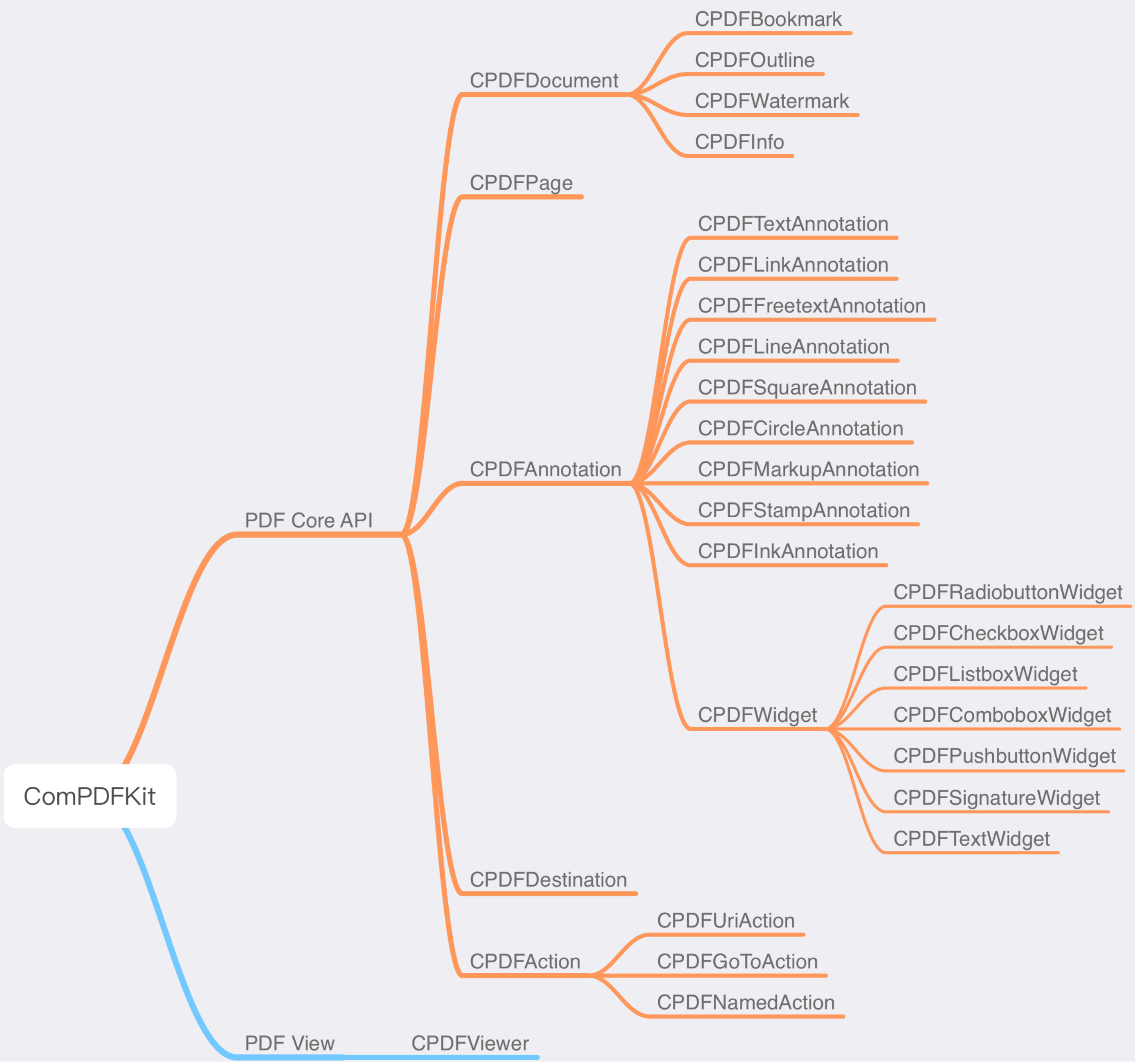
Contents

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 extract, form data collection, and editing capabilities into their applications and services running.

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 PDF SDK.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 PDF SDK.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, Freetext, 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.

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.

Conversion component offers:

- PDF to PDF/A.

Document comparison component offers:

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

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 program with a few lines of C# code. Takes just a few minutes and gets started.

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

2.1 Requirements

Please make sure that the .NET Desktop Development and .NET Framework 4.6.1+ development tools workload is part of your installation.

- Windows 7,8,10, and 11 (32-bit, 64-bit) .
- Visual Studio 2017 or higher.
- .NET Framework 4.6.1 or higher.

2.2 Windows Package Structure

The package of ComPDFKit PDF SDK for Windows includes the following files as shown in Figure 2-1:

- **lib** - Include the ComPDFKit dynamic library (x86,x64).
- **ComPDFKit.Demo** - A folder containing Windows sample projects.
- **api_reference_windows** - API reference.
- **developer_guide_windows.pdf** - Developer guide.
- **release_notes.txt** - Release information.

- **legal.txt** - Legal and copyright information.

