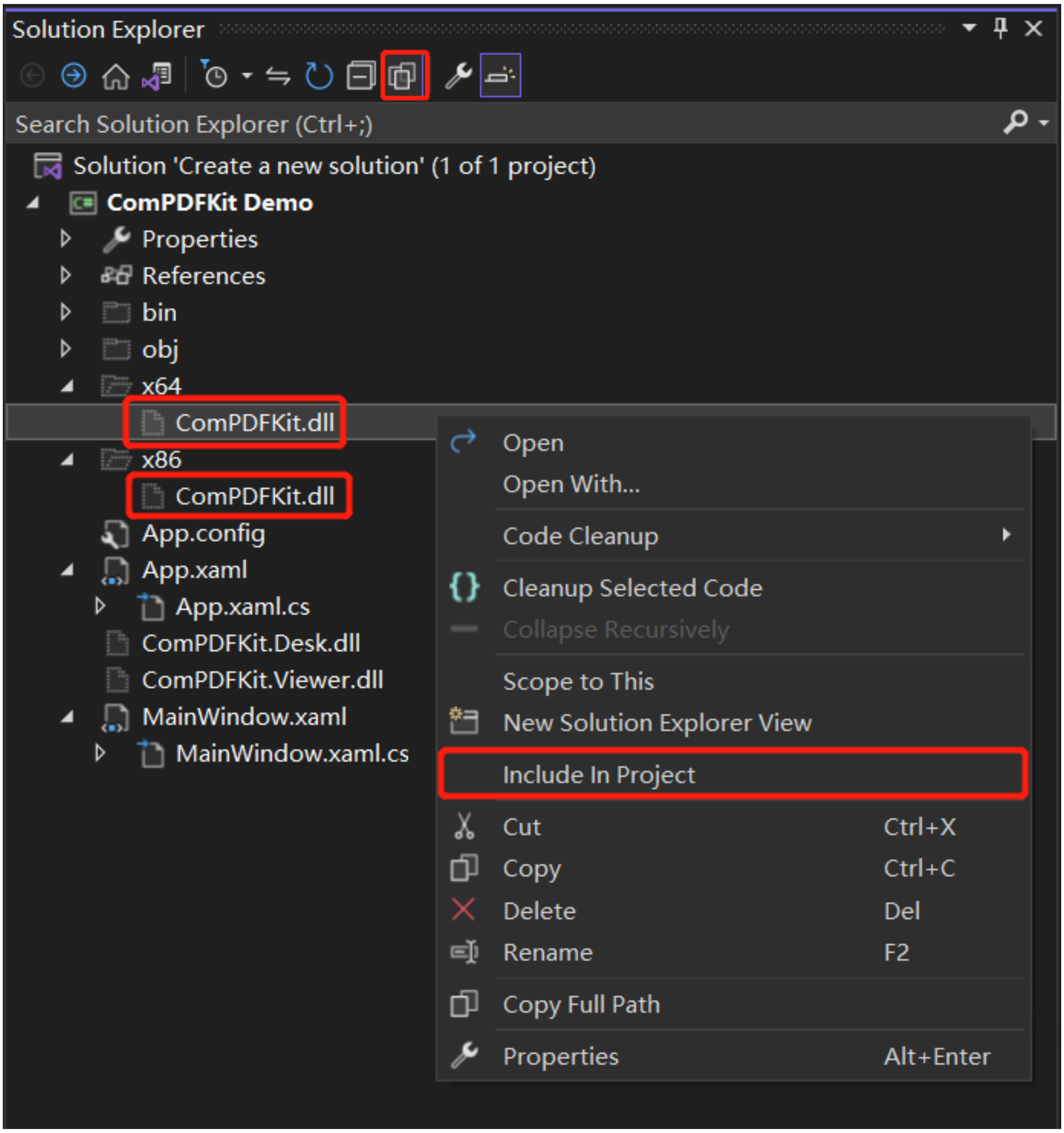
Alternatively, you can manually integrate ComPDFKit's dynamic libraries into your project.

1. Open the ComPDFKit SDK package that you have extracted, and go to the ***"lib"*** folder. Copy the ***"x64"*** folder, ***"x86"*** folder, ***"ComPDFKit.Desk.dll"*** file, and ***"ComPDFKit.Viewer.dll"*** file to the folder with the same name as your project that you created in 2.4.1. In this project, the folder is named "ComPDFKit Demo". Now, your folder should look like this:

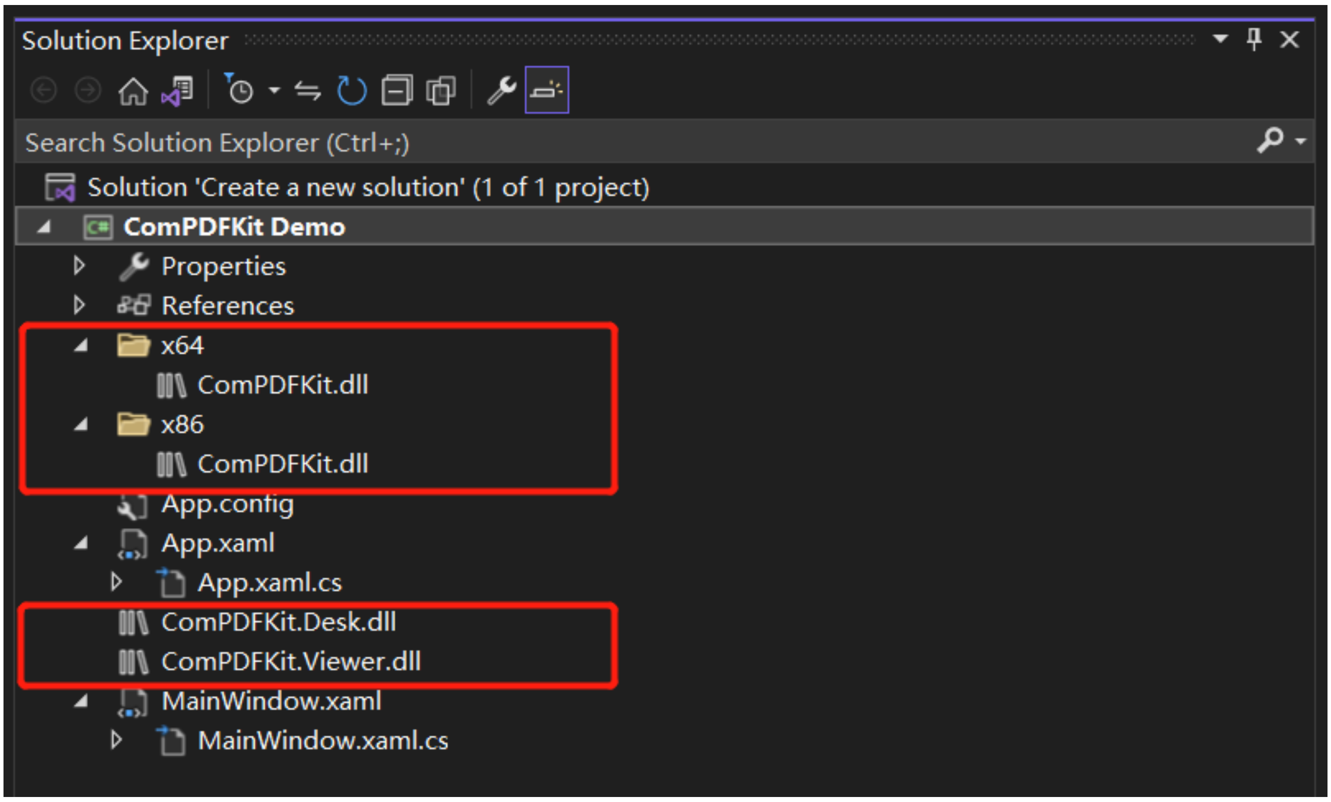

ComPDFKitExamples › Create a new solution › ComPDFKit Demo

Name	Type	Size
bin	File folder	
obj	File folder	
Properties	File folder	
x64	File folder	
x86	File folder	
App.config	CONFIG File	1 KB
App.xaml	Windows.XamlDocu...	1 KB
App.xaml.cs	CS File	1 KB
ComPDFKit Demo.csproj	VisualStudio.Launche...	5 KB
ComPDFKit Demo.csproj.user	Per-User Project Opti...	1 KB
ComPDFKit.Desk.dll	Application extension	376 KB
ComPDFKit.Viewer.dll	Application extension	727 KB
MainWindow.xaml	Windows.XamlDocu...	1 KB
MainWindow.xaml.cs	CS File	2 KB

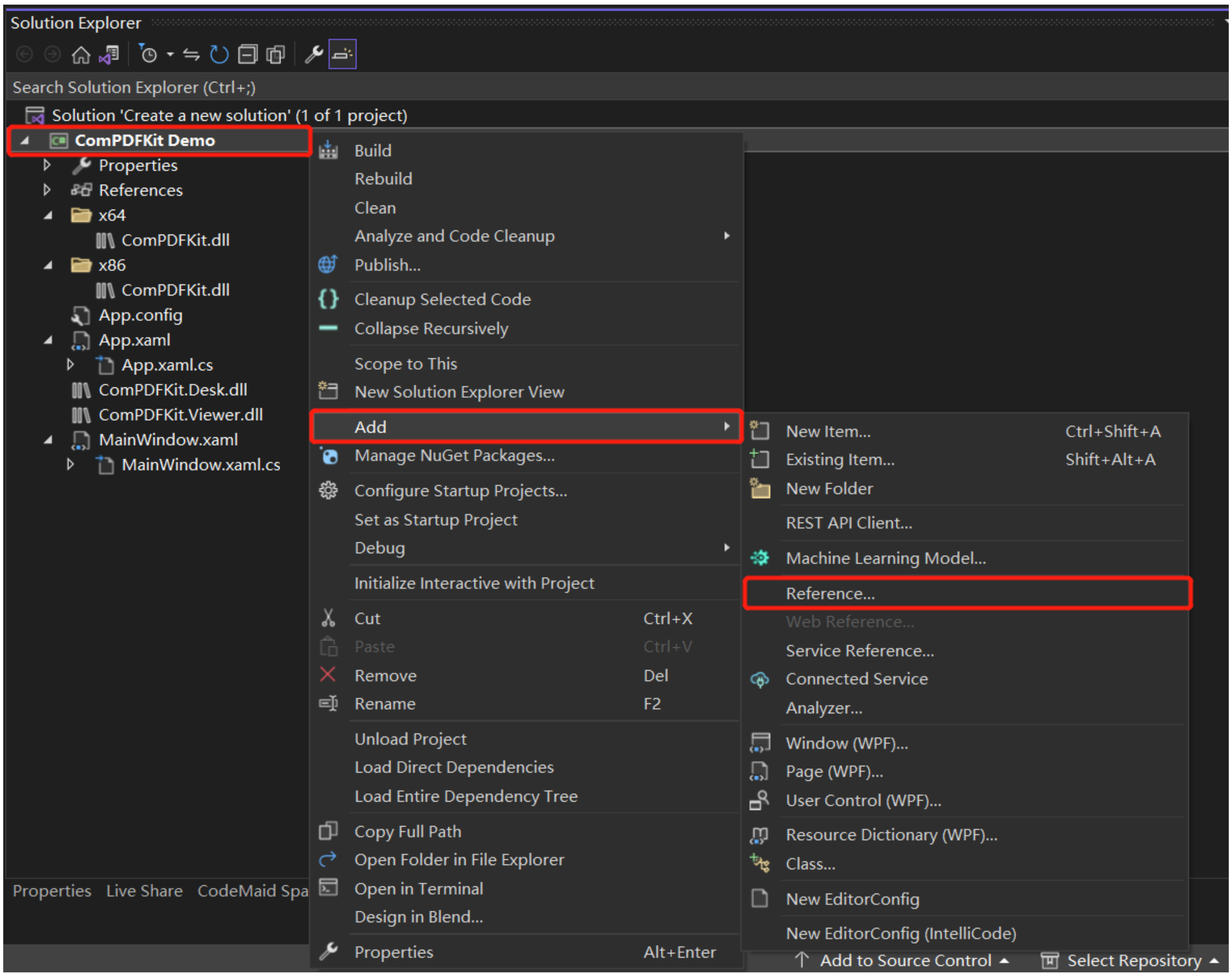
2. Then click the button **Show All Files** in the **Solution Explorer** menu. Find and right click the files you added before, and choose **Include In Project**.



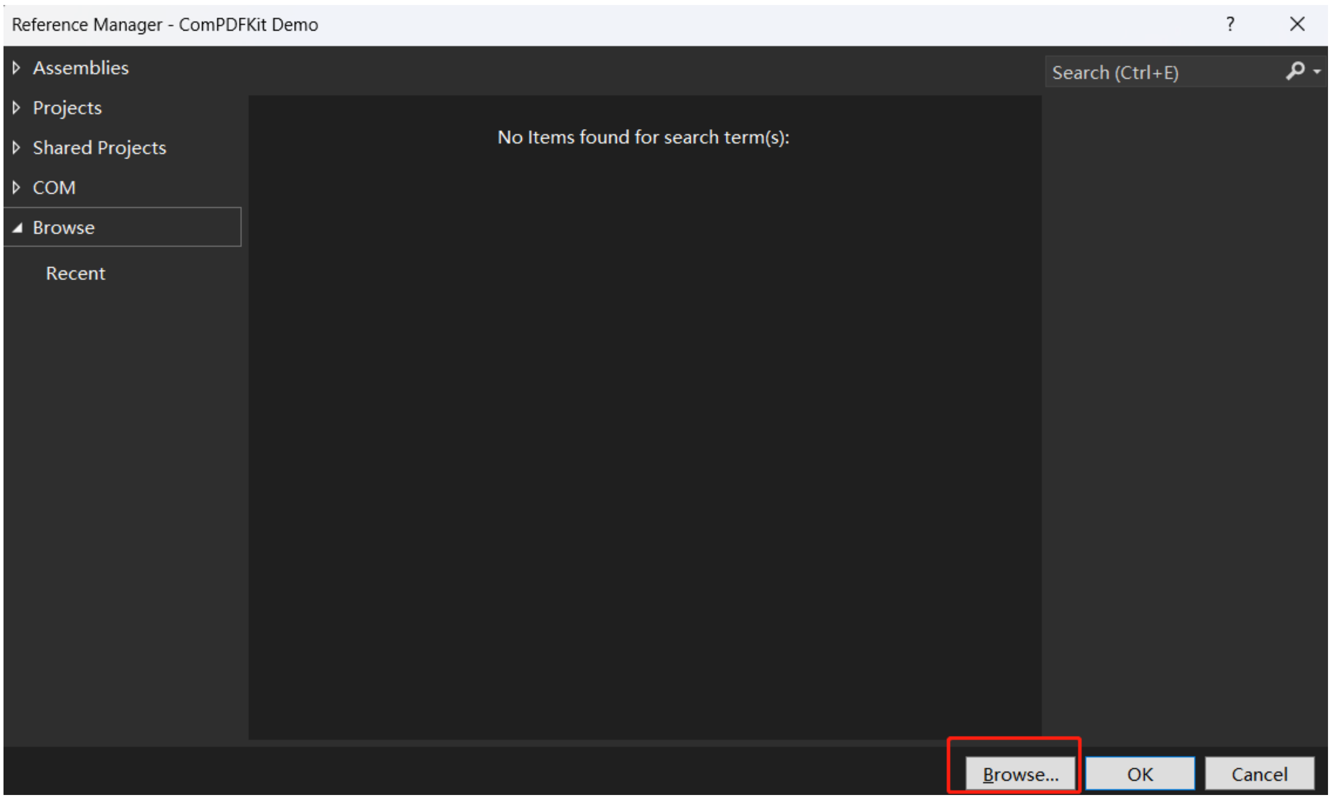
Now the structure of your project will look like this:



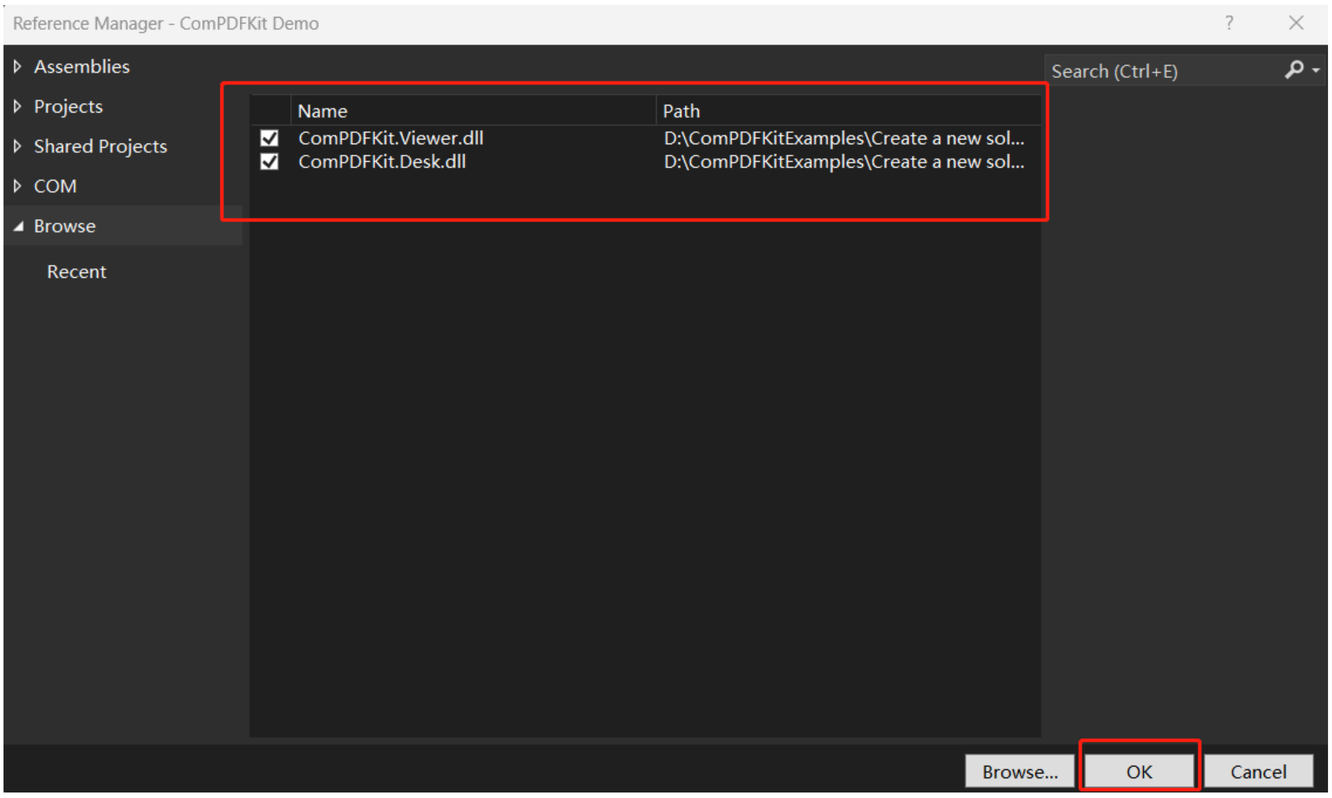
3. To use the APIs of ComPDFKit PDF SDK in your project, follow the instructions and add them to **Reference**. Right click the project that you need to add ComPDFKit PDF SDK and click **Add**. Then, click **Reference**.



In the Reference dialog, choose **Browse**, click another **Browse** button in the bottom right corner, and navigate to the ***"ComPDFKit Demo"*** folder which is in your project. Select ***"ComPDFKit.Desk.dll"*** and ***"ComPDFKit.Viewer.dll"*** dynamic library. Then, click **OK**.