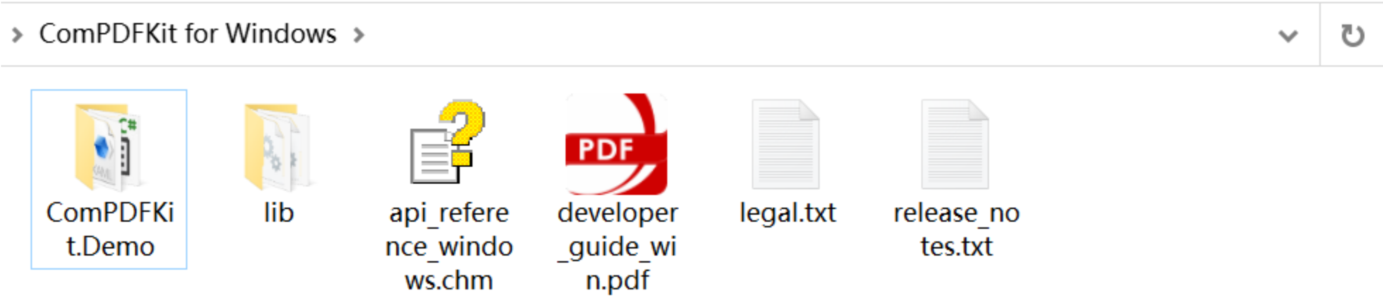
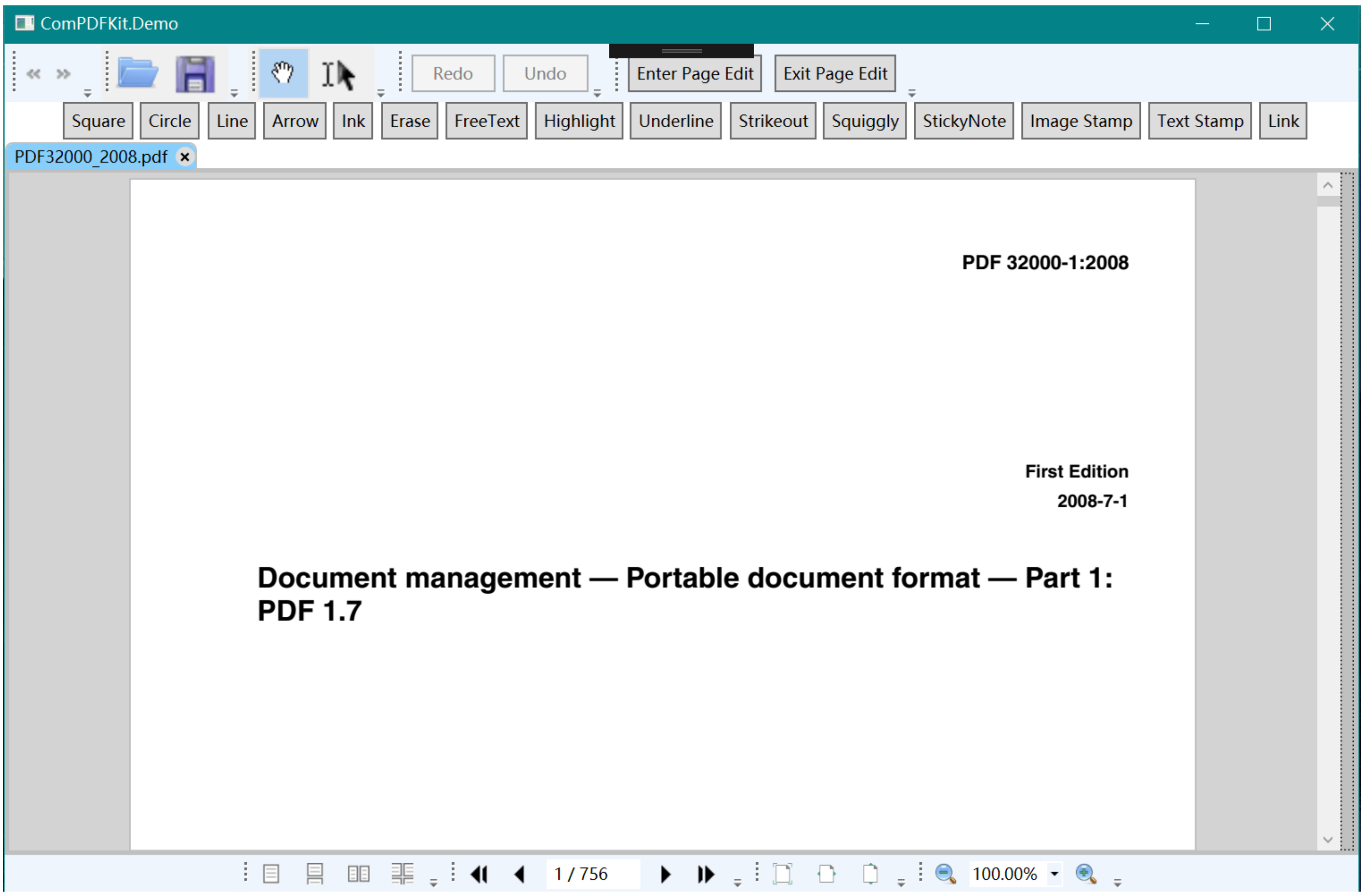


Figure 2-1

2.3 How to Run a Demo

ComPDFKit PDF SDK for Windows provides one demo in C# for developers to learn how to call the SDK on Windows. You can find them in the ***"ComPDFKit.Demo"*** folder. In this guide, it takes the "C#" demo as an example to show how to run it in Visual Studio 2017.

1. Double-click the ***"ComPDFKit.Demo.sln"*** found in the ***"ComPDFKit.Demo"*** folder to open the demo in Visual Studio 2017.
2. Click on ***"Start"*** to run the demo on a Windows device. In this guide, a Windows10 device will be used as an example. After building the demo successfully, click the ***"Open"*** button,and select a PDF Document, then it will be opened and displayed.



Note: *This is a demo project, presenting completed ComPDFKit PDF SDKfunctions. 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

This section will help you to quickly get started with ComPDFKit PDF SDK to make a Windows project in C# with step-by-step instructions, which include the following steps:

1. Create a new Windows project in C#.
2. Integrate ComPDFKit into your project.
3. Apply the license key.
4. Display a PDF document.

2.4.1 Create a New Windows Project in C#

In this guide, we use Visual Studio 2017 to create a new Windows project.

Fire up Visual Studio 2017, choose **File** -> **New** -> **Project...**, and then select **Visual C#**->**Windows Desktop** -> **WPF App(.NET Framework)** as shown in Figure 2-2.

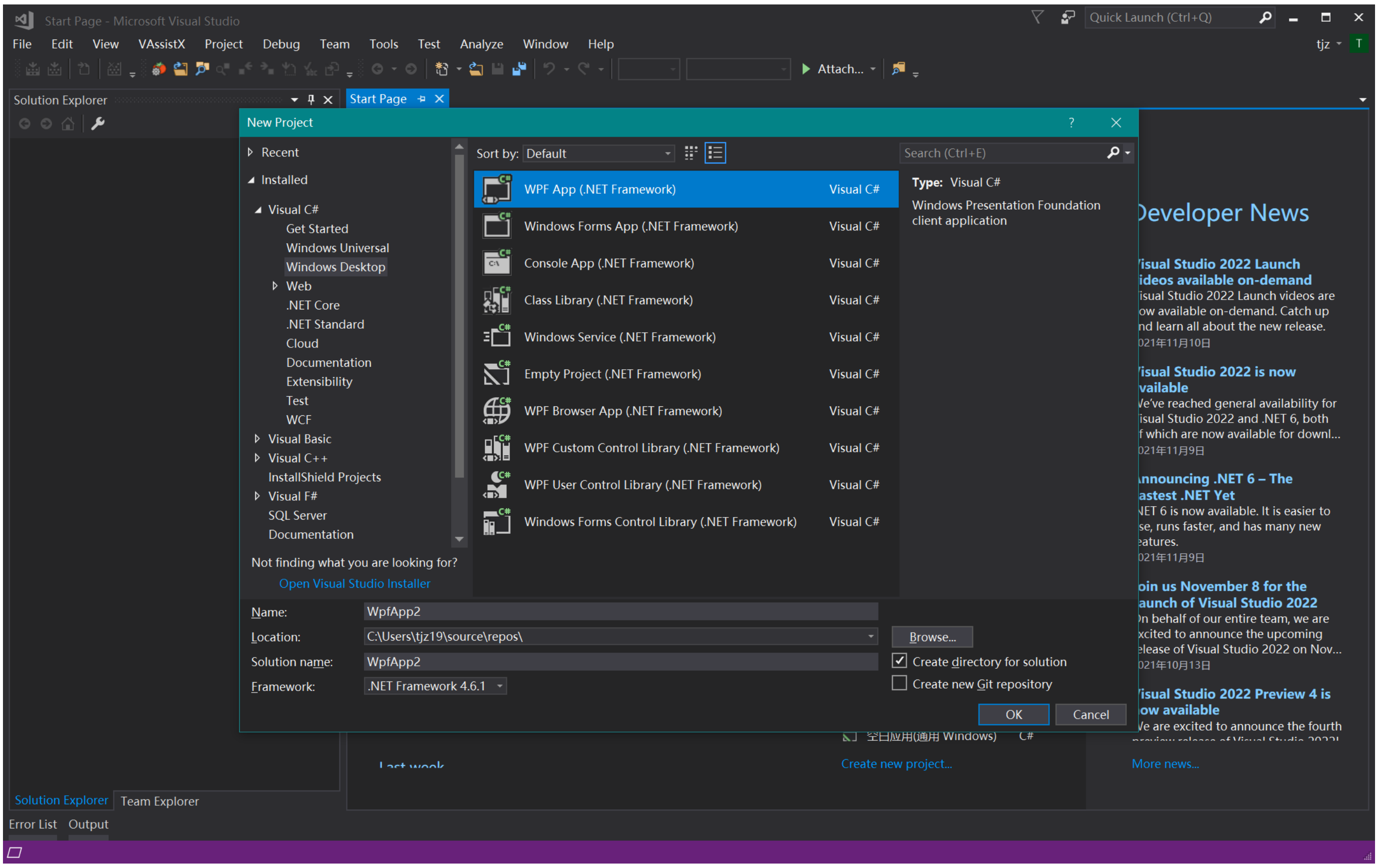


Figure 2-2

Choose the options for your new project as shown in Figure 2-3. Please make sure to choose .NET Framework 4.6.1 as the programming framework.