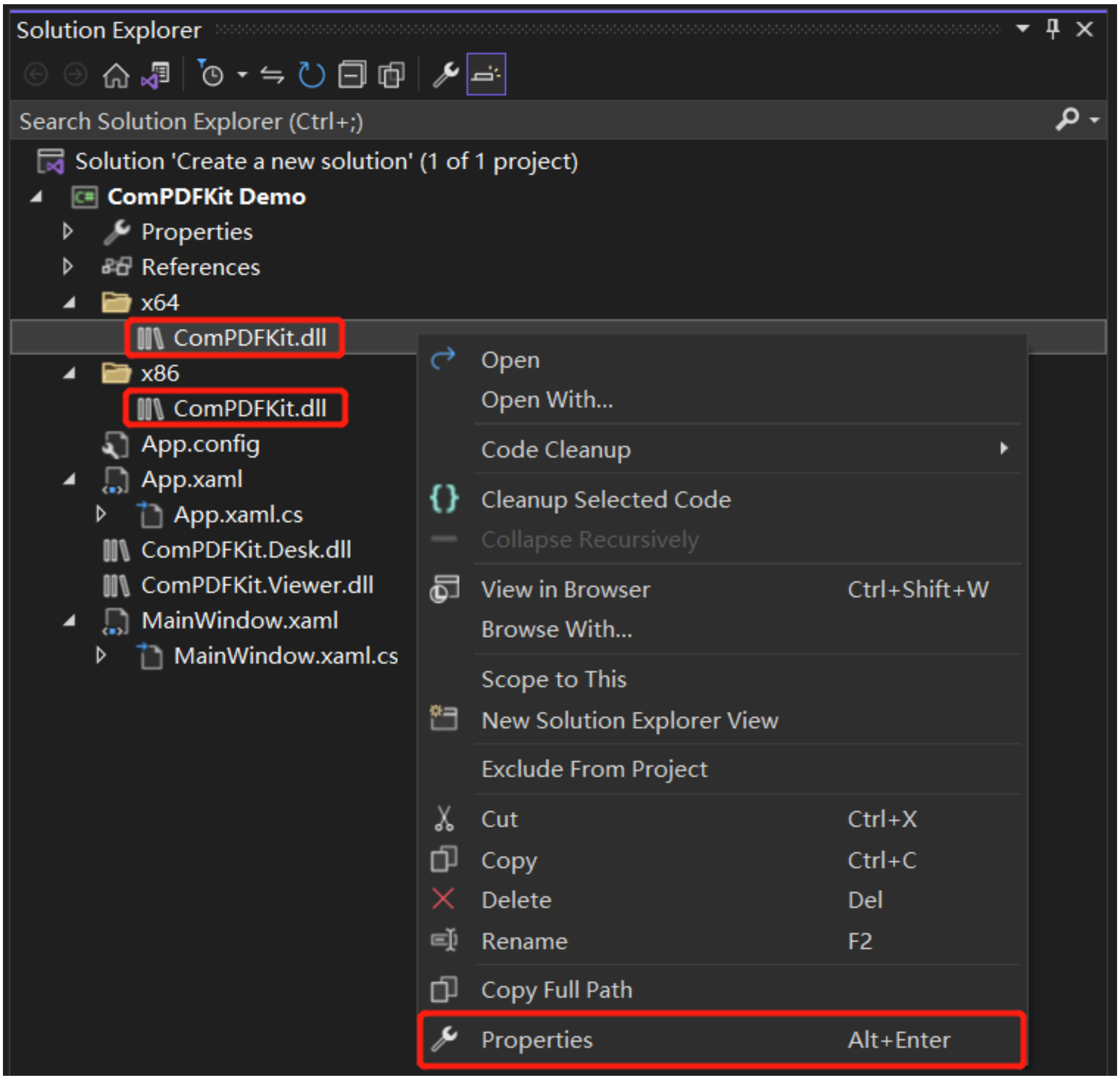


No Items found for search term(s):

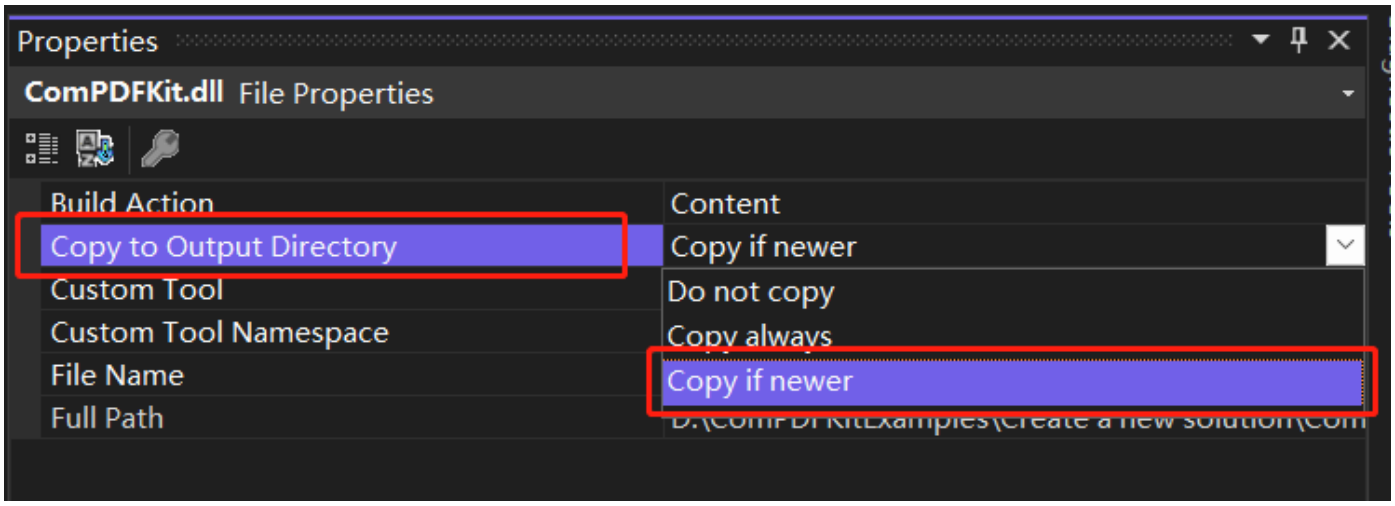


	Name	Path
✓	ComPDFKit.Viewer.dll	D:\ComPDFKitExamples\Create a new sol...
✓	ComPDFKit.Desk.dll	D:\ComPDFKitExamples\Create a new sol...

Right click ***"ComPDFKit.dll"*** -> click **Properties**.



Please make sure to set the property **Copy to Output Directory** of ***"ComPDFKit.dll"*** to **Copy if newer**. Otherwise, you should copy it to the same folder with the executable file manually before running the project.



2.4.3 Apply the License Key

You can contact ComPDFKit team to get a trial license. Before using any ComPDFKit PDF SDK classes, a required operation is to set the license key. Add the following method — `LicenseVerify()` to ***"MainWindow.xaml.cs"***.

```
bool LicenseVerify()
{
    if (!CPDFSDKVerifier.LoadNativeLibrary())
        return false;

    LicenseErrorCode verifyResult =
CPDFSDKVerifier.LicenseVerify("license_key_windows.txt", true);
    return (verifyResult == LicenseErrorCode.E_LICENSE_SUCCESS);
}
```

2.4.4 Display a PDF Document

We have finished all prepare steps. Let's display a PDF file.

Add the following code to ***"MainWindow.xaml"*** and ***"MainWindow.xaml.cs"*** to display a PDF document. Please make sure to replace "ComPDFKit_Demo" with the name of your project. Now, all you need to do is to create a `CPDFViewer` object, and then display the `CPDFViewer` object in the Grid (component) named "PDFGrid" using the `OpenPDF_Click` method.

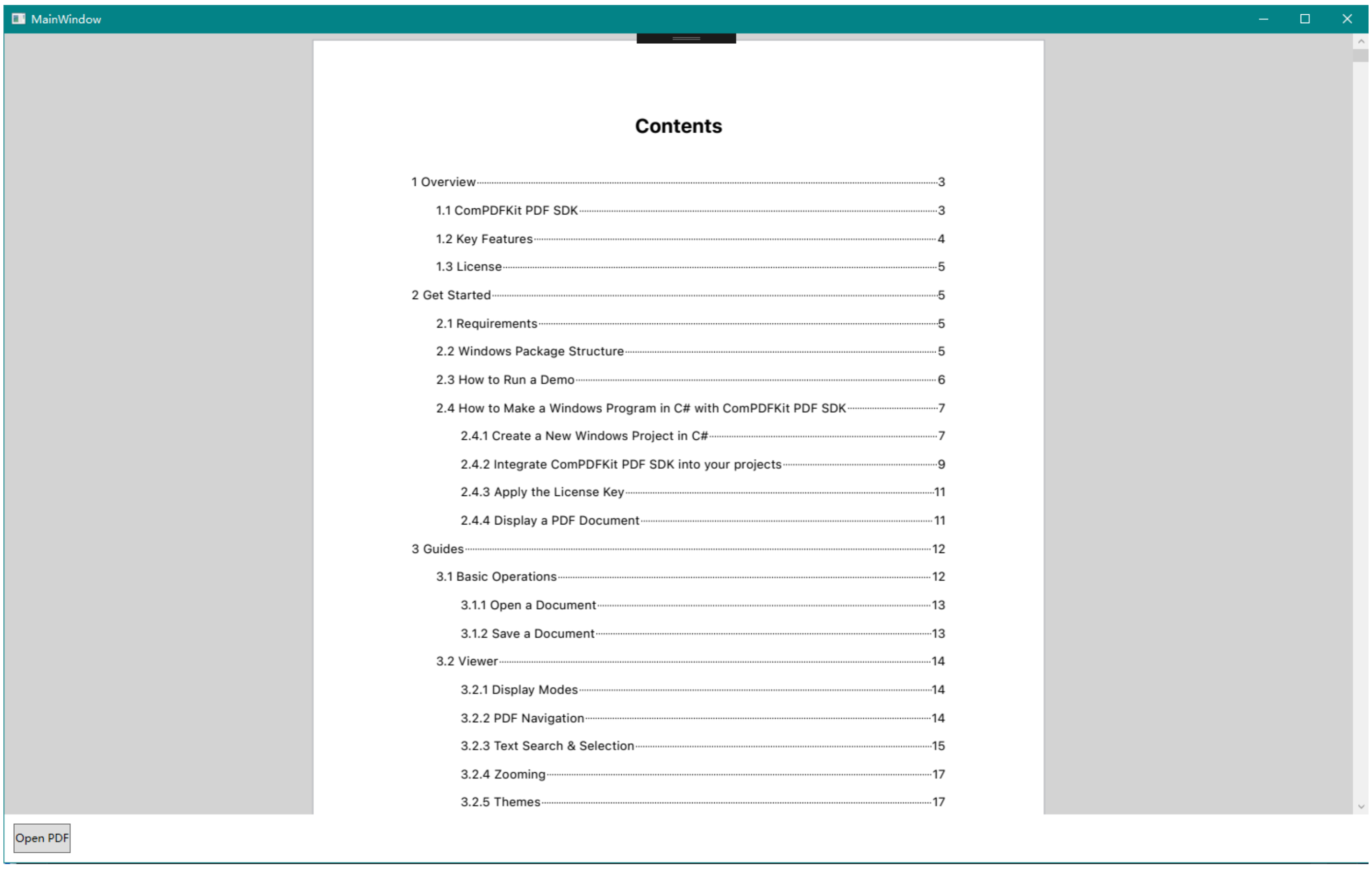
Now your ***"MainWindow.xaml"*** should look like the following code.

```
<Window x:Class="ComPDFKit_Demo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ComPDFKit_Demo"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800" UseLayoutRounding="True">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="52"/>
        </Grid.RowDefinitions>
        <Grid Name="PDFGrid" Grid.Row="0" />
        <Button Content="Open PDF" Grid.Row="1" HorizontalAlignment="Left" Margin="10"
Click="OpenPDF_Click"/>
    </Grid>
</Window>
```

Now your ***"MainWindow.xaml.cs"*** should look like the following code. Please note: You need to enter your license key. All the places that need to be modified in the code have been marked with comments in the code below. You just need to replace the string content below the comments by yourself.

```
using ComPDFKit.NativeMethod;
using ComPDFKit.PDFDocument;
using ComPDFKitViewer.PdfViewer;
using Microsoft.Win32;
using System.Windows;

namespace ComPDFKit_Demo
{
  public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            LicenseVerify();
        }

        bool LicenseVerify()
        {
            if (!CPDFSDKVerifier.LoadNativeLibrary())
            return false;

        LicenseErrorCode verifyResult =
CPDFSDKVerifier.LicenseVerify("license_key_windows.txt", true);
        return (verifyResult == LicenseErrorCode.E_LICENSE_SUCCESS);
        }

        private void OpenPDF_Click(object sender, RoutedEventArgs e)
        {
            // Get the path of a PDF file.
            var dlg = new OpenFileDialog();
            dlg.Filter = "PDF Files (*.pdf)|*.pdf";
            if (dlg.ShowDialog() == true)
            {
                // Use the PDF file path to open the document in CPDFViewer.
                CPDFViewer pdfViewer = new CPDFViewer();
                pdfViewer.InitDocument(dlg.FileName);
                if (pdfViewer.Document != null &&
                    pdfViewer.Document.ErrorType ==
CPDFDocumentError.CPDFDocumentErrorSuccess)
                {
                    pdfViewer.Load();
                    PDFGrid.Children.Add(pdfViewer);
                }
            }
        }
        }
    }
```

Now run the project and you will see the PDF file that you want to display. The PDF Viewer has been created successfully.


MainWindow

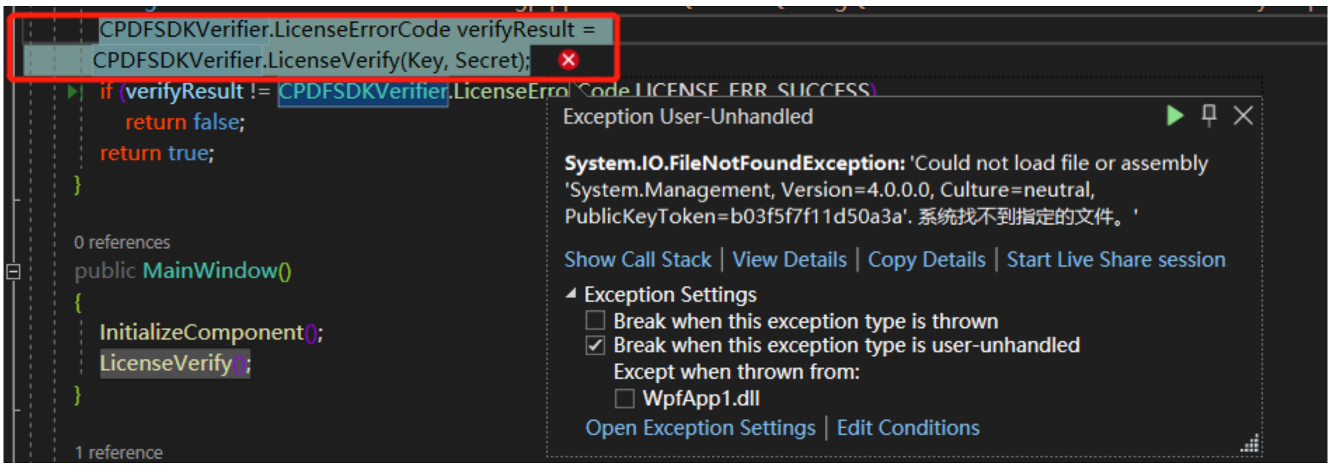
Contents

```
1 Overview ......................................................... 3
    1.1 ComPDFKit PDF SDK ............................................ 3
    1.2 Key Features ................................................. 4
    1.3 License ...................................................... 5
2 Get Started ...................................................... 5
    2.1 Requirements ................................................. 5
    2.2 Windows Package Structure .................................... 5
    2.3 How to Run a Demo ............................................ 6
    2.4 How to Make a Windows Program in C# with ComPDFKit PDF SDK ... 7
        2.4.1 Create a New Windows Project in C# ..................... 7
        2.4.2 Integrate ComPDFKit PDF SDK into your projects ......... 9
        2.4.3 Apply the License Key .................................. 11
        2.4.4 Display a PDF Document ................................. 11
3 Guides ........................................................... 12
    3.1 Basic Operations ............................................. 12
        3.1.1 Open a Document ........................................ 13
        3.1.2 Save a Document ........................................ 13
    3.2 Viewer ....................................................... 14
        3.2.1 Display Modes .......................................... 14
        3.2.2 PDF Navigation ......................................... 14
        3.2.3 Text Search & Selection ................................ 15
        3.2.4 Zooming ................................................ 17
        3.2.5 Themes ................................................. 17
```

Open PDF

2.4.5 Troubleshooting

1. If "System.IO.FileNotFoundException" occurred in the `LicenseVerify()` function like this:


```
CPDFSDKVerifier.LicenseErrorCode verifyResult =
 CPDFSDKVerifier.LicenseVerify(Key, Secret);
if (verifyResult != CPDFSDKVerifier.LicenseErro Code.LICENSE_ERR_SUCCESS)
    return false;
return true;
}

0 references
public MainWindow()
{
    InitializeComponent();
    LicenseVerify();
}

1 reference
```

Exception User-Unhandled

System.IO.FileNotFoundException: 'Could not load file or assembly 'System.Management, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a'. 系统找不到指定的文件。'

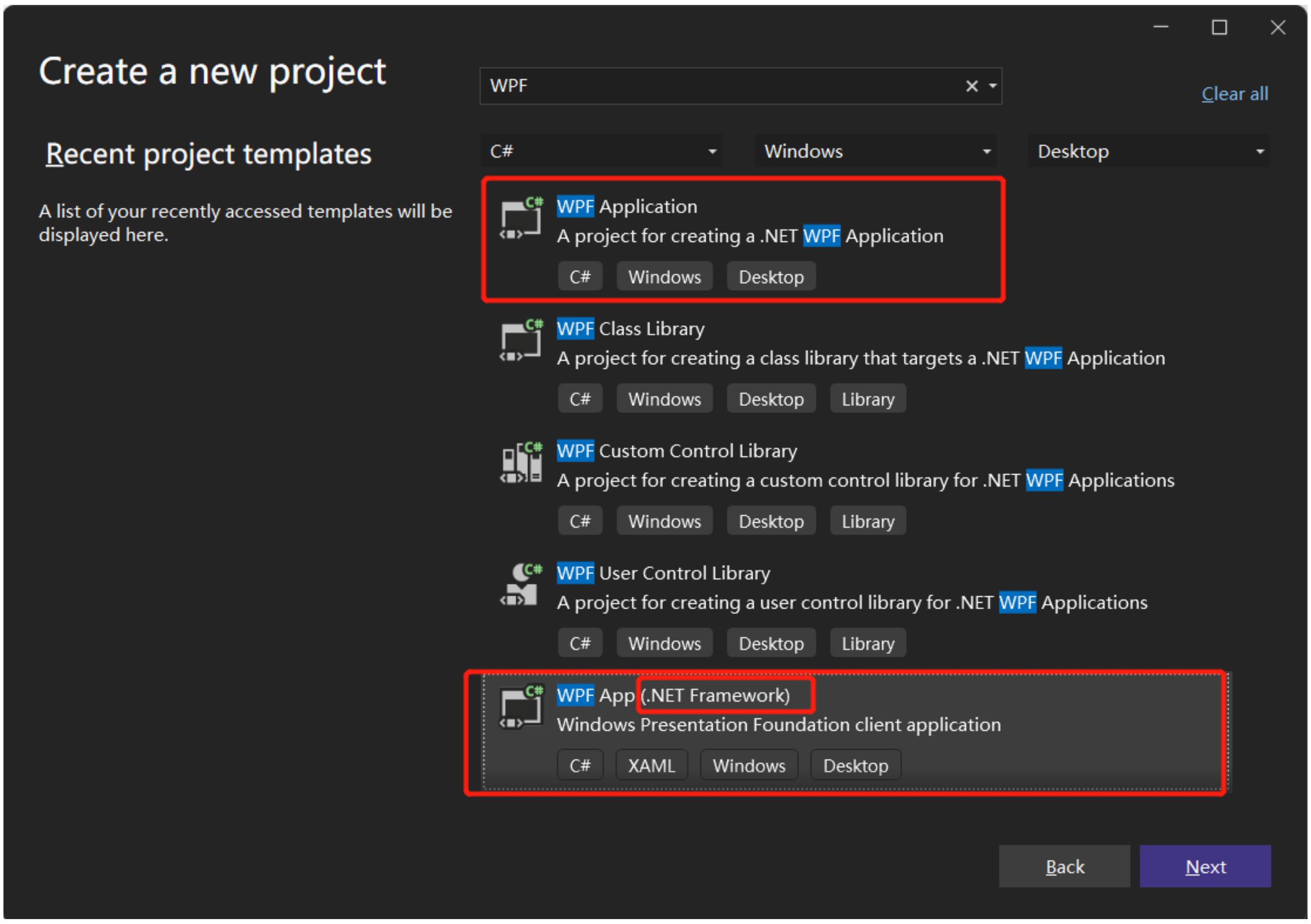
Show Call Stack | View Details | Copy Details | Start Live Share session

Exception Settings
- [ ] Break when this exception type is thrown
- [x] Break when this exception type is user-unhandled

  Except when thrown from:
  - [ ] WpfApp1.dll

Open Exception Settings | Edit Conditions

Check your WPF project and ensure that you chose **WPF App(.NET Framework)** instead of **WPF Application** when creating the project.


Create a new project
WPF
Clear all
Recent project templates
A list of your recently accessed templates will be displayed here.
C#
Windows
Desktop
WPF Application
A project for creating a .NET WPF Application
C# Windows Desktop
WPF Class Library
A project for creating a class library that targets a .NET WPF Application
C# Windows Desktop Library
WPF Custom Control Library
A project for creating a custom control library for .NET WPF Applications
C# Windows Desktop Library
WPF User Control Library
A project for creating a user control library for .NET WPF Applications
C# Windows Desktop Library
WPF App (.NET Framework)
Windows Presentation Foundation client application
C# XAML Windows Desktop
Back
Next

2. Other Problems

   If you meet some other problems when integrating our ComPDFKit PDF SDK for Windows, feel free to contact ComPDFKit team.

2.5 UI Customization

The folder of ***"ComPDFKit_Tools"*** includes the UI components to help conveniently integrate ComPDFKit PDF SDK. We have also built six standalone function programs, namely **Viewer**, **Annotations**, **ContentEditor**, **Forms**, **DocsEditor**, and **Digital Signature**, using this UI component library. Additionally, we have developed a program called **PDFViewer** that integrates all the above-mentioned example features for reference.

In this section, we will introduce how to use it from the following parts:

1. Overview of ***"ComPDFKit_Tools"*** Folder: Show the folder structure and the main features included in the corresponding component.
2. UI Components: Introduce the UI components and how to use them easily and fast.

2.5.1 Overview of *"ComPDFKit_Tools"* Folder

There are seven modules in ***"ComPDFKit_Tools"***: ***"Common"***, ***"Viewer"***, ***"Annotations"***, ***"ContentEditor"***, ***"Forms"***, ***"DocsEditor"***, and ***"Digital Signatures"***. Each of them includes the code and UI components like the following table to process PDFs.