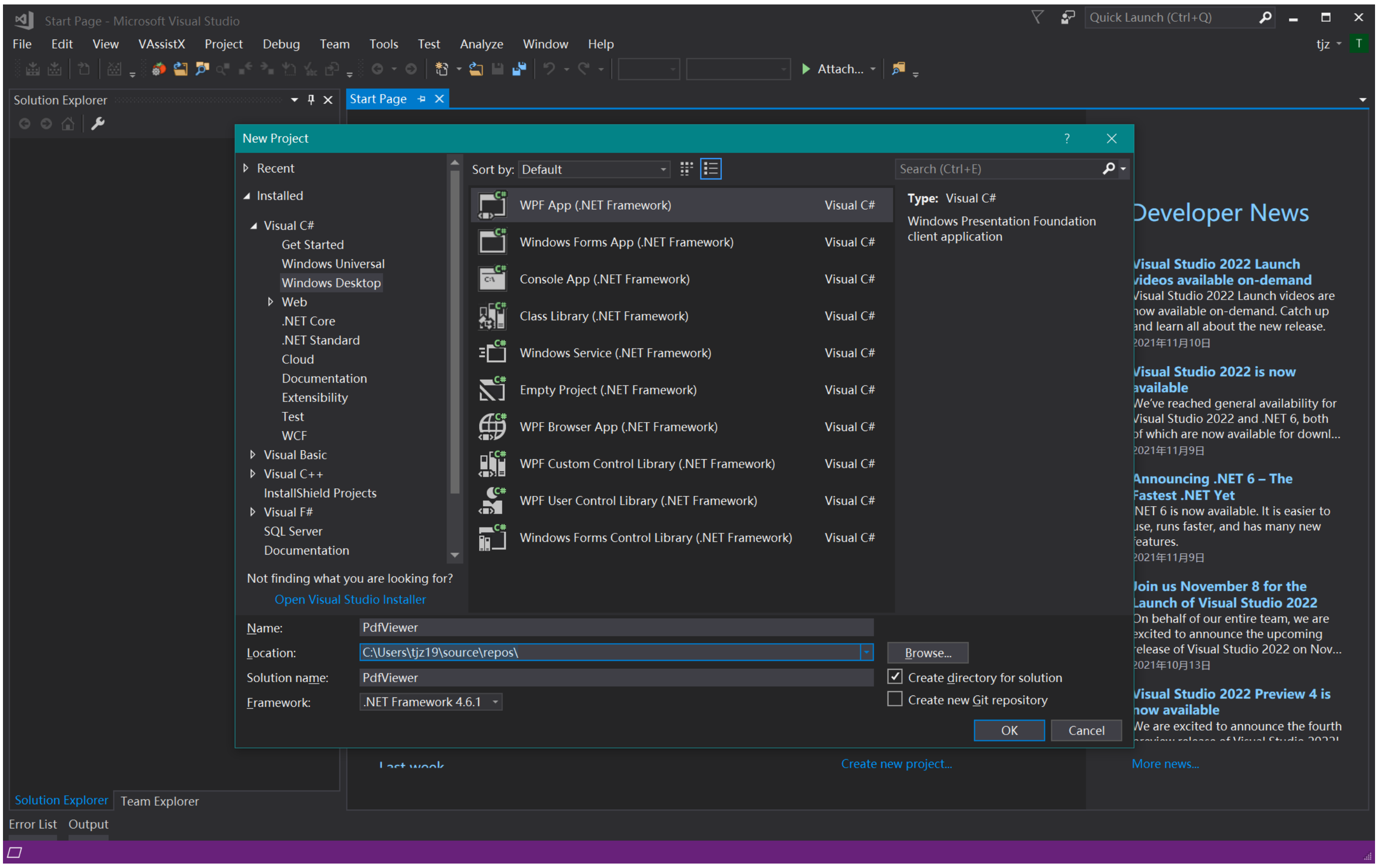
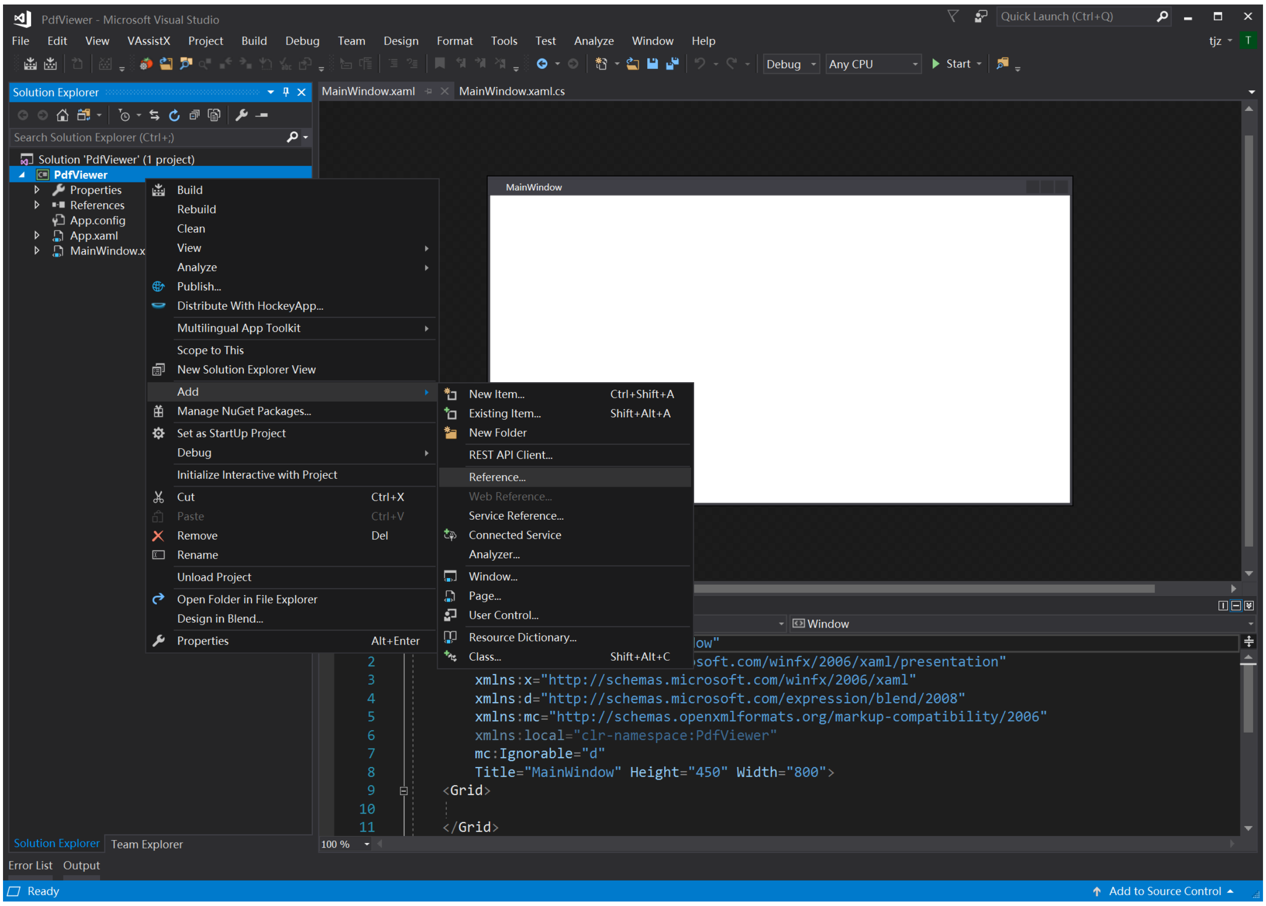


Figure 2-3

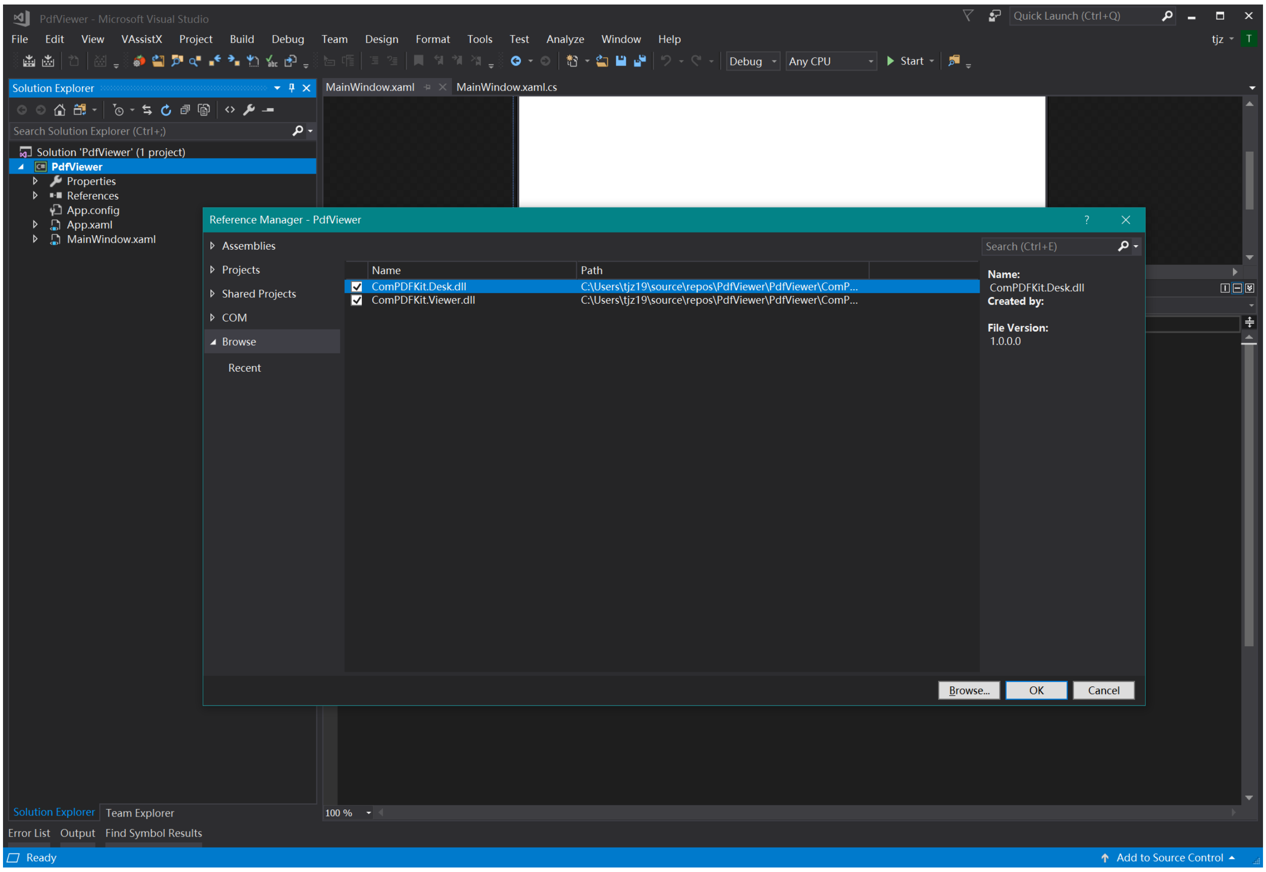
Place the project to the location as desired. Then, click **OK**.