Folder	SubFolder	Description
Common	BaseControl	Basic components used to compose other components, such as the value component to control the value of opacity, font size, border width, etc.
	Convert	Data converter
	Helper	Static classes and static methods that provide assistance for common functions, such as a static method that can invoke a file open dialog and get the selected PDF file path: GetFilePathOrEmpty.
	PasswordControl	Include the UI components and interaction of typing file passwords.
	PropertyControl	Include the UI components and interaction of handling specified data type inputting.
PDFView	PDFBookmark	Include the UI components and interaction of editing bookmarks and jumping pages.
	PDFInfo	Include the UI components and interaction of document information.
	PDFDisplaySettings	Include the UI components and interaction of PDF viewing like setting themes, display modes, etc.
	PDFOutline	Include the UI components and interaction of jumping and displaying the PDF outline.
	PDFSearch	Include the UI component and interaction for searching PDFs and generating the search list.
	PDFThumbnail	Include the UI component and interaction of PDF thumbnails.
Annotations	PDFAnnotationBar	Include the toolbar that indicates the required annotation type and order. Clicking on the navigation bar will pass the corresponding comment type enumeration through an event.
	PDFAnnotationPanel	When creating or modifying annotations, specific property panels are displayed, and controls for

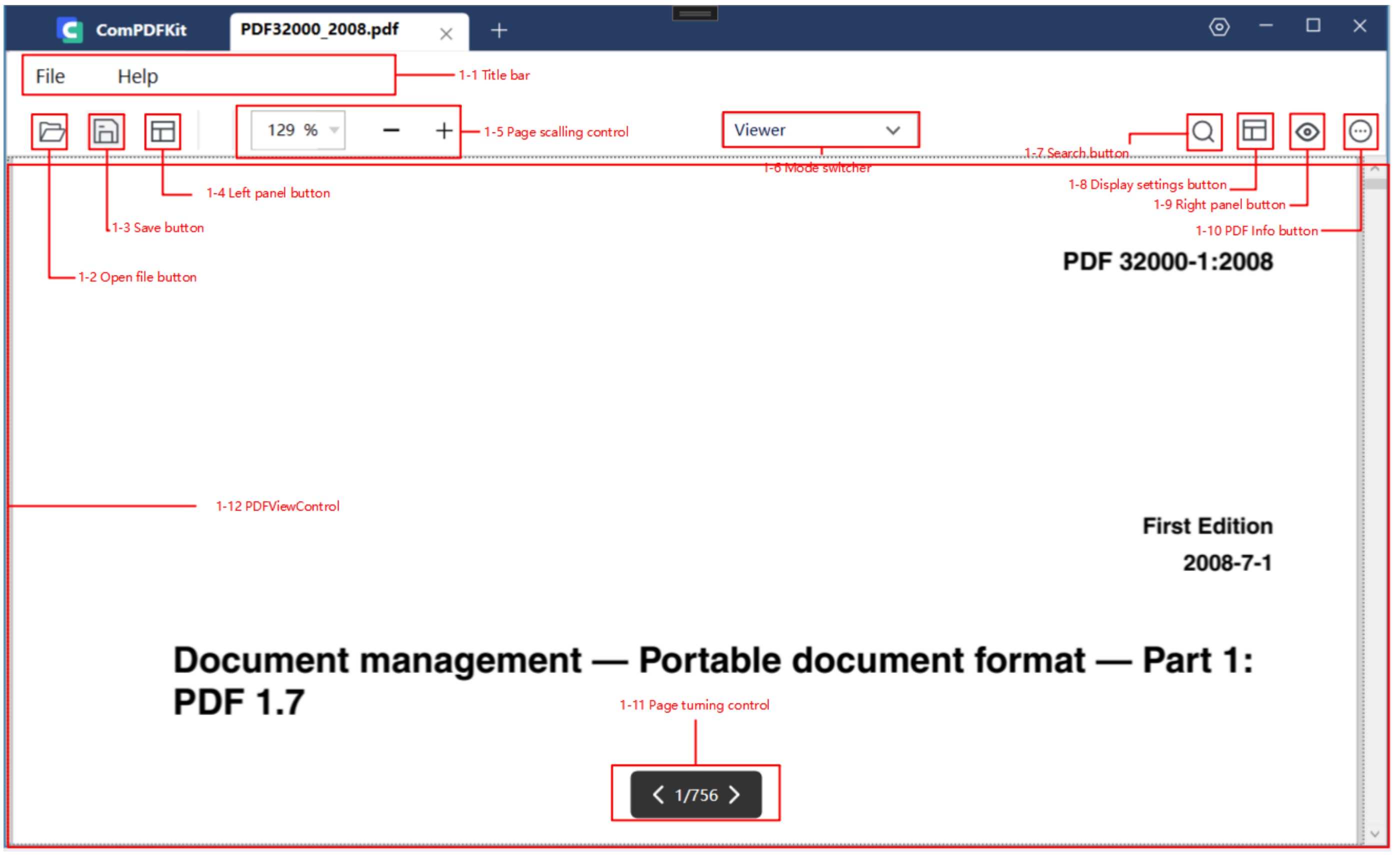
		handling data are provided.
	PDFAnnotationList	Include the UI component and interaction of displaying all annotations in a list, selecting and deleting specific annotations/all annotations.
ContentEditor	PDFImageEditControl	Include the toolbar to edit PDF images and undo/redo the processing of editing PDF images.
	PDFTextEditControl	Include the toolbar to edit PDF text and undo/redo the processing of editing PDF text.
Forms	FormBarControl	Include the UI component and interaction of specifying needed form fields and the order of displaying the form field types in UI.
	FormPropertyControl	Include the property panel and interaction to set the properties of forms.
Docs Editor	PDFPageEditBar	Include the toolbar for creating, replacing, rotating, extracting, and deleting PDF pages
	PDFPageEdit	Include the UI component and interaction of document editing like thumbnails, drag, right-click menu, etc.
	PDFPageExtract	Include the popup window of page extraction. It only processes and transfers the data. You can refer to PDFPageEdit for inserting pages.
	PDFPageInsert	Include the popup window of page insertion. It only processes and transfers the data. You can refer to PDFPageEdit for inserting pages.
Digital Signatures	AddCertificationDialog	Include the popup window to create new certificates or using existing certificates.
	CPDFSignatureListControl	Include the UI component and interaction of displaying the list of digital signatures and their status, with options to navigate to a specific signature location or open a signature status popup.
	VerifyDigitalSignatureControl	Include the popup window to display the signature status.
	SignatureStatusBarControl	Include the popup window to display all the signature statuses in this file.
	FillDigitalSignatureDialog	Include the popup window to create the signature appearance.

	ViewCertificateDialog	Include the popup window to view the signature certificates.

2.5.2 UI Component

This section mainly introduces the connection between the UI components and API configuration of ***"ComPDFKit_Tools"***, which can not only help you quickly get started with the default UI but also help you view the associated API configuration. These UI components could be used and modified to create your customize UI.

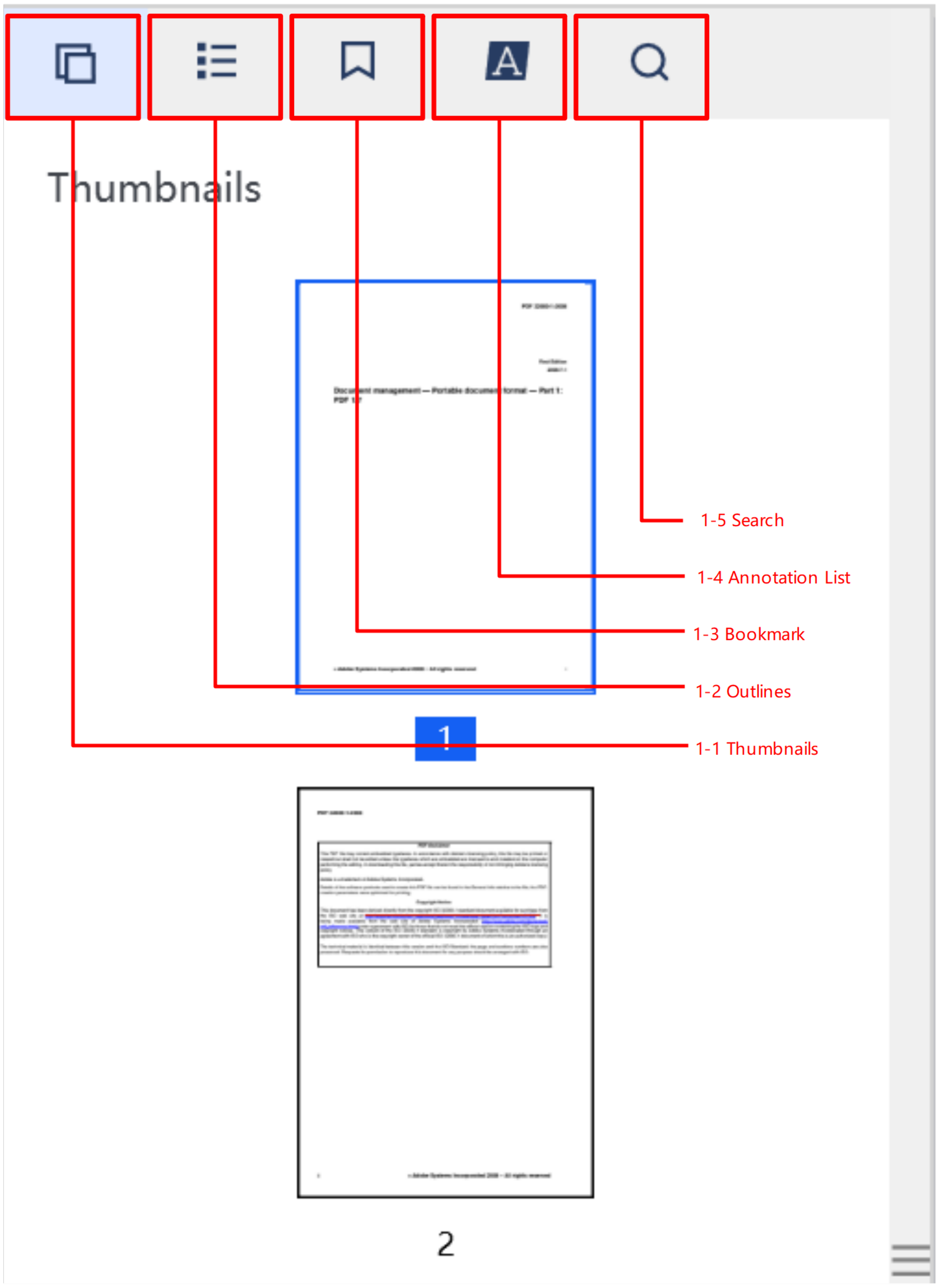
Part 1:



The picture above shows the main UI components associated with the API of Viewer, which are also the fundamental UI components of ***"ComPDFKit_Tools"***. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Title bar	CPDFTitleBarControl	The toolbar at the top of the PDF view window: Include the help center and file center.
1-2	Open file button	/	Control to switch a new document.
1-3	Save button	/	Control to save the current file.
1-4	Right panel button	CPDFBOTABarControl	Control the display status of the property panel.
1-5	Page scalling control	CPDFScalingControl	Control to change the zoom ratio of PDF.
1-6	Mode switcher	/	Switch the feature modules.
1-7	Search button	CPDFBOTABarControl	Enter the searching mode.
1-8	Display settings button	CPDFDisplaySettingsControl	Control to show or hide the setting panel.
1-9	Left panel button	CPDFAnnotationControl、FromPropertyControl、PDFContentEditControl	Control the displaying status of property panel.
1-10	PDF info button	CPDFInfoControl	Control the popup window of document information.
1-11	Page turning control	PageNumberControl	Control to jump to other specific pages quickly.
1-12	PDFViewControl	PDFViewControl	Basic interactions like zooming PDF view with mouse and executing page jumping or push button actions.

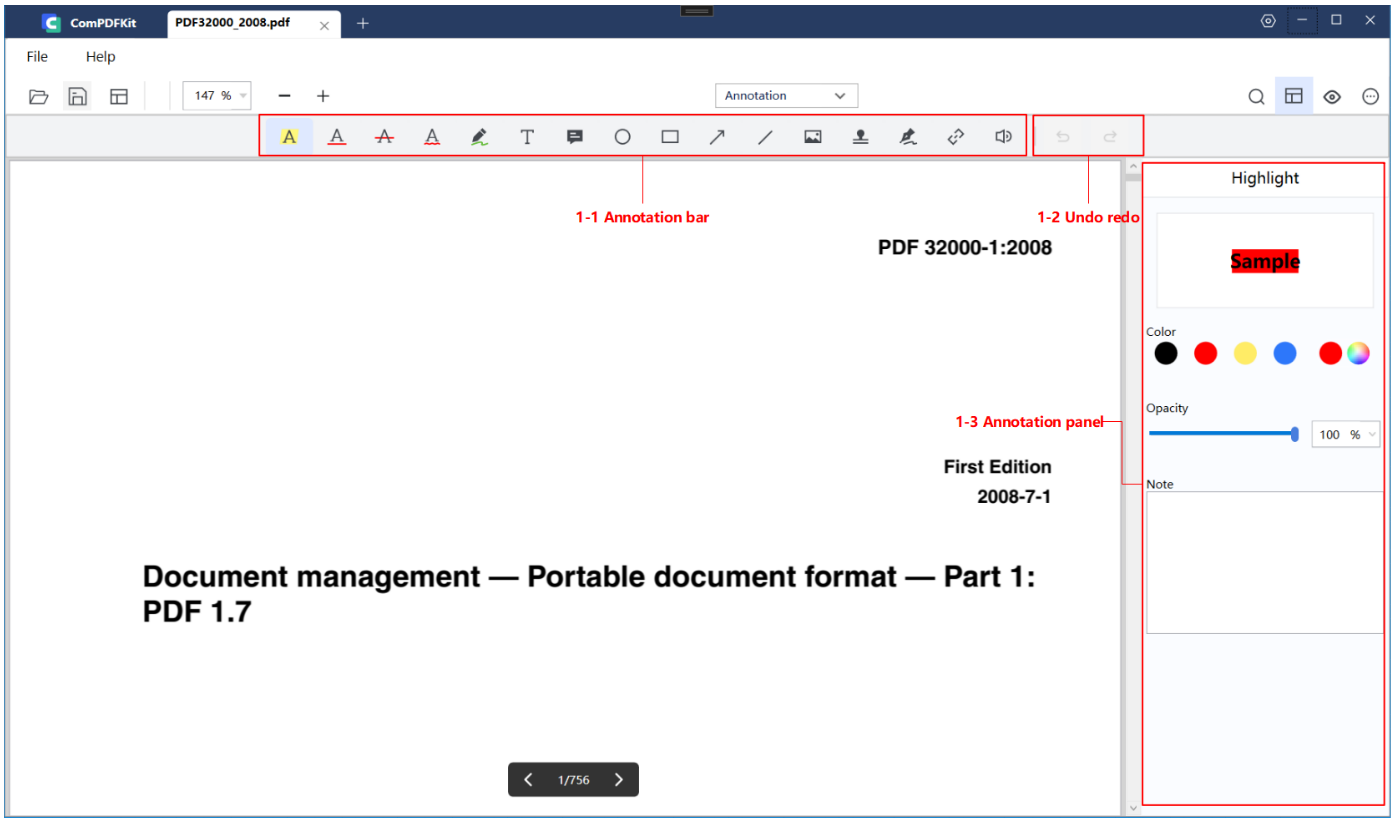
Part 2:



The picture above shows the UI components associated with the API of outline, bookmark, thumbnail, annotation list, and searching. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Thumbnails	CPDFThumbnailControl	Enter the thumbnails of PDFs.
1-2	Outlines	CPDFOutlineControl	Enter the outlines of PDFs.
1-3	Bookmark	CPDFBookmarkControl	Enter the bookmark list of PDFs.
1-4	Annotation List	CPDFThumbnailControl	Enter the annotation list of PDFs.
1-5	Search	CPDFSearchControl	Enter the PDF keywords searching.

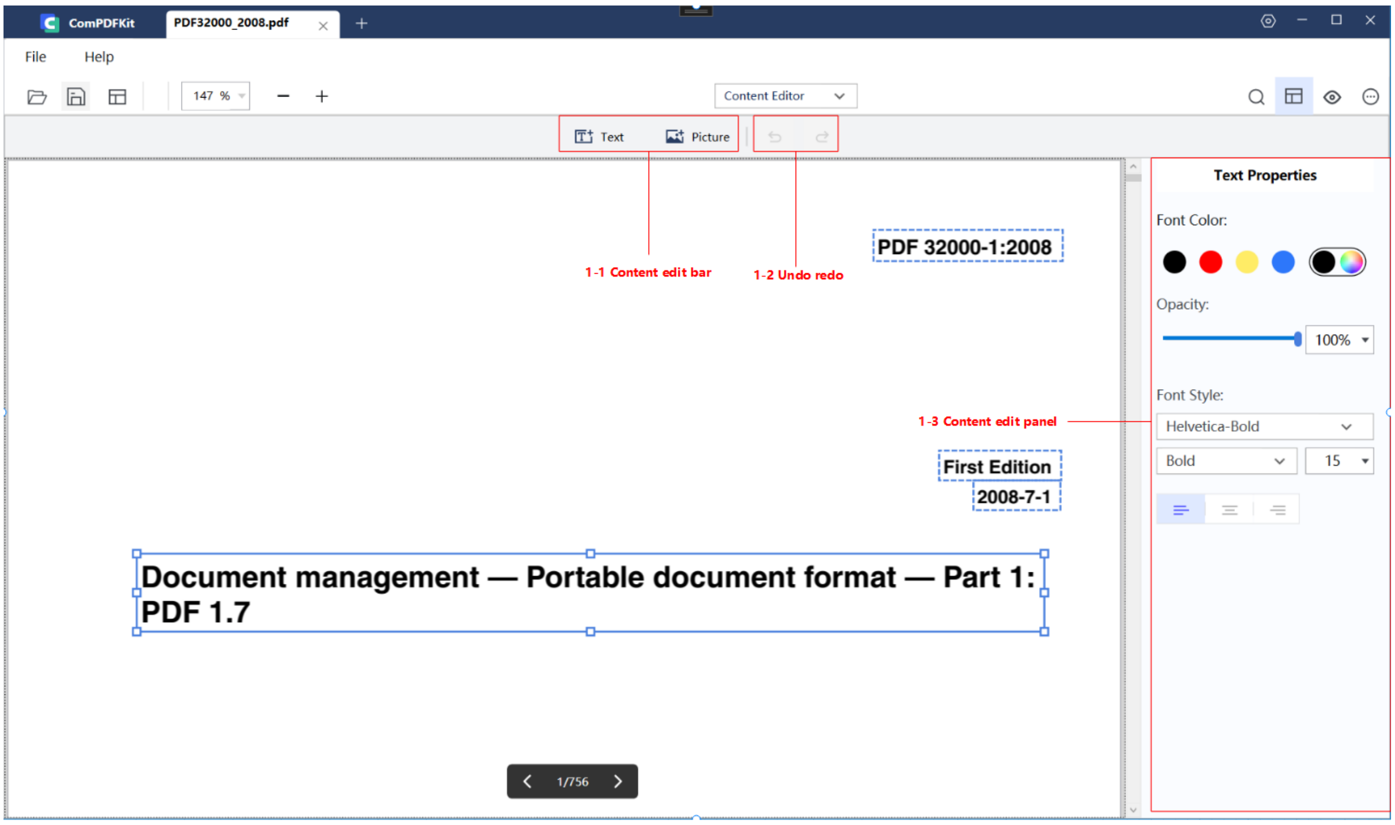
Part 3:



The picture above shows the UI components associated with the API of annotation. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Annotation bar	CPDFAnnotationBarControl	Annotated toolbar, allowing specifying the annotation types and the order of displaying the annotation types in UI.
1-2	Undo redo	/	Undo/redo the processing of annotations.
1-3	Annotation panel	CPDFAnnotationControl	Preset annotation properties for creating annotation.

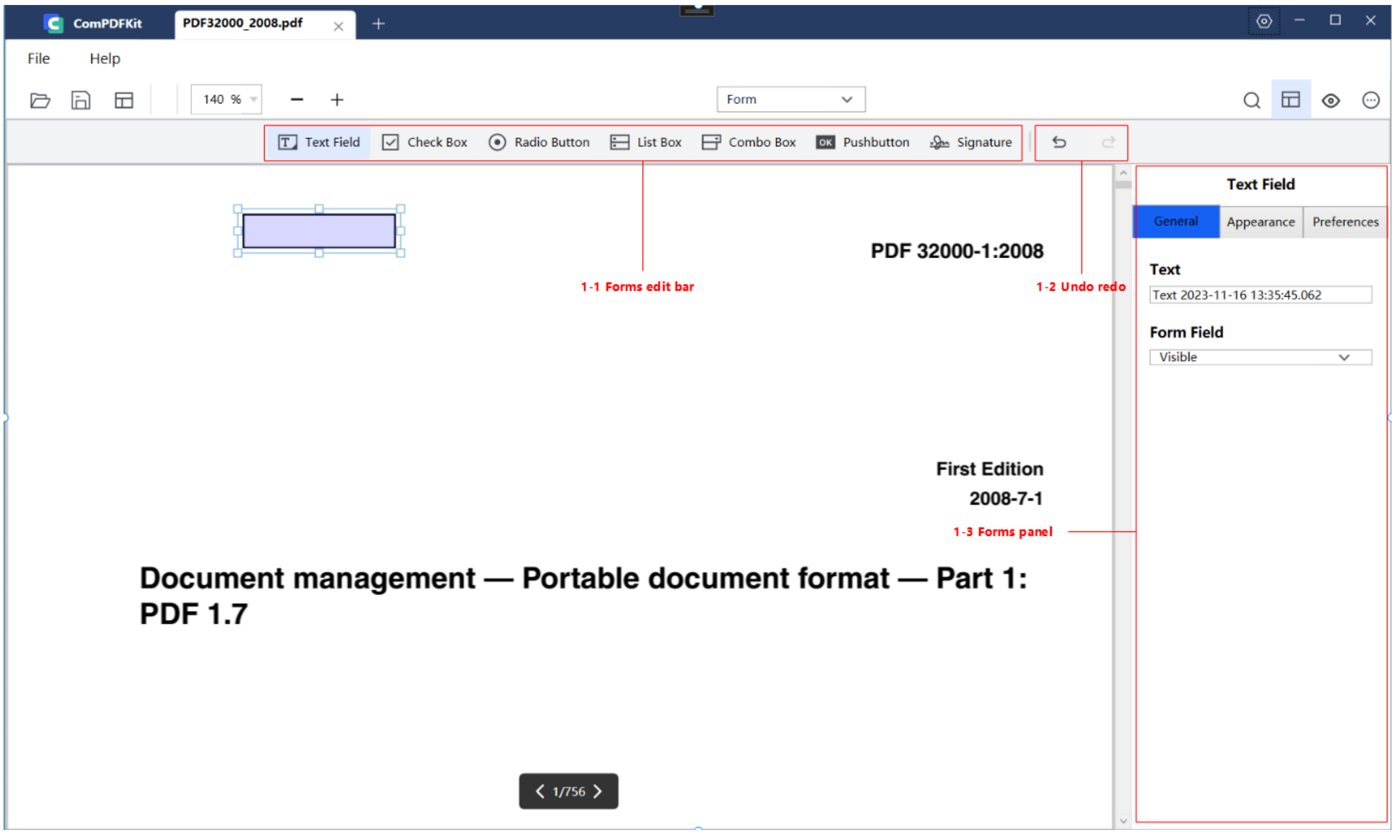
Part 4:



The picture above shows the UI components associated with the API of the content editor. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Content edit bar	/	The tool bar of content editor. Enter a state of creating text and only text editing, after clicking on text editing. After clicking on image editing, you can add images. After adding the images, you will enter the default mode, in which both images and text can be edited.
1-2	Undo redo	/	Undo redo the processing of editing PDF text/images.
1-3	Content edit panel	PDFContentEditControl	Preset text properties for adding text. After clicking on text or image, you will get the attributes of the currently selected object and can modify them.