2.4.2 Integrate ComPDFKit PDF SDK into your projects

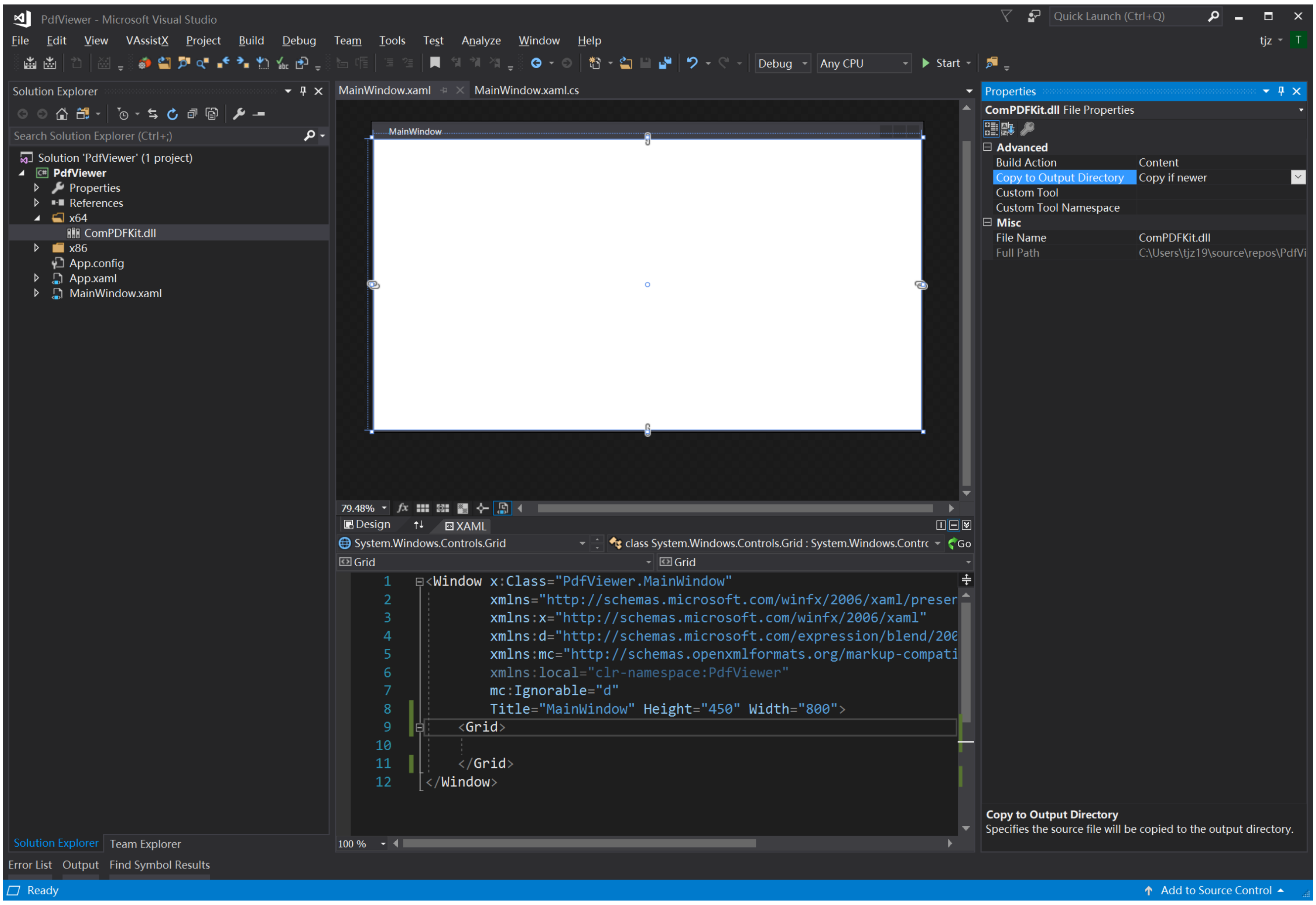
1. Copy the ***"ComPDFKit.Desk.dll"*** and ***"ComPDFKit.Viewer.dll"*** DLL files and ***"x64"*** or ***"x86"*** folder depending on your build configurations in the ***"lib"*** folder to the project ***"PdfViewer"*** folder.
2. Add ComPDFKit PDF SDK dynamic library to **References**. In order to use ComPDFKit PDF SDK APIs in the project, you must first add a reference to it.
   - In Solution Explorer, right-click the ***"PdfViewer"*** project and click **Add -> Reference...**



- In the Add Reference dialog, click Browse tab, navigate to the ***"PdfViewer"*** folder, select ***"ComPDFKit.Desk.dll"*** and ***"ComPDFKit.Viewer.dll"*** dynamic library, and then click **OK**.



3. Add ***"ComPDFKit.dll"*** to the project. Include the ***"x64"*** or ***"x86"*** folder into the project. 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

It is important that you set the license key before using any ComPDFKit PDF SDK classes.

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

    string licenseKey = "***";
    string licenseSecret = "***";
    CPDFSDKVerifier.LicenseErrorCode verifyResult =
CPDFSDKVerifier.LicenseVerify(licenseKey, licenseSecret);
    if (verifyResult != CPDFSDKVerifier.LicenseErrorCode.LICENSE_ERR_SUCCESS)
        return false;

    return true;
}
```

2.4.4 Display a PDF Document

So far, we have added ***"ComPDFKit.Desk.dll"***, ***"ComPDFKit.Viewer.dll"*** and ***"ComPDFKit.dll"*** to the ***"PdfViewer"*** project, and finished the initialization of the ComPDFKit. Now, let's start building a simple PDF viewer with just a few lines of code.

Then, add the following code to MainWindow.xaml and MainWindow.xaml.cs to display a PDF document. It's really easy to present a PDF on screen. All you need is to create a `CPDFDocument` object and then show it with a `CPDFViewer` object.

MainWindow.xaml

```
<Window x:Class="PdfViewer.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:PdfViewer"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800"
        Loaded="Window_Loaded">
    <Grid x:Name="PDFGrid">

    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;
using ComPDFKit.PDFDocument;
using ComPDFKitViewer.PdfViewer;

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

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            string pdfPath = "";
            CPDFViewer pdfViewer = new CPDFViewer();
            pdfViewer.InitDocument(pdfPath);
```

```
            if (pdfViewer.Document != null && pdfViewer.Document.ErrorType ==
CPDFDocumentError.CPDFDocumentErrorSuccess)
            {
                pdfViewer.Load();
                PDFGrid.Children.Add(pdfViewer);
            }
        }
    }
}
```

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 program 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();
```

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. However, even though this is the fastest way of saving a document, it does come with the cost of incrementing the file size with each save.

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

The viewer is a viewing and rasterization scroll view component that handles a number of viewing operations including rendering, caching, high-level zoom rendering, and layout modes. It offers developers a way to quickly embed a highly configurable PDF viewer in any Windows WPF program.

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

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

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

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

`ViewMode::Book`: Displays 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.