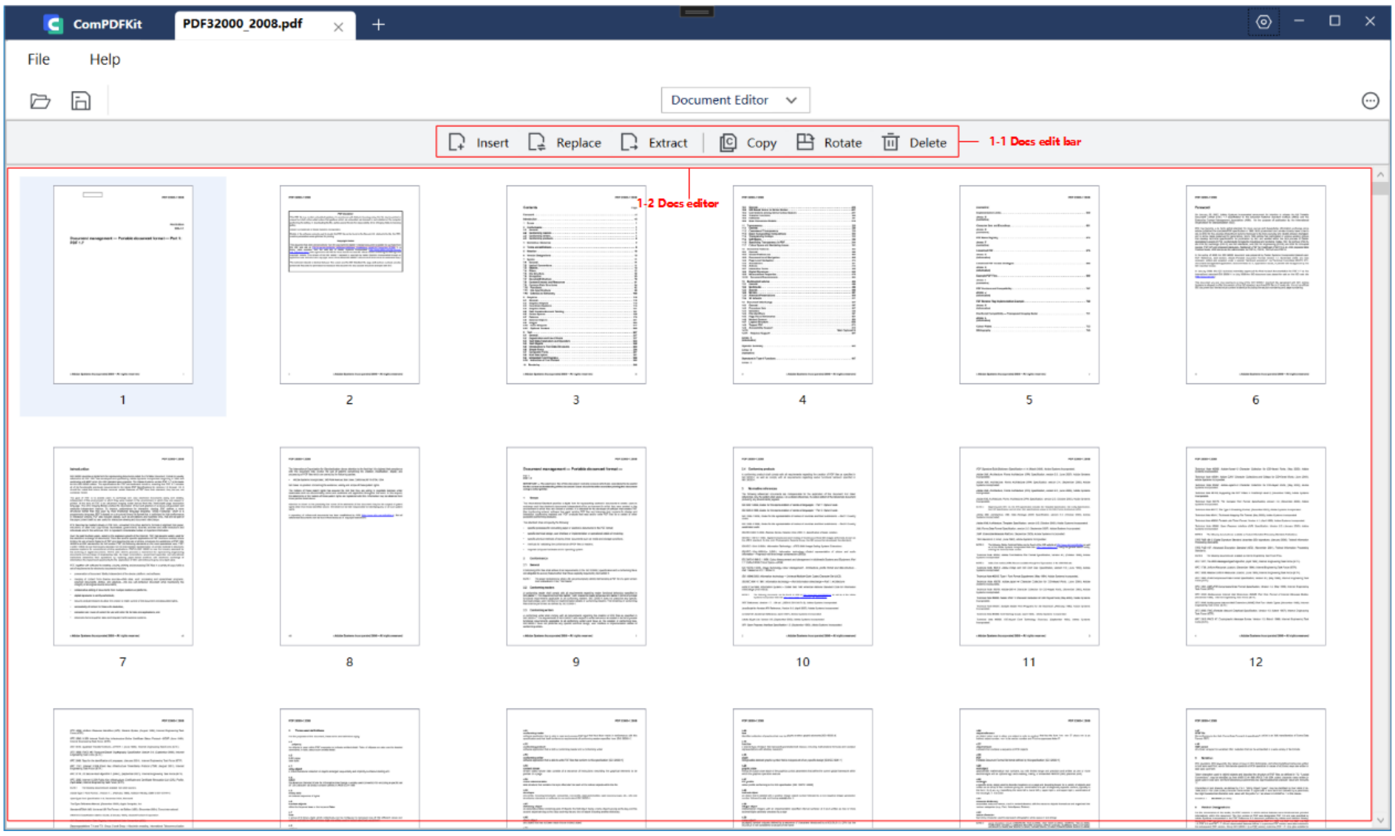
Part 5:



The picture above shows the UI components associated with the API of forms. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Forms edit bar	CPDFFormBarControl	Tool bar of PDF forms.
1-2	Undo redo	/	Undo/redo the processing of forms.
1-3	Forms panel	FromPropertyControl	Set the properties of forms.

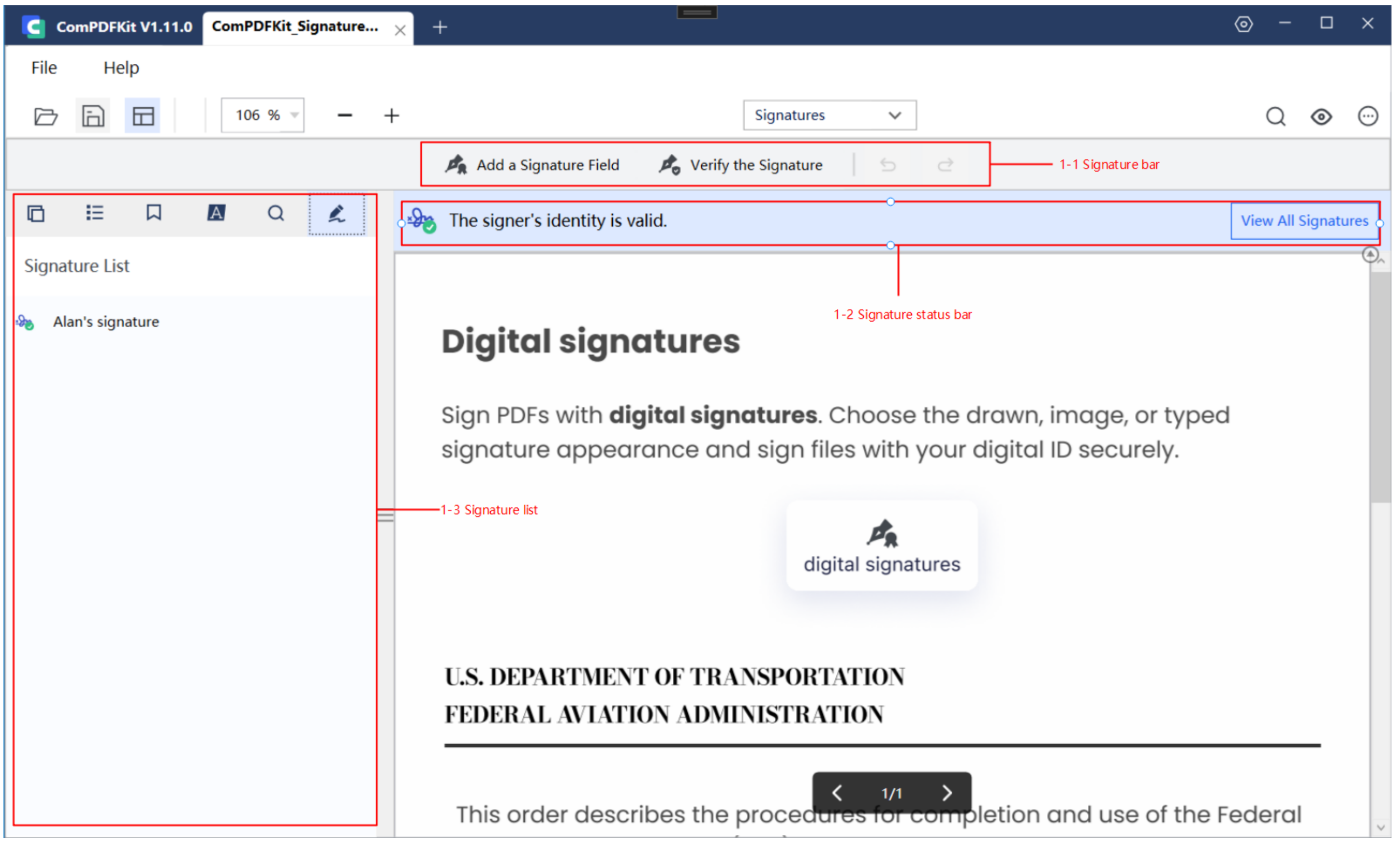
Part 6:



The picture above shows the UI components associated with the API of the document editor. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Docs edit bar	CPDFPageEditBarControl、CPDFPageExtractWindow、CPDFPageInsertWindow	Tool bar of document editor.
1-2	Docs editor	CPDFPageEditControl	Show the thumbnails of PDF pages and interaction for editing PDF pages.

Part 7:


ComPDFKit V1.11.0
ComPDFKit_Signature...
File
Help
106 %
Signatures
Add a Signature Field
Verify the Signature
1-1 Signature bar
The signer's identity is valid.
View All Signatures
1-2 Signature status bar
Signature List
Alan's signature
1-3 Signature list
Digital signatures
Sign PDFs with digital signatures. Choose the drawn, image, or typed signature appearance and sign files with your digital ID securely.
digital signatures
U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION
1/1
This order describes the procedures for completion and use of the Federal

The picture above shows the UI components associated with the API of the digital signature. The following table shows the details of the connection between UI components and APIs.

Number	Name	Functionality	Description
1-1	Signature bar	AddCertificationDialog、FillDigitalSignatureDialog	Add signature field, and verify all the signatures.
1-2	Signature status bar	SignatureStatusBarControl	Show the status of all the digital signatures.
1-3	Signature list	ViewCertificateDialog	A list to display all the digital signatures in PDFs.

2.6 Samples

The Samples use preset parameters and documentation to call the API of ComPDFKit PDF SDK for each function without UI interaction or parameter settings. This is achieved through modular code examples. The functions include creating, getting, and deleting various types of annotations and forms, extracting text and images, encrypting and decrypting documents, adding watermarks and bates numbers, and more.

These projects not only demonstrate the best practices for each function but also provide detailed introductions. The impact of each function on PDF documents can be observed in the output directory. With the help of the Samples, you can quickly learn how to use the functions you need and apply them to your projects.

Name	Description
Bookmark	Create a new bookmark, and access the existing bookmark.
Outline	Create a new outline, and get existing outline information.
PDFToImage	Convert PDF pages to PNG.
TextSearch	Perform full-text search and highlight keywords.
Annotation	Print the annotation list information, set the annotations (including markup, note, ink, free text, circle, square, line, stamp, and sound annotations), and delete the annotations.
AnnotationImportExport	Export and import annotations with an xfdf file.
InteractiveForms	Print form list information, set up interactive forms (including text, checkbox, radio button, button, list, combo boxes, signing and deleting forms), and fill out form information.
PDFPage	Manipulate PDF pages, including inserting, splitting, merging, rotating, and replacing, etc.
ImageExtract	Extract images from a PDF document.
DocumentInfo	Display the information of PDF files like the author, created time, etc.
Watermark	Create text/image watermarks and delete watermarks.
Background	Create a color/image background and delete the background.
Bates	Create and remove bates numbers.
PDFRedact	Create redaction to remove sensitive information or private data, which cannot be viewed and searched once applied.
Encrypt	Set passwords to encrypt PDFs and set document permissions. Allow decrypting PDFs.
PDFA	Convert PDF to PDF/A-1a and PDF/A-1b.
Flatten	Flatten PDF annotations and forms, and merge all layouts as one layout.
DocumentCompare	Compare different documents and show the result in new documents.
DigitalSignatures	Create, fill, and verify the signatures and certificates. Read the details of signatures and certificates. Remove digital signatures.

3 Guides

If you're interested in all of the features mentioned in Overview section, please go through our guides to quickly add PDF viewing, annotating, and editing to your application. The following sections list some examples to show you how to add document functionalities to Windows apps using our C# APIs.

3.1 Basic Operations

There are a few commonly used basic operations when working with documents.

3.1.1 Open a Document

- Open a Local File

```
// Get the path of a PDF
string filePath ="";
var dlg = new OpenFileDialog();
dlg.Filter = "PDF Files (*.pdf)|*.pdf";
if (dlg.ShowDialog() == true)
{
    filePath = dlg.FileName;
}
else
{
    return;
}

// Initialize a CPDFDocument object with the path to the PDF file
CPDFDocument document = CPDFDocument.InitWithFilePath(filePath);
if(document==null)
{
    return;
}

if(document.ErrorType != CPDFDocumentError.CPDFDocumentErrorSuccess
  && document.ErrorType != CPDFDocumentError.CPDFDocumentPasswordError)
{
    return;
}
```

- Create a New File

```
CPDFDocument document = CPDFDocument.CreateDocument();
```

Following are the descriptions of error codes in `CPDFDocumentError`:

Error	Description
CPDFDocumentErrorSuccess	Open PDF file successfully.
CPDFDocumentUnknownError	Unknown error.
CPDFDocumentFileError	File not found or could not be opened.
CPDFDocumentFormatError	File not in PDF format or corrupted.
CPDFDocumentPasswordError	Password required or incorrect password.
CPDFDocumentSecurityError	Security scheme not supported.
CPDFDocumentPageError	Page not found or content error.

3.1.2 Save a Document

To save a PDF document to path use `CPDFDocument.WriteToLoadedPath()`, the PDF document will be saved incrementally. Use this method if you are concerned about saving time. If you use this method, any changes to the document, even deleting annotations, will result in appending to the PDF file.

Incremental saving, which is enabled by default, is the strategy that ComPDFKit uses to ensure saving is as fast as possible. This strategy always appends changes to the end of the document, and it never deletes anything. However, even though this is the fastest way to save a document, it does come with the cost of increasing the file size with each save. You can read more about incremental saving and full saving of PDFs here.

In most cases, the file size increase is negligible. However, there might be some cases in which you want to prioritize file size over saving performance. Below you'll find a strategy to prevent the file size from growing unnecessarily when saving changes to a document.

To save as a PDF document to path use `CPDFDocument.WriteToFilePath(string filePath)`, the PDF document will be saved non-incremental. If you use this method, will rewrite the entire document instead of appending changes at the end.

3.2 Viewer

`CPDFViewer` scroll view is a scroll view component that can load and display PDF pages on screen on demand. It can handle various rendering and scaling operations of PDF pages, and maintain the clarity and sharpness of PDF pages. `CPDFViewer` scroll view usually supports multiple different layout modes, such as vertical scrolling, single page scrolling, and more. These layout modes can be configured according to users' needs to suit different application scenarios. When using `CPDFViewer` scroll view to view PDFs, users can also easily implement various functions such as zooming in and out, rotating, flipping, dragging, adding bookmarks, etc. It provides a simple way for developers to quickly embed a highly configurable PDF viewer into any Windows application.

3.2.1 Display Modes

- View Mode

  `CPDFViewer` supports a number of view modes, you can use `CPDFViewer.ChangeViewMode(ViewMode newMode)` to set View mode values are as follows:

  `ViewMode::Single`: Display one page at a time, swiping left and right to change pages.

  `ViewMode::SingleContinuous`: Display one page at a time, scrolling up and down to change pages.

  `ViewMode::Double`: Display two pages at a time, with odd-numbered pages on left, swiping left and right to change pages.

  `ViewMode::DoubleContinuous`: Display pages in two columns, with odd-numbered pages on left, scrolling up and down to change pages.

  `ViewMode::Book`: Display two pages at a time, with odd-numbered pages on right, swiping left and right to change pages.

  `ViewMode::BookContinuous`: Display pages in two columns, with odd-numbered pages on right, scrolling up and down to change pages.

- Crop Mode

  Automatically trim PDF white margins to resize pages. Crop mode can be enabled by setting the page crop mode to `true`.

  ```
  CPDFViewer.SetCropMode(true);
  ```

3.2.2 PDF Navigation

- Page Navigation

  After loading a PDF document, you can programmatically interact with it, which allows you to scroll to different pages or destinations. All of the interaction APIs are available on `CPDFViewer`, which is the core PDF viewer.

  - To scroll to the specified page, use the function `CPDFViewer.GoToPage(int pageIndex)`.
  - To go to the specified destination including a page and a point on the page specified in page space, use the function `CPDFViewer.GoToPage(int pageIndex, Point pagePoint)`.

- Outlines

  PDF outline is a function that creates a directory structure in PDF documents, which allows users to browse, locate, and navigate to the parts of a document they are interested in more quickly and conveniently. The outline can be created by nesting, organizing the document content by elements such as headings, chapters, paragraphs, etc., and adding hyperlinks to each element for easily jumping to the corresponding position. PDF outline is very useful when reading long documents, saving users' time and improving reading efficiency.

The outline structure is a tree-like hierarchy, so the function `CPDFDocument.GetOutlineRoot()` must be called first to get the root node of the entire outline before accessing the outline. Here, the "root node" is an abstract object which can only have some child outlines without the next sibling outline and any data (including outline data, destination data, and operation data).

After the root outline is retrieved, the following functions can be called to access other outlines:

  - To access the parent outline, use the function `CPDFOutline.GetParent()`.
  - To access the child outline, use the function `CPDFDocument.GetOutlineList()`.
  - To insert a new outline, use the function `CPDFOutline.InsertChildAtIndex(CPDFDocument document, int index)`.
  - To remove an outline, use the function `CPDFOutline.RemoveFromParent(CPDFDocument document)`.
  - To move an outline, use the function `CPDFOutline.MoveChildAtIndex(CPDFDocument document, CPDFOutline child, int index)`.
    re

- Bookmarks

  Since each bookmark is associated with a specific page, it provides the ability to link to a different page in a document allowing the user to navigate interactively from one part of the document to another.

  - To access bookmarks, use the function `CPDFDocument.GetBookmarkList()`.
  - To access a bookmark for a page, use the function `CPDFDocument.BookmarkForPageIndex(int pageIndex)`.
  - To add a new bookmark, use the function `CPDFDocument.AddBookmark(CPDFBookmark bookmark)`.
  - To remove a bookmark, use the function `CPDFDocument.RemoveBookmark(int pageIndex)`.

3.2.3 Text Search & Selection

- Text Search

  ComPDFKit PDF SDK offers developers an API for programmatic full-text search.

  There are some optional parameters provided when searching, such as ignoring case, matching whole words, and other Options parameter settings (It is case-insensitive and does not match whole words by default).

  To search text inside a document, create an instance of `CPDFTextSearcher`, passing in the loaded `CPDFTextPage` via its initializer. Searching can be triggered via calling `CPDFTextSearcher.FindStart(CPDFTextPage textPage, string keyword, C_Search_Options searchOption,int startIndex)`. To do the searching, use function `CPDFTextSearcher.FindNext(CPDFPage page,CPDFTextPage textPage,ref CRect rect,ref string content,ref int startIndex)`.

  Before triggering a search, you can configure various search options:

  - `C_Search_Options::Search_Case_Insensitive`: Case insensitive.
  - `C_Search_Options::Search_Case_Sensitive`: Case sensitive.
  - `C_Search_Options::Search_Match_Whole_Word`: Match whole word.

How to get the text area on a page by searching:

```
void SearchForPage(CPDFPage page,string searchKeywords, C_Search_Options option,ref
List<Rect> rects, ref List<string> strings)
{
    rects = new List<Rect>();
    strings = new List<string>();
    int findIndex = 0;

    CPDFTextPage textPage = page.GetTextPage();
    CPDFTextSearcher searcher = new CPDFTextSearcher();

    if (searcher.FindStart(textPage, searchKeywords, option, 0))
    {
        CRect textRect = new CRect();
        string textContent = "";
        while (searcher.FindNext(page, textPage, ref textRect, ref textContent, ref
findIndex))
        {
            strings.Add(textContent);
            rects.Add(new Rect(textRect.left, textRect.top, textRect.width(),
textRect.height()));
        }
    }
}
```

- Text Selection

  PDF text contents are stored in `CPDFPage` objects which are related to a specific page. `CPDFPage` class can be used to retrieve information about text in a PDF page, such as single character, single word, text content within specified character range or bounds and more.

  How to get the text bounds on a page by selection:

```
void SelectForPage(CPDFPage page, Point fromPoint, Point toPoint, ref List<Rect>
rects, ref string textContent)
{
    CPDFTextPage textPage = page.GetTextPage();
    textContent = textPage.GetSelectText(fromPoint, toPoint);
    rects = textPage.GetCharsRectAtPos(fromPoint, toPoint, new Point(10, 10));
}
```

3.2.4 Zooming

ComPDFKit PDF SDK provides super zoom out and in to unlock more zoom levels, you can programmatically interact with it by using the following method.

- Page Fit Mode

  You can set page fit mode in your `CPDFViewer.ChangeFitMode(FitMode newMode)` by using. `CPDFViewer` supports the following page fit modes:

  - `FitMode::FitWidth`: The zoom is set so that the page's width matches the viewer's width.
  - `FitMode::FitHeight`: The zoom is set so that the page's height matches the viewer's height.
  - `FitMode::FitSize`: The zoom is set so that the entire page is visible without scrolling.
  - `FitMode::FitFree`: The viewer's zoom is not adjusted based on the page.
- Manual Zooming

  You can use `CPDFViewer.Zoom(double newZoom)` to zoom the current document after setting page fit mode to `FitMode::FitFree`.

3.2.5 Themes

Theme color refers to rendering pages in the PDF document with a certain background color to enhance the visual effect when reading. For example, set the PDF document background to a solid background.

`CPDFViewer` has four special color modes: dark mode, sepia mode, reseda mode, and custom color mode.

In dark mode, colors are adjusted to improve reading at night or in a poorly-lit environment, in sepia mode, background color is set to emulate the look of an old book, in reseda mode, light-green background is displayed to protect your eyes after long-time reading, and in custom color mode, you can set a custom color for the background color.

Note: *Changing the appearance mode will change the PDF rendering style, but it does not modify the PDF on disk.*

To set the color mode:

1. Find the constant value of the color mode

Themes	Constant value
Normal color mode	`Draw_Mode_Normal`
Night mode	`Draw_Mode_Dark`
Sepia mode	`Draw_Mode_Soft`
Reseda mode	`Draw_Mode_Green`
Custom color mode	`Draw_Mode_Custom`

2. Call `CPDFViewer.SetDrawMode(DrawModes drawMode)`.

3. If you are using `Draw_Mode_Custom`, call `CPDFViewer.SetDrawMode(DrawModes.Draw_Mode_Custom, uint customBgColor)` to set the background color.

3.2.6 Custom Menu

When viewing a PDF, `CPDFViewer` will enter the corresponding interaction state according to the right-click event of mouse. The corresponding menu will be popped up in different interaction states, and you can perform related operations through the menu options.

The following are examples of the commonly used right-click menus of mouse:

```
CPDFViewer pdfViewer = new CPDFViewer();
pdfViewer.InitDocument("filePath");
pdfViewer.SetMouseMode(MouseModes.PanTool);
pdfViewer.AnnotCommandHandler += PdfViewer_AnnotCommandHandler;
private void PdfViewer_AnnotCommandHandler(object sender, AnnotCommandArgs e)
{
    switch (e.CommandType)
    {
        case CommandType.Context:
            e.Handle = true;
            e.PopupMenu = new ContextMenu();
            e.PopupMenu.Items.Add(new MenuItem() { Header = "Copy", Command =
ApplicationCommands.Copy, CommandTarget = (UIElement)sender });
            e.PopupMenu.Items.Add(new MenuItem() { Header = "Cut", Command =
ApplicationCommands.Cut, CommandTarget = (UIElement)sender });
            e.PopupMenu.Items.Add(new MenuItem() { Header = "Paste", Command =
ApplicationCommands.Paste, CommandTarget = (UIElement)sender });
            e.PopupMenu.Items.Add(new MenuItem() { Header = "Delete", Command =
ApplicationCommands.Delete, CommandTarget = (UIElement)sender });
            break;
        case CommandType.Copy:
            e.DoCommand();
            break;
        case CommandType.Cut:
        case CommandType.Paste:
        case CommandType.Delete:
            e.DoCommand();
            break;
        default:
            break;
    }
}
```

3.2.7 Highlight of Form Fields and Hyperlinks

The highlight feature of PDF form fields can help users quickly locate and fill out forms, which greatly improves work efficiency in scenarios where a large number of forms need to be filled out. At the same time, the highlight feature of hyperlink annotations allows users to add hyperlinks and annotations to important information in PDF documents, making it more convenient for other users to quickly find and understand the information, and also improving the readability and interactivity of PDF documents. these features make it easier and more efficient for users to use PDF documents for work and study. These features enable users to use PDF documents more conveniently and efficiently for work and study.

- Use `CPDFViewer.SetFormFieldHighlight(bool isHighlight)` to set whether to highlight form fields.
- Use `CPDFViewer.SetShowLink(bool showLink)` to set whether to highlight hyperlinks.

3.2.8 Get the Selected Text

ComPDFKit PDF SDK provides the API to get the currently selected text `CPDFViewer.GetSelectedText()` from the `CPDFViewer`.

3.3 Annotations

PDF annotations refer to adding, editing, and sharing various multimedia contents on PDF documents, such as note, link, free text, shapes, markup, stamps, ink, sound, etc. By adding annotations, users can more conveniently mark, annotate, revise, comment, sign, and share PDF documents.

PDF annotations are very useful for document communication, review, and revision, and can improve work efficiency and document quality. At the same time, ComPDFKit PDF SDK provides different annotation features and tools, and developers can choose suitable features and tools according to their needs.

3.3.1 Annotation Types

ComPDFKit PDF SDK supports all common annotation types:

Type	Description	Class
Note	Add notes to PDF documents to mark important text, tables, images, and more, making it convenient for you and other readers to read and understand.	`CPDFTextAnnotation`
Link	Add hyperlinks to PDF documents to link to web pages, emails, or specific locations in other documents.	`CPDFLinkAnnotation`
Free Text	Add text and comments to PDF documents.	`CPDFFreetextAnnotation`
Shapes: Square, Circle, and Line	Add graphic elements like shapes, lines, arrows, or images to PDF documents to emphasize or illustrate certain content.	`CPDFSquareAnnotation` `CPDFCircleAnnotation` `CPDFLineAnnotation`
Markup: Highlight, Underline, Strikeout, and Squiggly	Add markups to PDF documents to highlight, emphasize, or illustrate specific content, such as important paragraphs, lines or words, keywords or tables, etc.	`CPDFHighlightAnnotation` `CPDFUnderlineAnnotation` `CPDFStrikeoutAnnotation` `CPDFSquigglyAnnotation`
Stamp	Add stamps to PDF documents to identify and verify the source and authenticity of documents, using the formats of numbers or images to represent signers.	`CPDFStampAnnotation`
Ink	Draw the brush trails freely anywhere in PDF documents, making it easy for users to add doodles, charts, sketches, signatures, or other custom content.	`CPDFInkAnnotation`
Sound	Add sound and other multimedia content to PDF documents to enrich the presentation.	`CPDFSoundAnnotation`

ComPDFKit PDF SDK supports most annotation types defined in PDF Reference and provides APIs for annotation creation, properties access and modification, appearance setting, and drawing. These standard annotations can be read and written by many apps, such as Adobe Acrobat and Apple Preview.

3.3.2 Access Annotations

`CPDFAnnotation` is the base class for all annotations. It has no practical use and cannot instantiate objects, only subclasses such as `CPDFCircleAnnotation` and `CPDFTextAnnotation` are useful. Any unknown or unsupported annotations will be filtered out when parsing a PDF.

To access the list of annotations by using the following method:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
int pageCount = document.PageCount;
if(pageCount>0)
{
    for(int pageIndex = 0;pageIndex < pageCount;pageIndex++)
    {
        List<CPDFAnnotation>annotations =
document.PageAtIndex(pageIndex).GetAnnotations();
        if(annotations != null && annotations.Count != 0)
        {
          foreach(CPDFAnnotation annotation in annotations)
          {
            // do something
          }
        }
    }
}
```

The elements of the array will most likely be typed to subclasses of the `CPDFAnnotation` class.

3.3.3 Create & Edit Annotations

ComPDFKit PDF SDK includes a wide variety of standard annotations, and each of them is added to the project in a similar way.

- Note

  Add a sticky note (text annotation) to a PDF Document page by using the following method.

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
  CPDFPage page = document.PageAtIndex(0);
  CPDFTextAnnotation text = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_TEXT) as
  CPDFTextAnnotation;
  text.SetContent("test");
  text.SetRect(new CRect(0,50,50,0));
  byte[] color = {255,0,0};
  text.SetColor(color);
  text.UpdateAp();
  ```

- Link

  Add a hyperlink or intra-document link annotation to a PDF Document page by using the following method.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

CPDFDestination dest = new CPDFDestination();
CPDFLinkAnnotation link = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_LINK) as
CPDFLinkAnnotation;
link.SetRect(new CRect(0,50,50,0));
link.SetDestination(document,dest);
```

- Free Text

  Add a free text annotation to a PDF Document page by using the following method.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

CPDFFreeTextAnnotation freeText =
page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_FREETEXT) as
CPDFFreeTextAnnotation;
freeText.SetContent("test");
freeText.SetRect(new CRect(0, 50, 50, 0));

CTextAttribute textAttribute = new CTextAttribute();
textAttribute.FontName = "Helvetica";
textAttribute.FontSize = 12;
byte[] fontColor = { 255, 0, 0 };
textAttribute.FontColor = fontColor;
freeText.SetFreetextDa(textAttribute);
freeText.SetFreetextAlignment(C_TEXT_ALIGNMENT.ALIGNMENT_LEFT);
freeText.UpdateAp();
```

- Shapes

  Add a shape annotation to a PDF Document page by using the following method.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);
float[] dashArray = {2,1};
byte[] lineColor = {255,0,0};
byte[] bgColor = {0,255,0};

// Square.
CPDFSquareAnnotation square =
page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_SQUARE)   as CPDFSquareAnnotation;
square.SetRect(new CRect(0,50,50,0));
square.SetLineColor(lineColor);
square.SetBgColor(bgColor);
square.SetTransparency(120);
square.SetLineWidth(1);
square.SetBorderStyle(C_BORDER_STYLE.BS_DASHDED,dashArray);
square.UpdateAp();

// Circle.
CPDFCircleAnnotation circle =
page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_CIRCLE) as CPDFCircleAnnotation;
circle.SetRect(new CRect(0,50,50,0));
circle.SetLineColor(lineColor);
circle.SetBgColor(bgColor);
circle.SetTransparency(120);
circle.SetLineWidth(1);
circle.SetBorderStyle(C_BORDER_STYLE.BS_DASHDED,dashArray);
circle.UpdateAp();

// Line.
CPDFLineAnnotation line = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_LINE) as
CPDFLineAnnotation;
line.SetLinePoints(new CPoint(0,0),new CPoint(50,50));
line.SetLineType(C_LINE_TYPE.LINETYPE_NONE,C_LINE_TYPE.LINETYPE_CLOSEDARROW);
line.SetLineColor(lineColor);
line.SetTransparency(120);
line.SetLineWidth(1);
line.SetBorderStyle(C_BORDER_STYLE.BS_DASHDED,dashArray);
line.UpdateAp();
```

Note: *`CPDFLineAnnotation` properties (`Points`) point is specified in page-space coordinates. Page space is a coordinate system with the origin at the lower-left corner of the current page.*

For each line, users can choose separate styles for the start and the end. The styles are defined by the `C_LINE_TYPE` enumeration.

Name	Description
LINETYPE_UNKNOWN	Non-standard or invalid ending.
LINETYPE_NONE	No special line ending.
LINETYPE_ARROW	Two short lines meeting in an acute angle to form an open arrowhead.
LINETYPE_CLOSEDARROW	Two short lines meeting in an acute angle as in the LINETYPE_ARROW style and connected by a third line to form a triangular closed arrowhead filled with the annotation's interior color.
LINETYPE_SQUARE	A square filled with the annotation's interior color.
LINETYPE_CIRCLE	A circle filled with the annotation's interior color.
LINETYPE_DIAMOND	A diamond shape filled with the annotation's interior color.
LINETYPE_BUTT	A short line at the endpoint perpendicular to the line itself.
LINETYPE_ROPENARROW	Two short lines in the reverse direction from LINETYPE_ARROW.
LINETYPE_RCLOSEDARROW	A triangular closed arrowhead in the reverse direction from LINETYPE_CLOSEDARROW.
LINETYPE_SLASH	A short line at the endpoint approximately 30 degrees clockwise from perpendicular to the line itself.

- Markup

  Add a highlight annotation to a PDF Document page by using the following method, and add other markup annotations in similar way.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

CPDFTextPage textPage = page.GetTextPage();
List<Rect> rectList = textPage.GetCharsRectAtPos(new Point(0,0),new
Point(500,500),new Point(10,10));
List<CRect> cRectList = new List<CRect>();
foreach(var rect in rectList)
{
    cRectList.Add(new CRect((float)rect.Left, (float)rect.Top, (float)rect.Right,
(float)rect.Bottom));
}

CPDFHighlightAnnotation highlight =
page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_HIGHLIGHT) as
CPDFHighlightAnnotation;
byte[]color = {0,255,0};
highlight.SetColor(color);
highlight.SetTransparency(120);
highlight.SetQuardRects(cRectList);
highlight.UpdateAp();
```

Note: *Many annotation types are defined as markup annotations because they are used primarily to mark up PDF documents. Markup annotations may be divided into the following: highlight, underline, strikeout, and squiggly.*

- Stamp

  Add standard, text, and image stamps to a PDF document page by using the following method.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

// Standard.
CPDFStampAnnotation standard =
page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_STAMP) as CPDFStampAnnotation;
standard.SetStandardStamp("Approved");
standard.UpdateAp();

// Text.
CPDFStampAnnotation text = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_STAMP)
as CPDFStampAnnotation; ;
text.SetTextStamp("test", "detail text", C_TEXTSTAMP_SHAPE.TEXTSTAMP_LEFT_TRIANGLE,
C_TEXTSTAMP_COLOR.TEXTSTAMP_RED);
text.UpdateAp();

// Image.
CPDFStampAnnotation image = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_STAMP)
as CPDFStampAnnotation; ;
byte[] imageData = new byte[500 * 500];
image.SetImageStamp(imageData, 500, 500);
image.UpdateAp();
```

- Ink

  Add an ink annotation to a PDF Document page by using the following method.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

CPDFInkAnnotation ink = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_INK) as
CPDFInkAnnotation;
List<List<CPoint>> pointList = new List<List<CPoint>>();
ink.SetInkPath(pointList);
ink.SetInkRect(new CRect(0, 50, 50, 0));
byte[] color = { 0, 255, 0 };
ink.SetInkColor(color);
ink.SetTransparency(120);
ink.SetThickness(4);
ink.UpdateAp();
```

- Sound

  Add a sound annotation to a PDF Document page by using the following method.

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
  CPDFPage page = document.PageAtIndex(0);

  CPDFSoundAnnotation sound = page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_SOUND)
  as CPDFSoundAnnotation;
  sound.SetRect(new CRect(0, 50, 50, 0));
  sound.SetSoundPath("soundFilePath");
  sound.UpdateAp();
  ```

ComPDFKit PDF SDK provides APIs to modify the color appearance of properties in annotations, only the corresponding RGB values will be used, and additional values are required for transparency to take effect. The following APIs will introduce how to set the color values and transparency of related properties.

Set the draw color of note:

```
// Property to get / Method to set the opacity for the note annotation.
public byte Transparency { get; private set; }
public override bool SetTransparency(byte transparency);

// Property to get / Method to set color for the note annotation.
public byte[] Color { get; private set; }
public bool SetColor(byte[] color);
```

Set the stroke color and fill color of free text (`opacity` controls the transparency of both the stroke and fill color of free text):

```
// Property to get / Method to set the opacity for the freetext annotation.
public byte Transparency { get; private set; }
public override bool SetTransparency(byte transparency);

// Property to get / Method to set the line color for the free text annotation.
public byte[] LineColor { get; private set; }
public bool SetLineColor(byte[] lineColor);

// Property to get / Method to set and clear the background color for the free text
annotation.
public byte[] BgColor { get; private set; }
public bool SetBgColor(byte[] bgColor)
public bool ClearBgColor();
```

Set the stroke color and fill color of shapes (square, circle, and line):

```
// Property to get / Method to set the opacity for the shapes annotation.
public byte Transparency { get; private set; }
public override bool SetTransparency(byte transparency);

// Property to get / Method to set the line color for the shapes annotation.
public byte[] LineColor { get; private set; }
public bool SetLineColor(byte[] lineColor);

// Property to get / Method to set and clear the fill color for the shapes annotation.
public byte[] BgColor { get; private set; }
public bool SetBgColor(byte[] bgColor);
public bool ClearBgColor();
```

Set the color of markup (highlight, underline, strikeout, and squiggly):

```
// Property to get / Method to set the opacity for the markup annotation.
public byte Transparency { get; private set; }
public override bool SetTransparency(byte transparency);

// Property to get / Method to set color for the markup annotation.
public byte[] Color { get; private set; }
public bool SetColor(byte[] color);
```

3.3.4 Delete Annotations

The code below shows how to remove an annotation from a document.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

List<CPDFAnnotation>annotList = page.GetAnnotations();
annotList[0].RemoveAnnot();
```

3.3.5 Annotation Appearances

Annotations may contain properties that describe their appearance — such as annotation color or shape. However, these don't guarantee that the annotation will be displayed the same in different PDF viewers. To solve this problem, each annotation can define an appearance stream that should be used for rendering the annotation.

When you modify the annotation properties, you must call the `UpdateAp()` method in the `CPDFAnnotation` class.

```
- bool UpdateAp();
```

3.3.6 Import & Export Annotations

XFDF is an XML-based standard from Adobe XFDF for encoding annotations. An XFDF file will contain a snapshot of a PDF document's annotations and forms. It's compatible with Adobe Acrobat and several other third-party frameworks. ComPDFKit supports both reading and writing XFDF.

- Import from XFDF

  You can import annotations and form fields from an XFDF file to a document like so:

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
  document.ImportAnnotaitonFromXFDFPath("xfdfPath","tempPath");
  ```

- Export to XFDF

  You can export annotations and form fields from a document to an XFDF file like so:

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
  document.ExportAnnotationToXFDFPath("xfdfPath","tempPath");
  ```

3.3.7 Flatten Annotations

Annotation flattening refers to the operation that changes annotations into a static area that is part of the PDF document, just like the other text and images in the document. When flattening an annotation, the annotation is removed from the document, while its visual representation is kept intact. A flattened annotation is visible but is non-editable by your users or by your app.

Annotations in a PDF document can be flattened in the ComPDFKit by saving the document and choosing the Flatten mode.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
document.WriteFlattenToFilePath("savePath");
```

3.4 Forms

A PDF document may contain any number of form fields that allow a user to enter information on a PDF page. An interactive form (sometimes referred to as an AcroForm) is a collection of fields for gathering information interactively from the user. Under the hood, PDF form fields are a type of PDF annotation called widget annotations.

ComPDFKit PDF SDK fully supports reading, filling, and editing PDF forms and provides utility methods to make working with forms simple and efficient.

3.4.1 Supported Form Fields

ComPDFKit PDF SDK supports all form types specified by the PDF specification, including:

Type	Description	Annotation Object
Check Box	Select one or more options.	`CPDFCheckBoxWidget`
Radio Button	Select one option from the predefined options.	`CPDFRadioButtonWidget`
Push Button	Create custom buttons on the PDF document that will perform an action when pressed.	`CPDFPushButtonWidget`
List Box	Select one or more options from the predefined options, similar to the Combo Box.	`CPDFListBoxWidget`
Combo Boxes	Select one option from a drop-down list of available text options.	`CPDFComboBoxWidget`
Text	Enter text content such as name, address, email, etc.	`CPDFTextWidget`
Signatures	Sign a PDF document digitally or electronically.	`CPDFSignatureWidget`

`CPDFWidget` is the base class for all form fields and also a subclass of `CPDFAnnotation`. Its subclasses such as (`CPDFCheckBoxWidget`, `CPDFRadioButtonWidget`, `CPDFPushButtonWidget`, `CPDFListBoxWidget`, `CPDFComboBoxWidget`, `CPDFTextWidget` , `CPDFSignatureWidget`) are the most important. Any unknown or unsupported form fields will be filtered out when parsing a PDF.

3.4.2 Create & Edit Form Fields

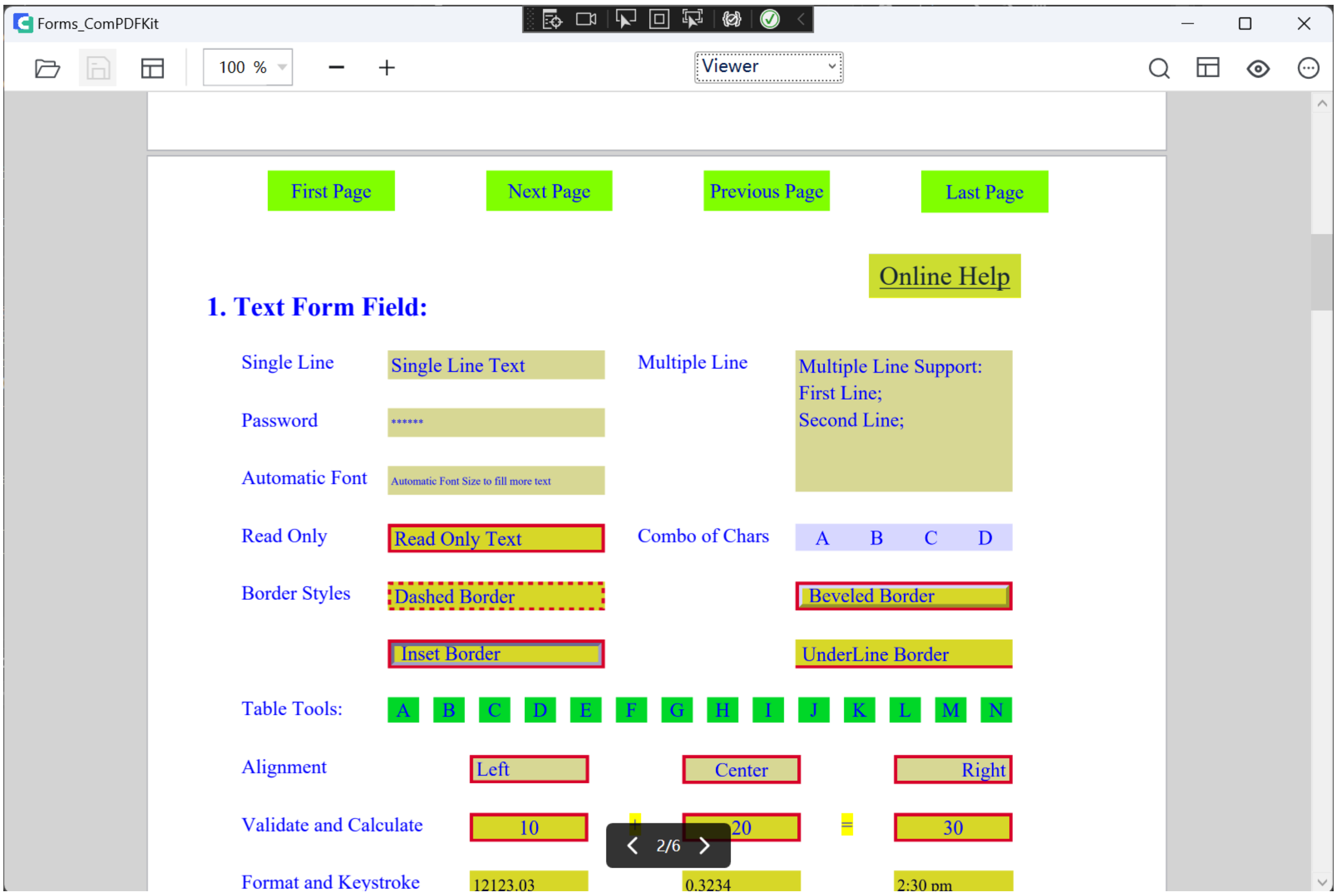
Creating form fields works the same as adding any other annotation, as can be seen in the guides for programmatically creating annotations.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

CPDFListBoxWidget listbox = page.CreateWidget(C_WIDGET_TYPE.WIDGET_LISTBOX) as
CPDFListBoxWidget;
listbox.AddOptionItem(0, "1", "a");
listbox.AddOptionItem(1, "2", "b");
```

3.4.3 Fill Form Fields

ComPDFKit PDF SDK fully supports the AcroForm standard, and forms can be viewed and filled inside the `CPDFView`.