  - Scrolls to the specified page, use function `CPDFViewer.GoToPage(int pageIndex)`.
  - Goes to the specified destination, destinations include a page and a point on the page specified in page space, use function `CPDFViewer.GoToPage(int pageIndex, Point pagePoint)`.
- Outline

  Outline allows users to quickly locate and link their point of interest within a PDF document. Each outline contains a destination or actions to describe where it links to. It is a tree-structured hierarchy, so function `CPDFDocument.GetOutlineRoot()` must be called first to get the root of the whole outline tree before accessing the outline tree. Here, "root outline" is an abstract object which can only have some child outline without the next sibling outline and any data (includes outline data, destination data, and action data). It cannot be shown on the application UI since it has no data. You can also use function `CPDFDocument.CreateOutlineRoot()` to create a new root outline.

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

  - To access the parent outline, use function `CPDFOutline.GetParent()`.
  - To access the child outline,use function `CPDFDocument.GetOutlineList()`.
  - To insert a new outline, use function `CPDFOutline.InsertChildAtIndex(CPDFDocument document, int index)`.
  - To remove a outline, use function `CPDFOutline.RemoveFromParent(CPDFDocument document)`.
  - To move a outline, use function `CPDFOutline.MoveChildAtIndex(CPDFDocument document, CPDFOutline child, int index)`.
- 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 function `CPDFDocument.GetBookmarkList()`.
- To access a bookmark for page, use function `CPDFDocument.BookmarkForPageIndex(int pageIndex)`.
- To add a new bookmark, use function `CPDFDocument.AddBookmark(CPDFBookmark bookmark)`.
- To remove a bookmark, use 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.

  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::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

`CPDFView` 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.

3.3 Annotations

In addition to its primary textual content, a PDF file can contain annotations that represent links, form elements, highlighting circles, textual notes, and so on. Each annotation is associated with a specific location on a page and may offer interactivity with the user. Annotations allow users to mark up and comment on PDFs without altering the original author's content.

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.

3.3.1 Annotation Types

ComPDFKit PDF SDK supports all common annotation types:

- Note
- Link
- Free Text
- Shapes: Square, Circle, and Line
- Markup: Highlight, Underline, Strikeout, and Squiggly
- Stamp
- Ink
- Sound

These are standard annotations (as defined in the PDF Reference) that 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. A `CPDFAnnotation` object by itself is not useful, only subclasses (like `CPDFCircleAnnotation`, `CPDFTextAnnotation`) are interesting. In parsing a PDF however, any unknown or unsupported annotations will be represented as this base class.

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

  To 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

To 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

To 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

To 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.*

- Markup

  To 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();
```

- Stamp

  To 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

  To 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

  To 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();
```

3.3.4 Delete Annotations

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.

- Importing 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");
  ```

- Exporting 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 (such as text boxes, checkboxes, radio buttons, drop-down lists, pushbuttons, and signatures).

`CPDFWidget` is the base class for all form fields, and `CPDFWidget` is subclass for `CPDFAnnotation`. A `CPDFWidget` object by itself is not useful, only subclasses (`CPDFCheckboxWidget`, `CPDFRadiobuttonWidget`, `CPDFAnnotationTextWidget`, `CPDFPushbuttonWidget`, `CPDFListboxWidget`, `CPDFComboboxWidget`, `CPDFTextWidget` , `CPDFSignatureWidget`) are interesting. In parsing a PDF however, any unknown or unsupported form fields will be represented as this base class.

We have to differentiate between field types and annotation objects:

Type	Annotation Object
Check Boxes	`CPDFCheckboxWidget`
Radio Buttons	`CPDFRadiobuttonWidget`
Push Buttons	`CPDFPushbuttonWidget`
List Boxes	`CPDFListboxWidget`
Combo Boxes	`CPDFComboboxWidget`
Text	`CPDFTextWidget`
Signatures	`CPDFSignatureWidget`

3.4.2 Create & Edit Form Fields

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

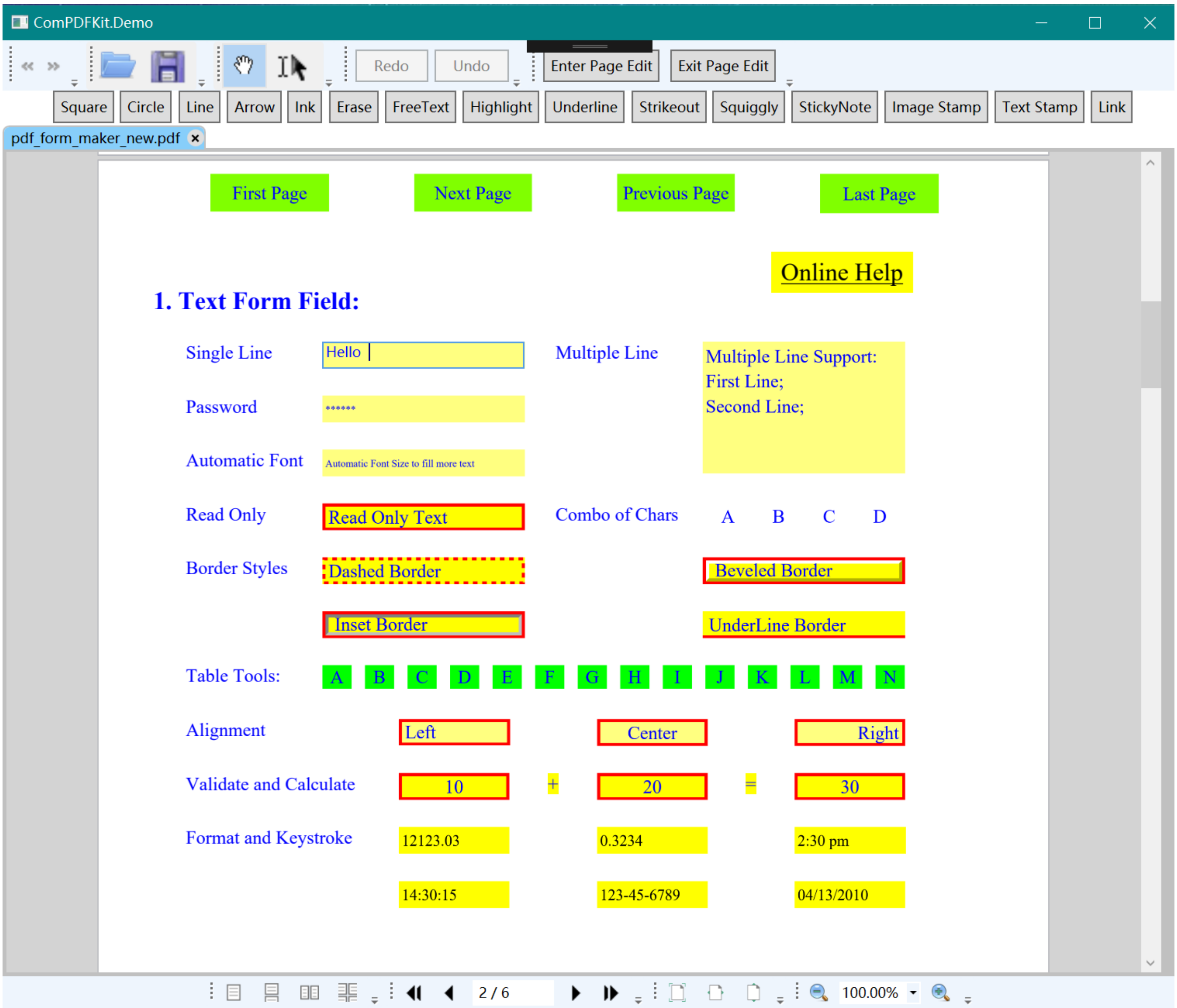
```
CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
CPDFPage page = document.PageAtIndex(0);