To fill in a text form element, tap it and then type text using either the onscreen keyboard or an attached hardware keyboard. Then tap either the Done button above the keyboard or any blank area on the page to deselect the form element, which will commit the changes.



First Page Next Page Previous Page Last Page

Online Help

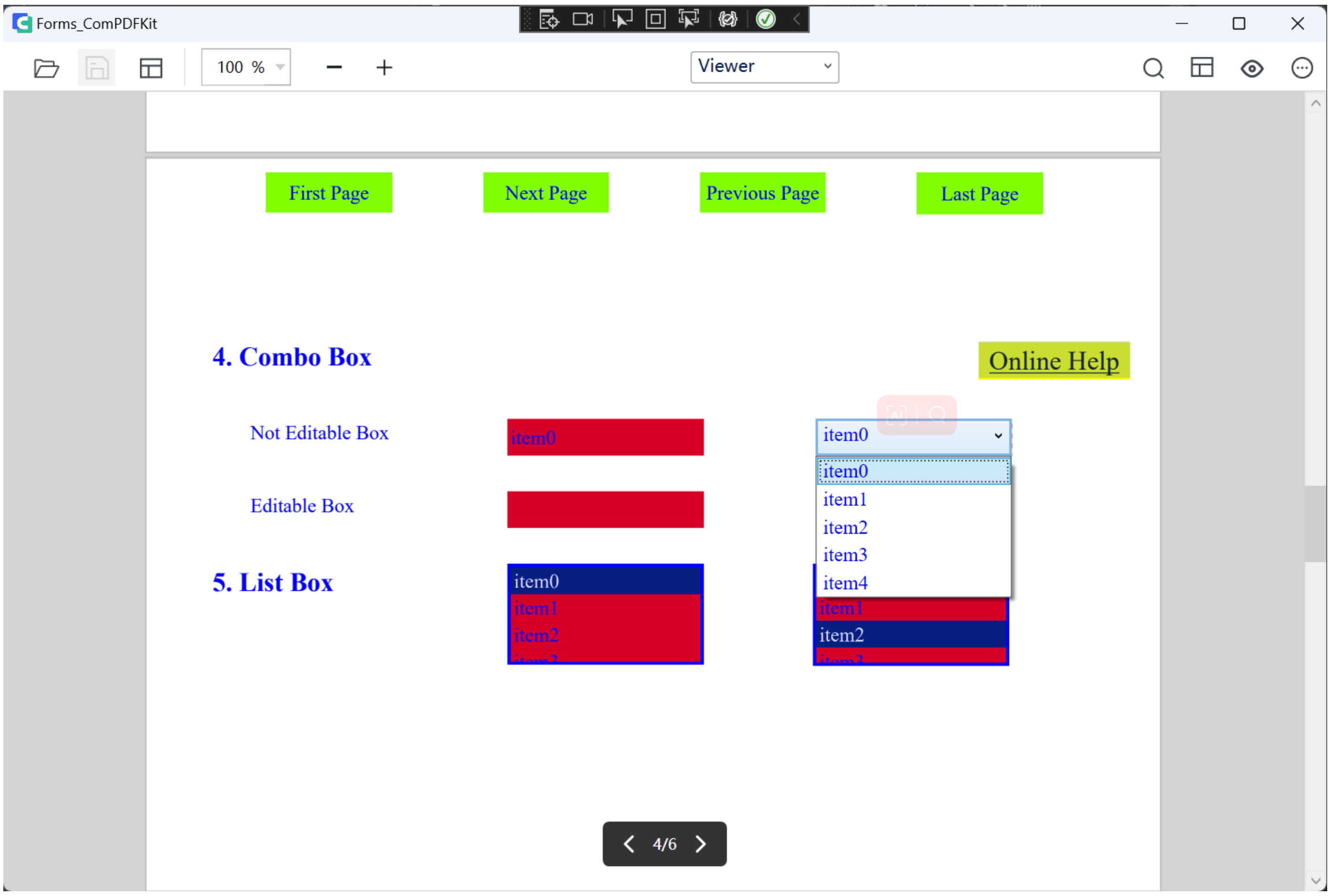
1. Text Form Field:

Single Line	Single Line Text	Multiple Line	Multiple Line Support: First Line; Second Line;
Password	******		
Automatic Font	Automatic Font Size to fill more text		
Read Only	Read Only Text	Combo of Chars	A B C D
Border Styles	Dashed Border		Beveled Border
	Inset Border		UnderLine Border

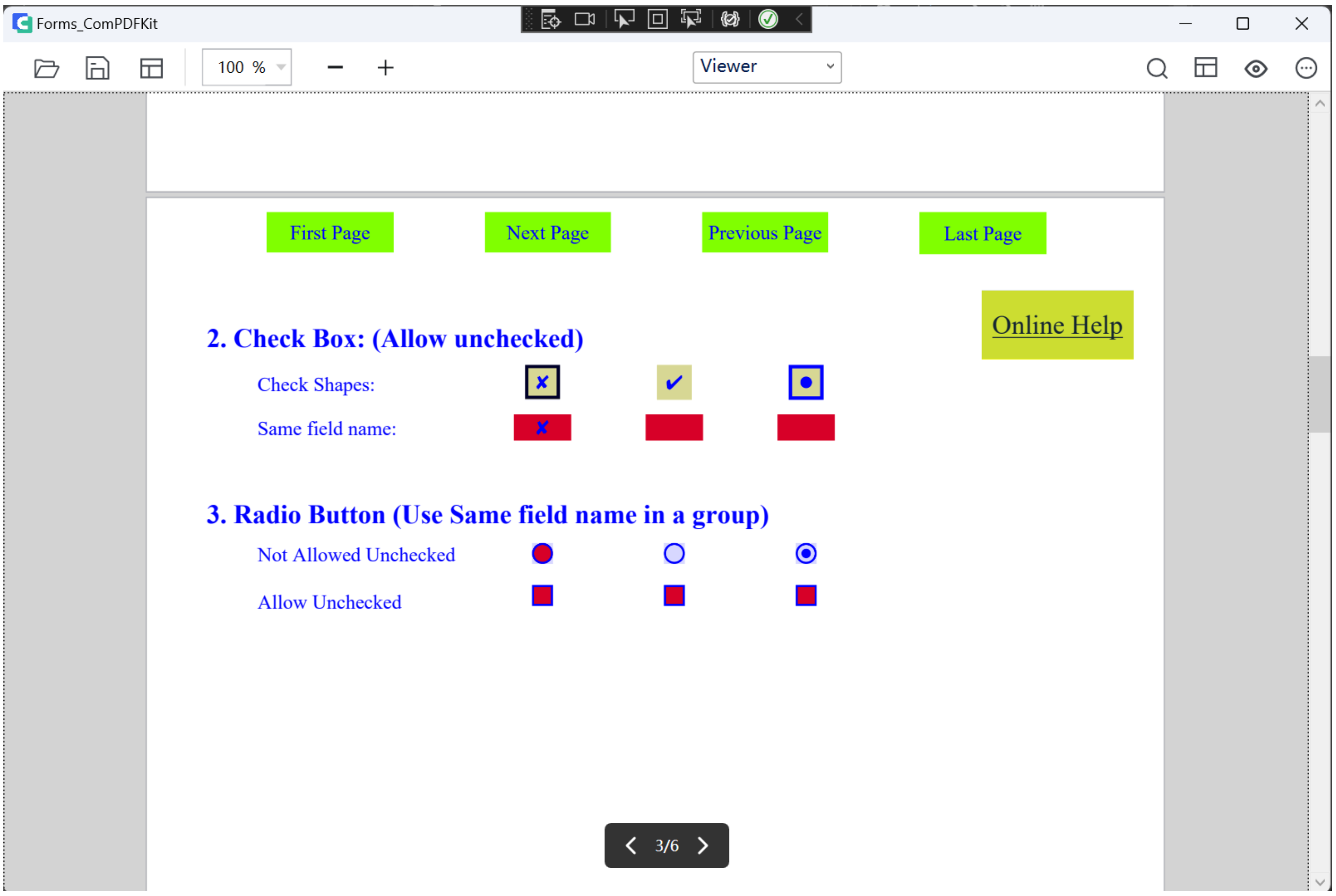
Table Tools: A B C D E F G H I J K L M N

Alignment	Left	Center	Right
Validate and Calculate	10 +	20 =	30
Format and Keystroke	12123.03	0.3234	2:30 pm

To set the value of a choice form element (a list or combo box), tap the element, and then select an item from the list, or type in a custom item.



To enable or disable a checkbox form element, tap it to toggle its state. And you can set the selection of a radio button form element by tapping the desired item.



While a form element is selected (focused), the left and right arrows above the keyboard may be used to move the focus sequentially between all the form elements on the page.

The following example demonstrates how form fields can be queried and filled with code:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

List<CPDFAnnotation> annotList = page.GetAnnotations();
foreach (CPDFAnnotation annot in annotList)
{
    if (annot.Type==C_ANNOTATION_TYPE.C_ANNOTATION_WIDGET)
    {
        CPDFWidget widget = annot as CPDFWidget;
        switch(widget.WidgeType)
        {
            case C_WIDGET_TYPE.WIDGET_TEXTFIELD:
                {
                    CPDFTextWidget text = widget as CPDFTextWidget;
                    text.SetText("test");
                    text.UpdateFormAp();
                }
                break;

            case C_WIDGET_TYPE.WIDGET_RADIOBUTTON:
                {
                    CPDFRadioButtonWidget radio = widget as CPDFRadioButtonWidget;
                    radio.SetChecked(true);
                    radio.UpdateFormAp();
                }
                break;

            case C_WIDGET_TYPE.WIDGET_LISTBOX:
                {
                    CPDFListBoxWidget listBox = widget as CPDFListBoxWidget;
                    listBox.SelectItem(0);
                    listBox.UpdateFormAp();
                }
                break;

            default:
                break;
        }
    }
}
```

3.4.4 Delete Form Fields

Deleting form fields works the same as deleting annotations, and check deleting annotations in the guides to see more.

3.4.5 Flatten PDF Forms

PDF Form flattening works the same as annotation flattening, refer to annotation flattening in the guides to see more.

3.5 Document Editor

ComPDFKit PDF SDK provides a wide range of APIs for document editing operations. These are mostly available through the `CPDFDocument` and `CPDFPage` classes.

ComPDFKit PDF SDK benefits include:

- PDF Manipulation
  - Split pages
  - Merge pages
  - Extract pages
- Page Edit
  - Delete pages
  - Insert pages (choose from another document, a blank page, or an image)
  - Move pages
  - Rotate pages
  - Exchange pages
  - Replace pages
  - Crop pages
- Access Document Information
- Extract Images

3.5.1 PDF Manipulation

- Split Pages

  `CPDFDocument` can extract range of pages from one document and put them into another document. If you run this operation multiple times with different page indexes, you can effectively split a PDF into as many documents as you require.

  To split a PDF document into multiple pages, please use the following method:

  1. Create a blank PDF document.

```
CPDFDocument document = CPDFDocument.CreateDocument();
```

2. Open a PDF document that contains the pages you want to split.

```
CPDFDocument document1 = CPDFDocument.InitWithFilePath("filePath");
```

3. Extract specific pages from the PDF document that you just opened, and import them into the blank PDF document.

```
// Pages that need to be split, e.g. 2 to 5 pages
document.ImportPagesAtIndex(document1,"2-5",0);
```

4. Save the document.

```
// Save path
document.WriteToFilePath("savePath");
```

- Merge Pages

  ComPDFKit PDF SDK allows you to instantiate multiple `CPDFDocument`, and you can use the `CPDFDocument` API to merge multiple PDF files into a single one.

  To merge PDF documents into one file, please use the following method:

  1. Create a blank PDF document.

  ```
  CPDFDocument document = CPDFDocument.CreateDocument();
  ```

  2. Open the PDF documents that contain the pages you want to merge.

  ```
  // File path
  CPDFDocument document1 = CPDFDocument.InitWithFilePath("filePath");

  // File path
  CPDFDocument document2 = CPDFDocument.InitWithFilePath("filePath2");
  ```

  3. Merge all the pages from the documents you just opened, and import them into the blank PDF document.

  ```
  document.ImportPagesAtIndex(document1,"1-10",document.PageCount);
  document.ImportPagesAtIndex(document2,"1-10",document.PageCount);
  ```

  4. Save the document.

  ```
  // Save path
  document.WriteToFilePath("savePath");
  ```

The sample code above allows you to merge all the pages from the two documents. If you're looking to merge or add specific pages from one document to another, you can use `pageRange` of `CPDFDocument.ImportPagesAtIndex(CPDFDocument otherDocument, string pageRange,int pageIndex)` to set specific pages.

- Extract Pages

  `CPDFDocument` can extract range of pages from one document and put them into a blank document. If you run this operation, you can effectively extract a PDF as you require. Refer to split pages for more details.

3.5.2 Page Edit

Page manipulation is the ability to perform changes to pages.

- To delete pages from a PDF document, use the function `CPDFDocument.RemovePages(int[] pageIndexs)`.
- To insert a blank page into a PDF document, use the function `CPDFDocument.InsertPage(int pageIndex, double width, double height, string imagePath="")`.
- To insert an image as an entire page into a PDF document, use the function `CPDFDocument.InsertPage(int pageIndex, double width, double height, string imagePath)`.
- To insert a specific page from one document to another, use the function `CPDFDocument.ImportPagesAtIndex(CPDFDocument otherDocument, string pageRange,int pageIndex)`.
- To move a page to a new location, use the function `CPDFDocument.MovePage(int startIndex, int endIndex)`.
- To exchange the location of two document pages, use the function `CPDFDocument.ExchangePage(int firstIndex, int secondIndex)`.
- To replace original document pages with new pages from a different document, use the function `CPDFDocument.RemovePages(int[] pageIndexs)` and `CPDFDocument.ImportPagesAtIndex(CPDFDocument otherDocument, string pageRange,int pageIndex)`.
- To rotate a page in a PDF document, refer to the following method in the `CPDFPage` class.

```
// Rotation on a page. Must be 0, 1, 2, or 3(negative rotations will be
"normalized" to one of 0, 1, 2, or 3).
// Some PDFs have an inherent rotation and so -[rotation] may be non-zero when a
PDF is first opened.
CPDFPage.RotatePage(int pageIndex, int rotation).
```

3.5.3 Document Information

To access document information, refer to the following method in the `CPDFDocument` class.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFInfo info = document.GetInfo();
string title = info.Title;          //document title.
string author = info.Author;        //document author.
string subject = info.Subject;      //document subject.
string creator = info.Creator;      //name of app that created document.
string producer = info.Producer;    //name of app that produced PDF data.
string keywords = info.Keywords;    //document keywords.
string creationDate = info.CreationDate;  //document creation date.
string modificationDate = info.ModificationDate;    //last document modification date.
```

3.5.4 Extract Images

To extract images from a PDF document, use the function `CPDFDocument.ExtractImage(string pageRange, string filePath)`.

The code below will grab all images from the first page of the given PDF document:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
document.ExtractImage("1-10", "saveFolderPath");
```

3.6 Security

ComPDFKit PDF SDK protects the content of PDF documents from unauthorized access like copying or printing. It offers developers a way to encrypt and decrypt PDFs, add a password, insert a watermark, and more. For controlling document security in ComPDFKit PDF SDK, security handlers perform user authorization and set various permissions over PDF documents.

3.6.1 PDF Permission

A PDF file can have two different password sets, a permissions or owner password and an open or user password.

A PDF user password is used to secure access to PDF documents and requires the correct password to view the content. It is commonly used to protect confidential reports and financial documents. The PDF reader also displays document rights such as copying and printing permissions when a user password is set. It is different from the owner's password which controls full access to the document, including modifying content and adding comments.

A PDF permission password, also called an owner or master password, protects the permissions of a PDF document. It restricts actions such as making changes or comments and allows control over copying, printing, and modification. Permission passwords ensure integrity and allow management of advanced editing and security settings. They protect the user's copyright and differ from user passwords.

Description and permissions description about PDF user password and permissions password:

- When the document does not have a permission password nor a user password, the permission is `PermissionsNone`.
- When there is a permission password without a user password, the permission is `PermissionsNone` before the permission password is entered, and the correct permission is `PermissionsOwner` after input.
- When there is no permission password and there is a user password, the permission is `PermissionsNone` before the password is entered, and the correct permission is `PermissionsUser` after the password is entered.
- When there is a permission password and an open password, the permission is `PermissionsNone` before any user password is entered, the correct permission is `PermissionsOwner` after entering, and the correct open password permission is `PermissionsUser`.
- When the permission password is the same as the user password, the permission becomes `PermissionsOwner` after entering the password.

If you want to open a document with a user password programmatically, you can use the `CPDFDocument.UnlockWithPassword()` API.

The PDF specification defines the permissions are shown below:

- Printing — Print the document.
- High-quality printing — Print the document in high fidelity.
- Copying — Copy content from the document.
- Document changes — Modify the document contents except for document attributes.
- Document assembly — Insert, delete, and rotate pages.
- Commenting — Create or modify document annotations, including form field entries.
- Form field entry — Modify form field entries even if you can't edit document annotations.

To access the corresponding permissions, use the function `CPDFDocument.GetPermissionsInfo()`.

```
/// <summary>
/// A Boolean value indicating whether the document allows printing.
/// </summary>
public bool AllowsPrinting { get; set; }

/// <summary>
/// A Boolean value indicating whether the document allows printing in high fidelity.
/// </summary>
public bool AllowsHighQualityPrinting { get; set; }

/// <summary>
/// A Boolean value indicating whether the document allows copying of content to the
Pasteboard.
```

```
/// </summary>
public bool AllowsCopying { get; set; }

/// <summary>
/// A Boolean value indicating whether you can modify the document contents except for
document attributes.
/// </summary>
public bool AllowsDocumentChanges { get; set; }

/// <summary>
/// A Boolean value indicating whether you can manage a document by inserting,
deleting, and rotating pages.
/// </summary>
public bool AllowsDocumentAssembly { get; set; }

/// <summary>
/// A Boolean value indicating whether you can create or modify document annotations,
including form field entries.
/// </summary>
public bool AllowsCommenting { get; set; }

/// <summary>
/// A Boolean value indicating whether you can modify form field entries even if you
can't edit document annotations.
/// </summary>
public bool AllowsFormFieldEntry { get; set; }
```

- Encrypt

  ComPDFKit's `CPDFDocument` API can generate a password-protected document. You can use `CPDFDocument` to create a new password-protected PDF document on disk based on a current document. The user password prevents users from viewing the PDF. If you specify it, you also need to specify an owner password.

  If the encryption level is set to `CPDFDocumentEncryptionLevelNoEncryptAlgo`, it is equivalent to encrypting with the RC4 algorithm; if the document encryption level obtained is `CPDFDocumentNoEncryptAlgo`, it means that the document is not encrypted.

  Support for 128 and 256 bit AES (Advanced Encryption Standard) encryption.

  ```
  public enum CPDFDocumentEncryptionLevel
  {
      CPDFDocumentRC4,
      CPDFDocumentAES128,
      CPDFDocumentAES256,
      CPDFDocumentNoEncryptAlgo
  }
  ```

For example, you can set the encrypt level to AES 256 and configure a *256-bit* owner password and the "printing" permission when saving a document to make sure that users who don't know that owner password can only print the document, but not modify it.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPermissionsInfo permission = new CPDFPermissionsInfo();
permission.AllowsPrinting = true;
CPDFDocumentEncryptAlgo algorithm = CPDFDocumentEncryptAlgo.CPDFDocumentAES256;
document.Encrypt("UserPassword", "ownerPassword", permission,algorithm);
document.WriteToFilePath("savePath");c
```

- Decrypt

  ComPDFKit PDF SDK fully supports the reading of secured and encrypted PDF documents.

  To check whether a document requires a password:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
if(document != null && document.ErrorType ==
CPDFDocumentError.CPDFDocumentPasswordError)
{
    // Password required
}
```

  To read a PDF document with password protection, use the function `CPDFDocument.UnlockWithPassword(string password)`. If the password is correct, this method returns `true`. Once unlocked, you cannot use this function to relock the document.

  To remove PDF security, call the `CPDFDocument.Decrypt(string filePath)` method:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
document.Decrypt("savePath");
```

3.6.2 Background

The background of a PDF document can be provided with a background image or color that is applied to the entire page of the document. In some cases, the background can use brand-related elements such as a company logo or trademark to enhance the brand image. In addition, another reason for applying a background to a PDF document is to improve readability, especially if the document has little content and the pages appear empty. In these cases, applying some background colors and images can make the page more appealing and easy to read. In addition, highlighting the background of editable areas in a PDF form design can make the form more intuitive to use and increase the efficiency of the user filling out the form. And ensure that the same look and formatting are maintained across multiple platforms and devices.

You can use `CPDFDocument` to get the `CPDFBackground` object, and use the API in `CPDFBackground` to set the background image or color, etc.

The following example shows you how to set the background of the first three pages of a document to a cloth-covered black background:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFBackground background = document.GetBackground();
background.SetBackgroundType(C_Background_Type.BG_TYPE_COLOR);
byte[] color = { 0, 0, 0 };
background.SetColor(color);
background.SetOpacity(255);
background.SetScale(1);
background.SetRotation(0);
background.SetHorizalign(C_Background_Horizalign.BG_HORIZALIGN_CENTER);
background.SetVertalign(C_Background_Vertalign.BG_VERTALIGN_CENTER);
background.SetXOffset(0);
background.SetYOffset(0);
background.SetPages("1-3");
background.Update();
```

The following example shows you how to set the background of the first three pages of a document to a specified image:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFBackground background = document.GetBackground();
background.SetBackgroundType(C_Background_Type.BG_TYPE_IMAGE);

int imageWidth = 100;
int imageHeight = 100;
byte[] image = new byte[imageWidth*imageHeight];
background.SetImage(image,imageWidth,imageHeight,C_Scale_Type.center);
background.SetOpacity(255);
background.SetScale(1);
background.SetRotation(0);
background.SetHorizalign(C_Background_Horizalign.BG_HORIZALIGN_CENTER);
background.SetVertalign(C_Background_Vertalign.BG_VERTALIGN_CENTER);
background.SetXOffset(0);
background.SetYOffset(0);
background.SetPages("1-3");
background.Update();
```

Note: *Adding a background can only be done once. Each call to `CPDFBackground.Update()` will cover the previous background data.*

3.6.3 Page Header and Footer

- To add headers and footers, ComPDFKit provides an API to add customized headers and footers on the top and bottom of each page in a PDF document.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFHeaderFooter headerFooter = document.GetHeaderFooter();
headerFooter.SetText(0, @"<<#3#5#Prefix-#-Suffix>>");
byte[] color = { 255, 0, 0 };
headerFooter.SetTextColor(0, color);
headerFooter.SetFontSize(0, 14);
headerFooter.SetPages("1-3");
headerFooter.Update();
```

- To remove headers and footers, ComPDFKit provides an API to remove existing headers and footers from a page.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFHeaderFooter headerFooter = document.GetHeaderFooter();
headerFooter.Clear();
```

- To manage the position of headers and footers, ComPDFKit provides APIs to help developers change the position and margins of headers and footers to suit the size and layout of the page.

```
// Method to get the margins of the header & footer.
CPDFHeaderFooter.GetMargin();
// Method to set the margins of the header & footer.
CPDFHeaderFooter.SetMargin(float[] margins);
```

- Supporting multiple header and footer types, ComPDFKit provides API support for various types of headers and footers, such as simple text, page index.

`index`: Instructions for adding the position of the header and footer: 0 for top left, 1 for top center, 2 for top right, 3 for bottom left, 4 for bottom center, and 5 for bottom right.

`text`: The regular expressions that Text supports special formats are: <<\d+,\d+>>|<<\d+>>|<<\d+,>>

- <<i>>: 'i' is the starting value of the page number.
- <<i,f>>: 'i' is the starting value of the page number, and 'f' is the number of digits in the page number, if the actual page number is not enough, it will be automatically filled with 0 in front.

eg: When text is set to "<<1,2>> page", the text displayed on the first page is "01 page".

```
// Gets the text of the header & footer at the specified index.
CPDFHeaderFooter.GetText(int index)
// Sets the text of the header & footer at the specified index.
CPDFHeaderFooter.SetText(int index,string text)
```

The following example shows you how to add text as a page index (1 of n) with the start page 5, the font size 14, and the red lower-middle footer.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFHeaderFooter headerFooter = document.GetBates();
headerFooter.SetText(4, @"<<#3#5#Prefix-#-Suffix>>");
byte[] color = { 255, 0, 0 };
headerFooter.SetTextColor(4, color);
headerFooter.SetFontSize(4, 14);
headerFooter.Update();
```

Note: *Adding a header or footer can only be done once. Each call to `CPDFHeaderFooter.Update()` will cover the previous data.*

3.6.4 Bates Number

- To add Bates numbers, ComPDFKit provides an API to add customized Bates numbers on the top and bottom of each page in a PDF document.

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

  CPDFBates bates = document.GetBates();
  bates.SetText(0, @"<<#3#5#Prefix-#-Suffix>>");
  byte[] color = { 255, 0, 0 };
  bates.SetTextColor(0, color);
  bates.SetFontSize(0,14);
  bates.SetPages("0-" + (document.PageCount - 1));
  bates.Update();
  ```

- To remove Bates numbers , ComPDFKit provides an API to remove existing Bates numbers from the page.

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

  CPDFBates bates = document.GetBates();
  bates.Clear();
  ```

- To manage the position of the Bates numbers, ComPDFKit provides an API to help developers change the position and margins of the Bates numbers to suit the size and layout of the page.

```
// Method to get the margins of the bates.
CPDFBates.GetMargin();
// Method to set the margins of the bates.
CPDFBates.SetMargin(float[] margins);
```

- Support multiple types of Bates numbers, ComPDFKit provides API support for a variety of types of Bates numbers, such as simple text, page index.

`index`: Add the position of the Bates numbers: 0 for top left, 1 for top center, 2 for top right, 3 for bottom left, 4 for bottom center, and 5 for bottom right.

`text`: The regular expressions that Text supports special formats are: <<#\d+#\d+#{0,1}[^#]#*{0,1}[^#]*>>

- The first # is followed by the minimum number of digits in the page number. If the number of page digits is not enough, 0 is added in front.
- The second # is followed by the starting value of the page number.
- The third # is followed by the Bates prefix.
- The fourth # is followed by the Bates suffix.

eg: When text is set to "front<<#3#1#ab#cd>>back", the text displayed on the first page is "frontab001cdback".

```
// Get the text of the header & footer at the specified index.
CPDFBates.GetText(int index)
// Set the text of the header & footer at the specified index.
CPDFBates.SetText(int index,string text)
```

The following example shows you how to add text as a Bates number with the prefix "Prefix-", the suffix "-Suffix", the start page of 5, the bit count of 3, the font size of 14, and the red lower middle footer.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFBates bates = document.GetBates();
bates.SetText(4, @"<<#3#5#Prefix-#-Suffix>>");
byte[] color = { 255, 0, 0 };
bates.SetTextColor(4, color);
bates.SetFontSize(4, 14);
bates.Update();
```

Note: *Adding a Bates number can only be done once. Each call to `CPDFBates.Update()` will cover the last Bates number.*

3.7 Redaction

Redaction is the process of removing images, text, and vector graphics from a PDF page. This not only involves obscuring the content, but also removing the data in the document within the specified area.

Redaction typically involves removing sensitive content within documents for safe distribution to courts, patent and government institutions, the media, customers, vendors, or any other audience with restricted access to the content. Redaction is a two-step process.

- First, redaction annotations have to be created in the areas that should be redacted. This step won't remove any content from the document yet; it just marks regions for redaction.
- Second, to actually remove the content, the redaction annotations need to be applied. In this step, the page content within the region of the redaction annotations is irreversibly removed.

This means that the actual removal of content happens only after redaction annotations are applied to the document. Before applying, the redaction annotations can be edited and removed the same as any other annotations.

Redacting PDFs programmatically:

- Creating Redactions Programmatically

  You can create redactions programmatically via `CPDFRedactAnnotation`. Use the `SetQuardRects` or `SetRect` method to set the areas that should be covered by the redaction annotation.

  You also have a few customization options for what a redaction should look like, both in its marked state, which is when the redaction has been created but not yet applied, and in its redacted state, which is when the redaction has been applied. It is impossible to change the appearance once a redaction has been applied since the redaction annotation will be removed from the document in the process of applying the redaction.

  This is how to create a redaction annotation that covers the specified region on the first page of a document:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);
CPDFRedactAnnotation redact =
page.CreateAnnot(C_ANNOTATION_TYPE.C_ANNOTATION_REDACT) as CPDFRedactAnnotation;
redact.SetRect(new CRect(0, 50, 50, 0));
redact.SetOverlayText("REDACTED");

CTextAttribute textDa = new CTextAttribute();
textDa.FontName = "Helvetica";
textDa.FontSize = 12;
byte[] fontColor = { 255, 0, 0 };
textDa.FontColor = fontColor;
redact.SetTextDa(textDa);

redact.SetTextAlignment(C_TEXT_ALIGNMENT.ALIGNMENT_LEFT);
byte[] fillColor = { 255, 0, 0 };
redact.SetFillColor(fillColor);
byte[] outlineColor = { 0, 255, 0 };
redact.SetOutlineColor(outlineColor);

redact.UpdateAp();
```

- Applying Redactions Programmatically

  ```
  document.ApplyRedaction();
  ```

3.8 Watermark

A PDF watermark is an element such as a backward layer of transparent text or image added to a PDF document to maintain confidentiality and copyright protection of the document and to highlight information about the company or group to which it belongs. Watermarks can be text, images, shapes, a fixed logo, or dynamically generated text or images.

Adding a watermark to ComPDFKit PDF SDK is very simple and can be done through the API provided by `CPDFWatermark` to add a text watermark or an image watermark. You can set the watermark position, color, transparency, font, size, and other parameters to achieve different needs.

- To add a watermark, use the function `CPDFDocument.InitWatermark(C_Watermark_Type type)`.
- To remove all the watermarks in a PDF document, use function `CPDFDocument.DeleteWatermarks()`.

How to generate a PDF with a watermark on all its pages using the `CPDFDocument` API:

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");

CPDFWatermark watermark = document.InitWatermark(C_Watermark_Type.WATERMARK_TYPE_TEXT);
watermark.SetText("test");
watermark.SetFontName("Helvetica");
byte[] color = { 255, 0, 0 };
watermark.SetTextRGBColor(color);
watermark.SetScale(2);
watermark.SetRotation(0);
watermark.SetOpacity(120);
watermark.SetVertalign(C_Watermark_Vertalign.WATERMARK_VERTALIGN_CENTER);
watermark.SetHorizalign(C_Watermark_Horizalign.WATERMARK_HORIZALIGN_CENTER);
watermark.SetVertOffset(0);
watermark.SetHorizOffset(0);
watermark.SetFront(true);
watermark.SetFullScreen(false);
watermark.SetVerticalSpacing(10);
watermark.SetHorizontalSpacing(10);
watermark.CreateWatermark();

document.WriteToFilePath("savePath");
```

3.9 Conversion

3.9.1 PDF/A

The conversion option analyzes the content of existing PDF files and performs a sequence of modifications in order to produce a PDF/A compliant document.

Features that are not suitable for long-term archiving (such as encryption, obsolete compression schemes, missing fonts, or device-dependent color) are replaced with their PDF/A compliant equivalents. Because the conversion process applies only necessary changes to the source file, the information loss is minimal.

Converts existing PDF files to PDF/A compliant documents, including PDF/A-1a and PDF/A-1b only.

```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
document.WritePDFAToFilePath(CPDFType.CPDFTypePDFA1a,"savePath");
```

3.10 Content Editor

Content editor provides the ability to change content so that its data can be improved or re-purposed.

- Create, move, or delete text and images.
- Edit text and image properties.
- Undo or redo any change.

When editing content, other operations are not supported like adding or deleting annotations, adding watermarks, modifying form properties, etc.

Content editor supports the following editing modes:

- Text Mode. In text mode, the text blocks surrounded by dotted lines will be displayed in the PDF document, then you can copy, paste, add, or delete text.
- Image Mode. In image mode, the images surrounded by dotted lines will be displayed in the PDF document, then you can delete, crop, rotate, mirror, replace, save images, or set transparency.
- Text-Image Mode. In text-image mode, the text blocks and images surrounded by dotted lines will be displayed in the PDF document, then you can edit text and images.

3.10.1 Initialize the Editing Mode

Before editing, you should initialize editing mode. ComPDFKit provides methods to initialize editing mode. The following code shows you how to initialize the editing mode:

```
CPDFViewer viewer = new CPDFViewer();
viewer.InitDocument("***");
viewer.SetMouseMode(MouseModes.PDFEdit);
viewer.SetPDFEditType(CPDFEditType.EditText|CPDFEditType.EditImage);
```

3.10.2 Create, Move, or Delete Text and Images

ComPDFKit provides methods to do various operations like creating text/images.

You can use the mouse and keyboard to manipulate text or image areas on `CPDFViewer` as in Microsoft Word, if you want to copy, paste, cut, or delete text or images, you can use the `CPDFViewer`'s `PDFEditCommandHandler` event. The following code will show you how to do this.

```
viewer.PDFEditCommandHandler += Viewer_PDFEditCommandHandler;
private void Viewer_PDFEditCommandHandler(object sender, TextEditCommand e)
{
    e.Handle = true;
    e.PopupMenu = new ContextMenu();
    e.PopupMenu.Items.Add(new MenuItem() { Header = "Copy", Command =
ApplicationCommands.Copy, CommandTarget = (UIElement)sender });
    e.PopupMenu.Items.Add(new MenuItem() { Header = "Cut", Command =
ApplicationCommands.Cut, CommandTarget = (UIElement)sender });
    e.PopupMenu.Items.Add(new MenuItem() { Header = "Paste", Command =
ApplicationCommands.Paste, CommandTarget = (UIElement)sender });
    e.PopupMenu.Items.Add(new MenuItem() { Header = "Delete", Command =
ApplicationCommands.Delete, CommandTarget = (UIElement)sender });
    e.PopupMenu.Items.Add(new MenuItem() { Header = "Select All", Command =
ApplicationCommands.SelectAll, CommandTarget = (UIElement)sender });
}
```

You can use `CPDFViewer`'s `SetPDFEditCreateType` method to insert the text and image. The following code will show you how to do this.

```
//Insert text.
viewer.SetPDFEditCreateType(CPDFEditType.EditText);

//Insert image.
viewer.SetPDFEditCreateType(CPDFEditType.EditImage);
```

3.10.3 Edit Text and Images Properties

ComPDFKit provides multiple methods to modify text properties. You can modify text font size, name, color, alignment, italic, bold, transparency, etc.

You can use `CPDFViewer`'s `PDFEditActiveHandler` event to set the text and image properties. The following code shows you how to set text to 12pt, red, and bold.

```
viewer.PDFEditActiveHandler += Viewer_PDFEditActiveHandler;
private void Viewer_PDFEditActiveHandler(object sender, PDFEditEvent e)
{
    //Text properties.
    if (e.EditType == CPDFEditType.EditText)
    {
        e.FontColor = Colors.Red;
        e.FontSize = 12;
        e.TextAlign = TextAlignType.AlignJustify;
        e.FontWeight = FontWeights.Bold;
        e.FontStyle = FontStyles.Italic;
        e.FontFamily = new FontFamily("TimesNewRoman");
        e.Transparency = 255;
        e.UpdatePDFEditByEventArgs();
    }
}
```

ComPDFKit provides multiple methods to modify image properties. You can modify image properties, such as rotating, cropping, mirroring, and setting transparency.

The following code shows you how to rotate an image and set it to translucent.

```
viewer.PDFEditActiveHandler += Viewer_PDFEditActiveHandler;
private void Viewer_PDFEditActiveHandler(object sender, PDFEditEvent e)
{
    //Image properties.
    if (e.EditType == CPDFEditType.EditImage)
    {
        e.VerticalMirror = true;
        e.HorizontalMirror = true;
        e.ClipImage = true;
        e.Rotate = 90;
        e.ReplaceImagePath = "***";
        e.Transparency = 255;
        e.UpdatePDFEditByEventArgs();
    }
}
```

3.10.4 Undo and Redo Text Editing

You can use `CPDFViewer`'s `UndoManager` class to undo and redo when editing text. The following code will show you how to do this.

```
//Undo
if (viewer.UndoManager.CanUndo)
{
    viewer.UndoManager.Undo();
}
//Redo
if (viewer.UndoManager.CanRedo)
{
    viewer.UndoManager.Redo();
}
```

3.10.5 End Text Editing and Save

You can end the editing mode at any time. You can use `CPDFViewer`'s `SetMouseMode` to exit the editing mode. To save the changes of editing text, use `WriteToLoadedPath` or `WriteToFilePath`.

```
//Exit the editing mode to PanTool mode
viewer.SetMouseMode(MouseModes.PanTool);

if(viewer.UndoManager.CanSave)
{
  //save changes to local file
  viewer.Document.WriteToLoadedPath();
}
```

3.11 Compare Documents

ComPDFKit provides two methods to compare documents:

- Overlay Comparison
- Content Comparison

3.11.1 Overlay Comparison

Overlay Comparison is used to visually compare pages of different documents. It's helpful for things such as construction plans and detailed drawings, as well as other content that requires precise placement.

This can be done in ComPDFKit using `CPDFCompareOverlay`. The process of preparing documents and comparing them involves a few steps to specify how the comparison should happen.

- Generating the Comparison Document

You can use `CPDFCompareOverlay` to generate a comparison document. First, initialize it with the two versions of a document to be compared, then call the `Compare` function first and the `ComparisonDocument` function next as arguments:

```
CPDFDocument oldDocument = CPDFDocument.InitWithFilePath("***");
CPDFDocument newDocument = CPDFDocument.InitWithFilePath("***");
CPDFCompareOverlay compareOverlay = new
CPDFCompareOverlay(oldDocument,newDocument);
compareOverlay.Compare();
CPDFDocument comparisonDocument = compareOverlay.ComparisonDocument();
```

By default, the `CPDFCompareOverlay` will generate a comparison document according to the order of pages, starting from the first page of both versions of the document. If the page you want to compare has moved, or if it's not the first page, you can specify explicit indices using extra `pageRange` arguments:

```
CPDFDocument oldDocument = CPDFDocument.InitWithFilePath("***");
CPDFDocument newDocument = CPDFDocument.InitWithFilePath("***");
CPDFCompareOverlay compareOverlay = new CPDFCompareOverlay(oldDocument,"1-
5",newDocument,"2-6");
compareOverlay.Compare();
CPDFDocument comparisonDocument = compareOverlay.ComparisonDocument();
```

- Changing the Stroke Colors

  One of the most important steps of generating a comparison document is the ability to change the stroke colors, which makes it easier to see the differences between two versions of a document.

  Setting a different stroke color is usually necessary when trying to compare documents, as this will enable you to make any differences between pages more obvious. This will only affect stroke objects in the PDF, and it will leave the color of other elements, such as text or images, unchanged.

  The stroke colors of both versions of a document can be changed using the `SetOldDocumentStrokeColor` and `SetNewDocumentStrokeColor` properties.

  You can also change the blend mode used to overlay the new version of a document on top of the old one by changing the `SetBlendMode` property.

  Trying out various stroke colors and blend modes will result in different-looking comparison documents, and you can make sure the final result fits your needs.

3.11.2 Content Comparison

Quickly pinpoint changes by comparing two versions of a PDF file.

This can be done in ComPDFKit using `CPDFCompareContent`. The process of preparing documents and comparing them involves a few steps to specify how the comparison should happen.

- Generating the Comparison Results

You can use `CPDFCompareContent` to generate the comparison results. First, initialize it with the two versions of a document to be compared, and then call the `Compare` function as an argument:

```
CPDFDocument oldDocument = CPDFDocument.InitWithFilePath("***");
CPDFDocument newDocument = CPDFDocument.InitWithFilePath("***");
CPDFCompareContent compareContent = new CPDFCompareContent(oldDocument,
newDocument);
int pageCount = Math.Min(oldDocument.PageCount,newDocument.PageCount);

for(int i=0;i<pageCount-1;i++)
{
    CPDFCompareResults compareResults =
compareContent.Compare(i,i,CPDFCompareType.CPDFCompareTypeAll,true);
}
```

You can compare the different content types of the document by setting `type`. For example, using `CPDFCompareTypeText` to compare text only or using `CPDFCompareTypeAll` to compare all content.

- Changing the Highlight Colors

  One of the most important steps in generating the comparison results is the ability to change the highlight colors, which makes it easier to see the differences between two versions of a document.

  The highlight colors of both versions of a document can be changed using the `SetReplaceColor`, `SetInsertColor`, and `SetDeleteColor` properties.

3.12 Digital Signatures

3.12.1 Overview

The Concept of Digital Signatures

A digital signature is legally binding and can be equivalent to an ink pen signature on paper contracts and other documents.

Unlike electronic signatures, digital signatures have a unique digital ID that identifies the signer's identity. The function of digital signatures is to obtain information about whether the signature is trustworthy and whether the document has been modified after the signature, thereby ensuring the legal validity of the document.

Advantages of ComPDFKit Digital Signatures

- Authentication:

  Digital signatures can accurately identify the creator and the signer of a document.

- Integrity:

  Digital signatures allow users to easily verify whether the document's content has been altered after signing.

- Non-Repudiation:

When the signature is valid, it can prove the signer's intent to sign, they can't deny that they have signed the document.

- Built-in Certificate Support:

  Full support for PFX and P12 certificates.

- Custom Appearance:

  Customize the appearance of signatures through drawn, image, or typed signatures.

- Default UI:

  Achieve quick integration and customization using the extensible UI components provided by the ComPDFKit team.

Supported Features of ComPDFKit Digital Signatures

- Certificate Creation:

  Create certificates in PFX or P12 formats, which can be used for digital signatures.

- Signature Creation:

  This function allows users to generate a digital signature using a digital certificate with a personal private key, and attach it to a specific document to ensure data integrity and origin verification.

- Signature Information Verification:

  By verifying signature information, users can determine whether specific data or documents have been authorized and remain unaltered.

- Certificate Information Verification:

  Certificate information verification allows users to confirm the validity and authenticity of the digital certificates.

- Extracting Signature Information:

  Extracting signature information refers to extracting the information of a digital signature so that other users can view or archive this information.

- Trusting Certificates:

  Trusting a certificate refers to the act of considering a specific certificate or a certificate authority as trustworthy. This is a key part of the digital signature system as it ensures the trustworthiness of the signature and the certificate, thereby building a secure digital communication and interaction environment.

- Signature Deletion:

  Signature deletion refers to the revocation or invalidation of a digital signature. This may occur due to the loss or compromise of the signer's private key or when a signature is considered no longer valid. Signature deletion is a part of digital security management to ensure data integrity and security.

3.12.2 Concepts Related to Digital Signatures

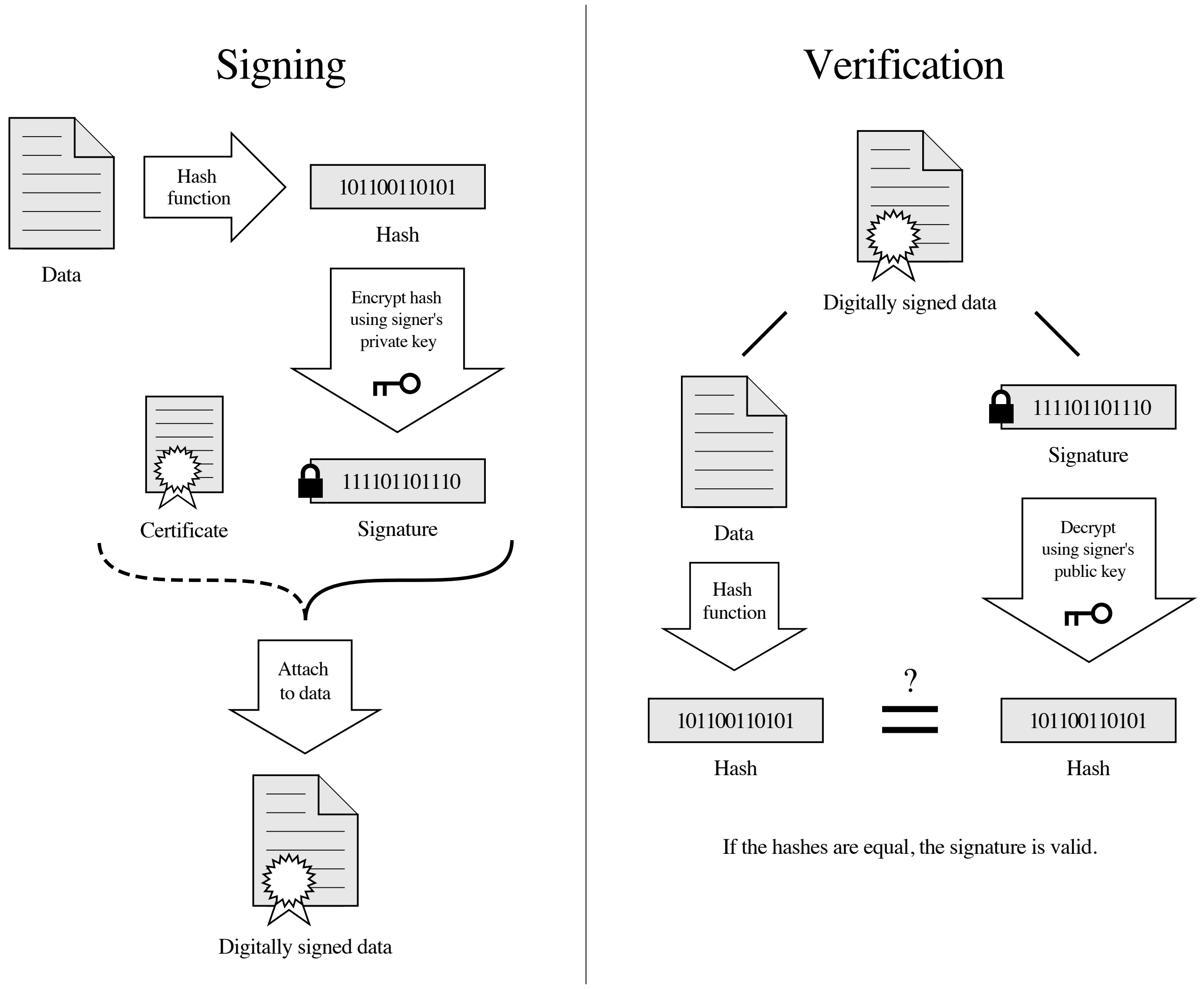
How Digital Signatures Work

Principle of Signature:

A hash value of the data to be encrypted is obtained through a hash function (a unique fingerprint of the data. Any tampering with the data content will result in a different hash). The hash value is encrypted using the signer's private key to obtain the digital signature. The signed data will be generated after attaching the digital signature to the data.

Verification Principle:

Separate the signature from the data, and obtain the hash value of the data through the same hash function used by the signer. Decrypt the hash value using the signer's public key to get the signer's hash value. By comparing the two values, we can confirm whether the file has been tampered with.



Digital Signatures vs Electronic Signatures

An electronic signature is essentially an annotation within a document. Apart from the customizable appearance of the signature, it lacks identifiable information about the creator and cannot verify whether the document has been altered.

However, a digital signature uses complex encryption algorithms to create a unique identifier that is linked to both the document's content and the creator's information. Any modification to the document's content results in a failed digital signature verification, ensuring the uniqueness and legitimacy of the signer's identity.

What Is a Digital Certificate

A digital certificate is a digital authentication that marks the identity information of the parties in Internet communication. It can be used online to identify the identity of the other party, hence, it is also known as a digital ID. The format of the digital certificate typically adopts the X.509 international standard and will generally include the certificate's public key, user information, the validity period of the public key, the name of the certificate authority, the serial number of the digital certificate, and the digital signature of the issuing organization.

Digital certificates provide the transmission of information and data in an encrypted or decrypted form during communication between network users, ensuring the integrity and security of information and data.

Support PKCS12 Certificate

PKCS12 (Or PKCS #12) is one of the family of standards called Public-Key Cryptography Standards (PKCS) published by RSA Laboratories. ComPDFKit supports signing PDFs with PKCS12 files which are with ".p12" or ".pfx" file extensions.

What Is Certificate Chain

A Certificate Chain (Chain of Trust), is an ordered collection of digital certificates used to verify the authenticity and trustworthiness of a digital certificate. Certificate chains are typically employed to establish trust, ensuring that both the public key and the identity of entities are legitimate and trustworthy.

Here are some key concepts within a certificate chain:

- **Root Certificate**

  The starting point of a certificate chain is the Root Certificate. Root certificates are top-level certificates issued by trusted Certificate Authorities (CAs) and are often built into operating systems or applications. These root certificates serve as the foundation of trust because they are considered inherently trustworthy.

- **Intermediate Certificates**

  Intermediate certificates, also known as issuer certificates or sub-certificates, are issued by root certificate authorities and are used to issue certificates for end entities. Intermediate certificates form an intermediate link within the certificate chain.

- **End Entity Certificate**

  An end entity certificate is the certificate of the subject of a digital signature (typically an individual, server, or device). These certificates are issued by intermediate certificate authorities and contain the public key and relevant identity information.

- **Trust Establishment**

  Trust is established through the certificate chain, passing trust from the root certificate to the end entity certificate. If the root certificate is trusted, then the end entity certificate is also trusted, as the trust chain between them is continuous.

Certificate Authority (CA)

A digital certificate issuing authority is an authoritative body responsible for issuing and managing digital certificates, and as a trusted third party in e-commerce transactions, it bears the responsibility for verifying the legality of public keys in the public key system.

The CA center issues a digital certificate to each user who uses a public key, the function of the digital certificate is to prove that the user listed in the certificate legally owns the public key listed in the certificate. The CA is responsible for issuing, certifying, and managing issued certificates. It needs to formulate policies and specific steps to verify and identify user identities and sign user certificates to ensure the identity of the certificate holder and the ownership of the public key.

Whether a Digital Signature Needs a CA

It is not necessary. When there is not a third-party notary, a CA is not needed, and we can use a self-signed certificate. With ComPDFKit, you can manually set to trust self-signed certificates, which is very useful for trusted parties to sign and check files. However, since there is no digital certificate issuing authority for certification, self-signed digital identity cards cannot guarantee the validity of identity information, and they may not be accepted in some use cases.

How to Confirm the Identity of the Digital Certificate Creator

Subject contains identity information about the certificate holder, commonly including fields such as C (Country), ST (Province), L (Locality), O (Organization), OU (Organizational Unit), CN (Common Name), and others. These details help identify who the certificate holder is.
DN (Distinguished Name) represents the complete and hierarchical representation of the "Subject" field. It includes all the information from the "Subject" field and organizes it in a structured manner.

The X.509 standard specifies a specific string format for describing DN, for example:

```
CN=Alan, OU=RD Department, O=ComPDFKit, C=SG, Email=xxxxxx@example.com
```

3.12.3 Create Digital Certificates

PKCS12 (Public Key Cryptography Standard #12) format digital certificates usually contain a public key, a private key, and other information related to the certificate. PKCS12 is a standard format used to store security certificates, private keys, and other related information. This format is commonly used to export, backup, and share digital certificates and private keys which are used in secure communications and identity verification.

When creating a PKCS12 standard certificate, in addition to the data confirming your identity, a password is typically required to protect your certificate. Only those who possess the password can access the private key contained within and perform actions such as signing documents through the certificate.

Key Code

```
// Generate certificate.
//
// Password: ComPDFKit
```

```
//
// info: /C=SG/O=ComPDFKit/D=R&D Department/CN=Alan/emailAddress=xxxx@example.com
//
// C=SG: This represents the country code "SG," which typically stands for Singapore.
// O=ComPDFKit: This is the Organization (O) field, indicating the name of the
organization or entity, in this case, "ComPDFKit."
// D=R&D Department: This is the Department (D) field, indicating the specific
department within the organization, in this case, "R&D Department."
// CN=Alan: This is the Common Name (CN) field, which usually represents the name of
the individual or entity. In this case, it is "Alan."
// emailAddress=xxxx@example.com: Email is xxxx@example.com
//
// CPDFCertUsage.CPDFCertUsageAll: Used for both digital signing and data validation
simultaneously
//
// is_2048 = true: Enhanced security encryption
//
string password = "ComPDFKit";
string info = "/C=SG/O=ComPDFKit/D=R&D
Department/CN=Alan/emailAddress=xxxx@example.com";
string filePath = outputPath + "\\Certificate.pfx";
CPDFPKCS12CertHelper.GeneratePKCS12Cert(info, password, filePath,
CPDFCertUsage.CPDFCertUsageAll);
```

3.12.4 Create Digital Signatures

Creating a digital signature involves two steps:

- Create a Signature Field
- Sign within the Signature Field

By following these two steps, you can either self-sign a document or invite others to sign within the signature field you've created.

ComPDFKit offers support for customizing the styles of the signature form field and allows you to customize the appearance of your signature using drawn, image, and typed signatures.

Key Code for Creating a Signature Field

```
// Create a Signature Field.
//
// Page Index: 0
// Rect: CRect (28, 420, 150, 370)
// Border RGB: { 0, 0, 0 }
// Widget Background RGB: { 150, 180, 210 }
//
CPDFPage page = document.PageAtIndex(0);
            CPDFSignatureWidget signatureField =
page.CreateWidget(C_WIDGET_TYPE.WIDGET_SIGNATUREFIELDS) as CPDFSignatureWidget;
            signatureField.SetRect(new CRect(28, 420, 150, 370));
            signatureField.SetWidgetBorderRGBColor(new byte[] { 0, 0, 0 });
            signatureField.SetWidgetBgRGBColor(new byte[] { 150, 180, 210 });
            signatureField.UpdateAp();
```

Sign Within the Signature Field

To sign within the signature field, you need to do three things:

- Possess a certificate that conforms to the PKCS12 standard (in PFX or P12 format) and ensure that you know its password. You can create a compliant digital certificate using the built-in methods within the ComPDFKit SDK.
- Set the appearance of the digital signature.
- Write the data into the signature field.

Key Code for Signing Within the Signature Field

```
// Sign in the signature field.
//
// Text: Grantor Name
// Content:
// Name: Get the grantor's name from the certificate
// Date: Now (yyyy.mm.dd)
// Reason: I am the owner of the document
// DN: Subject
// IsContentAlginLeft: False
// IsDrawLogo: True
// LogoBitmap: logo.png
// text color RGB: { 0, 0, 0 }
// Output file name: document.FileName + "_Signed.pdf"
//
string name = GetGrantorFromDictionary(certificate.SubjectDict) + "\n";
string date = DateTime.Now.ToString("yyyy.MM.dd HH:mm:ss");
string reason = "I am the owner of the document.";
string location = certificate.SubjectDict["C"];
string DN = certificate.Subject;
CPDFSignatureConfig signatureConfig = new CPDFSignatureConfig
            {
                Text = GetGrantorFromDictionary(ce string filePath = outputPath + "\\"
+ document.FileName + "_Signed.pdf";

 signatureField.UpdataApWithSignature(signatureConfig);rtificate.SubjectDict),
                Content =
                "Name: " + name + "\n" +
                "Date: " + date + "\n" +
                "Reason: "+ reason +" \n" +
                "Location: "+ location + "\n" +
                "DN: " + DN + "\n",
                IsContentAlginLeft = false,
                IsDrawLogo = true,
                LogoBitmap = new Bitmap("Logo.png"),
                textColor = new float[] { 0, 0, 0 },
                contentColor = new float[] { 0, 0, 0 }
            };
string filePath = outputPath + "\\" + document.FileName + "_Signed.pdf";
            signatureField.UpdataApWithSignature(signatureConfig);
document.WriteSignatureToFilePath(signatureField,
                filePath ,
                certificatePath, password,
                location,
                reason, CPDFSignaturePermissions.CPDFSignaturePermissionsNone);
```

3.12.5 Read Digital Signature Information

You can read various pieces of information from a document's digital signature, including the signature itself, the signer of the signature, and certain details of the signer's digital certificate.

For a comprehensive list of retrievable information, please refer to the API Reference.

Key Code

```
foreach (var signature in document.GetSignatureList())
{
    signature.VerifySignatureWithDocument(document);
    Console.WriteLine("Name: " + signature.Name);
    Console.WriteLine("Location: " + signature.Location);
    Console.WriteLine("Reason: " + signature.Reason);
    foreach (var signer in signature.SignerList)
    {
        Console.WriteLine("Date: " + signer.AuthenDate);
        foreach (var certificate in signer.CertificateList)
        {
            Console.WriteLine("Subject: " + certificate.Subject);
        }
    }
}
```

The Connection Between Digital Signatures, Signers, and Digital Certificates

A digital signature is generated by encrypting a document using the private key of the signer and then verifying the validity of the signature using the public key from the signer's certificate. The signature, signer, and digital certificate constitute a crucial part of digital signatures in a PDF document.

In most cases, one signature corresponds to one signer. However, in some situations, a digital signature can include multiple signers, each with their own certificate chain. This multi-signer mechanism can be very useful in certain application scenarios because it allows multiple entities to digitally sign the same document, each using their certificate and private key.

3.12.6 Verify Digital Certificates

When verifying digital certificates, the system automatically checks the trustworthiness of all certificates in the certificate chain and also verifies whether the certificates have expired. Only certificates that are both not expired and considered trustworthy in the entire certificate chain are considered trusted digital certificates.

Key Code

```
// Verify certificate.
//
// To verify the trustworthiness of a certificate, you need to verify that all
certificates in the certificate chain are trustworthy.
// In ComPDFKit, this progress is automatic.
// You should call the "CPDFSignatureCertificate.CheckCertificateIsTrusted" first. Then
you can view the "CPDFSignatureCertificate.IsTrusted" property.
//
CPDFSignatureCertificate certificate =
CPDFPKCS12CertHelper.GetCertificateWithPKCS12Path(certificatePath, password);
certificate.CheckCertificateIsTrusted();
if (certificate.IsTrusted)
{
  // Certificate is trusted.
}
else
{
    // Certificate is not trusted.
}
```

3.12.7 Verify Digital Signatures

Verifying a digital signature consists of signature validity and certificate trustworthiness.

- Signature validity indicates that the document has not been tampered with.
- Certificate trustworthiness confirms that the signer is trustworthy.

Generally, a signature is verified only when both the signature is valid and the certificate is trustworthy.

Key Code

```
foreach (var signature in document.GetSignatureList())
{
    signature.VerifySignatureWithDocument(document);
    foreach (var signer in signature.SignerList)
    {
        Console.WriteLine("Is the certificate trusted: " +
signer.IsCertTrusted.ToString());
        Console.WriteLine("Is the signature verified: " +
signer.IsSignVerified.ToString());
        // Take appropriate actions based on the verification results.
        if (signer.IsCertTrusted && signer.IsSignVerified)
        {
            // Signature is valid and the certificate is trusted.
            // Perform the corresponding actions.
         }
         else if (!signer.IsCertTrusted && signer.IsSignVerified)
         {
              // Signature is valid but the certificate is not trusted.
              // Perform the corresponding actions.
         }
         else
         {
               // Signature is invalid.
               // Perform the corresponding action.
          }
      }
}
```

3.12.8 Trust Certificate

Trusting certificates involve two steps:

- Specify the trust path (folder) for certificates. This path serves as the location where certificates are placed when they are trusted. Additionally, when checking the trustworthiness of certificates, the SDK will look for the corresponding certificates within this folder. Please ensure that this path is valid. If the path does not exist or is inaccessible, the ComPDFKit SDK will not automatically create the trust path folder.

- Execute the method to trust certificates, and the certificates will be added to the trust path.

Key Code

```
CPDFSignature signature = document.GetSignatureList()[0];
            CPDFSignatureCertificate signatureCertificate =
signature.SignerList[0].CertificateList[0];
 Console.WriteLine("Certificate trusted status: " +
signatureCertificate.IsTrusted.ToString());
Console.WriteLine("---Begin trusted---");
string trustedFolder = AppDomain.CurrentDomain.BaseDirectory + @"\TrustedFolder\";
if (!Directory.Exists(trustedFolder))
{
    Directory.CreateDirectory(trustedFolder);
}
// Set your trust path as a folder path.
CPDFSignature.SignCertTrustedFolder = trustedFolder;
// Add your certificate to the trust path.
signatureCertificate.AddToTrustedCertificates();
Console.WriteLine("Certificate trusted status: " +
signatureCertificate.IsTrusted.ToString());
```

3.12.9 Remove Digital Signatures

You can easily remove a digital signature, and when you do so, both the appearance and data associated with the signature will be deleted.

It's important to note that removing a signature does not remove the signature field.

Key Code

```
// Remove digital signature.
// You can choose if you want to remove the appearance.
CPDFSignature signature = document.GetSignatureList()[0];
document.RemoveSignature(signature, true);
string filePath = outputPath + "\\" + document.FileName + "_RemovedSign.pdf";
document.WriteToFilePath(filePath);
```

3.11.10 Trouble Shooting

Inaccurate Signature Information Retrieval

Before retrieving signature information, it is necessary to call the `VerifySignatureWithDocument` method within the `CPDFSignature` class. This method refreshes the document's integrity and checks the validity of the certificate. Failure to call this method before retrieving the signature information may result in obtaining outdated or incorrect results.

Failure to Add Certificate to Trust Path

Before calling the `AddToTrustedCertificates` method within the `CPDFSignatureCertificate` class, you must first set the value of the `SignCertTrustedFolder` parameter in the `AddToTrustedCertificates` class to your trust path folder. Failure to do so will result in the addition process failing. When you trust a certificate, it is added to the specified path, and the SDK also checks for the existence of certificates in that path to determine their trustworthiness. The trust path folder is not automatically created by the SDK, so you should ensure it exists and is a valid path.

4 Support

4.1 Reporting Problems

Thank you for your interest in ComPDFKit PDF SDK, an easy-to-use but powerful development solution to integrate high quality PDF rendering capabilities to your applications. If you encounter any technical questions or bug issues when using ComPDFKit PDF SDK for Windows, please submit the problem report to the ComPDFKit team. More information as follows would help us to solve your problem:

- ComPDFKit PDF SDK product and version.
- Your operating system and IDE version.
- Detailed descriptions of the problem.
- Any other related information, such as an error screenshot.

4.2 Contact Information

Website:

- Home Page: https://www.compdf.com
- API Page: https://api.compdf.com/
- Developer Guides: https://www.compdf.com/guides/pdf-sdk/windows/overview
- API Reference: https://www.compdf.com/guides/pdf-sdk/windows/api-reference/html/3a1f08b6-6ac4-f8b5-bad1-a31c98e96105.htm
- Code Examples: https://www.compdf.com/guides/pdf-sdk/windows/examples

Contact ComPDFKit:

- Contact Sales: https://api.compdf.com/contact-us
- Technical Issues Feedback: https://www.compdf.com/support
- Contact Email: support@compdf.com

Thanks,
The ComPDFKit Team