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

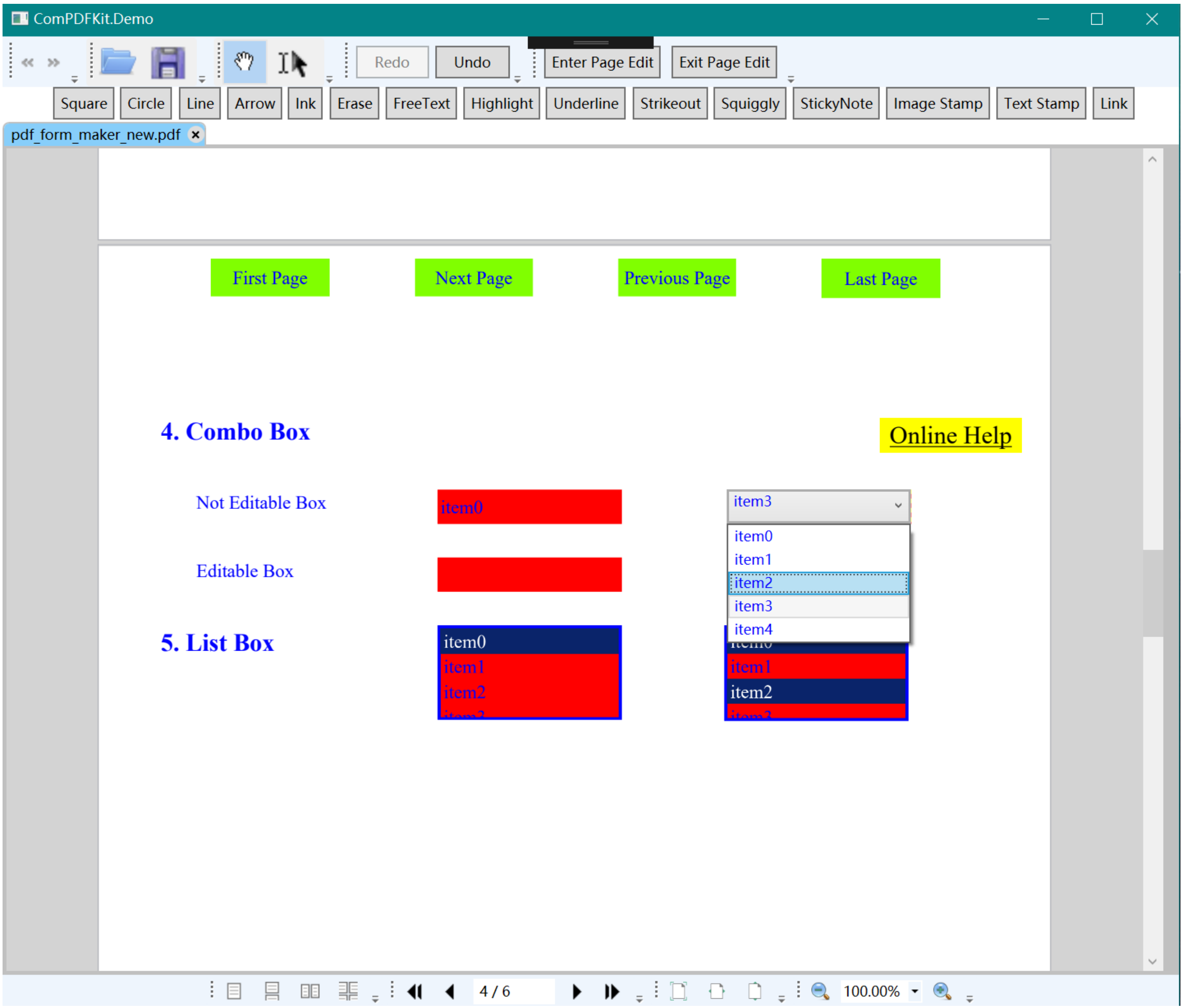
3.4.3 Fill Form Fields

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

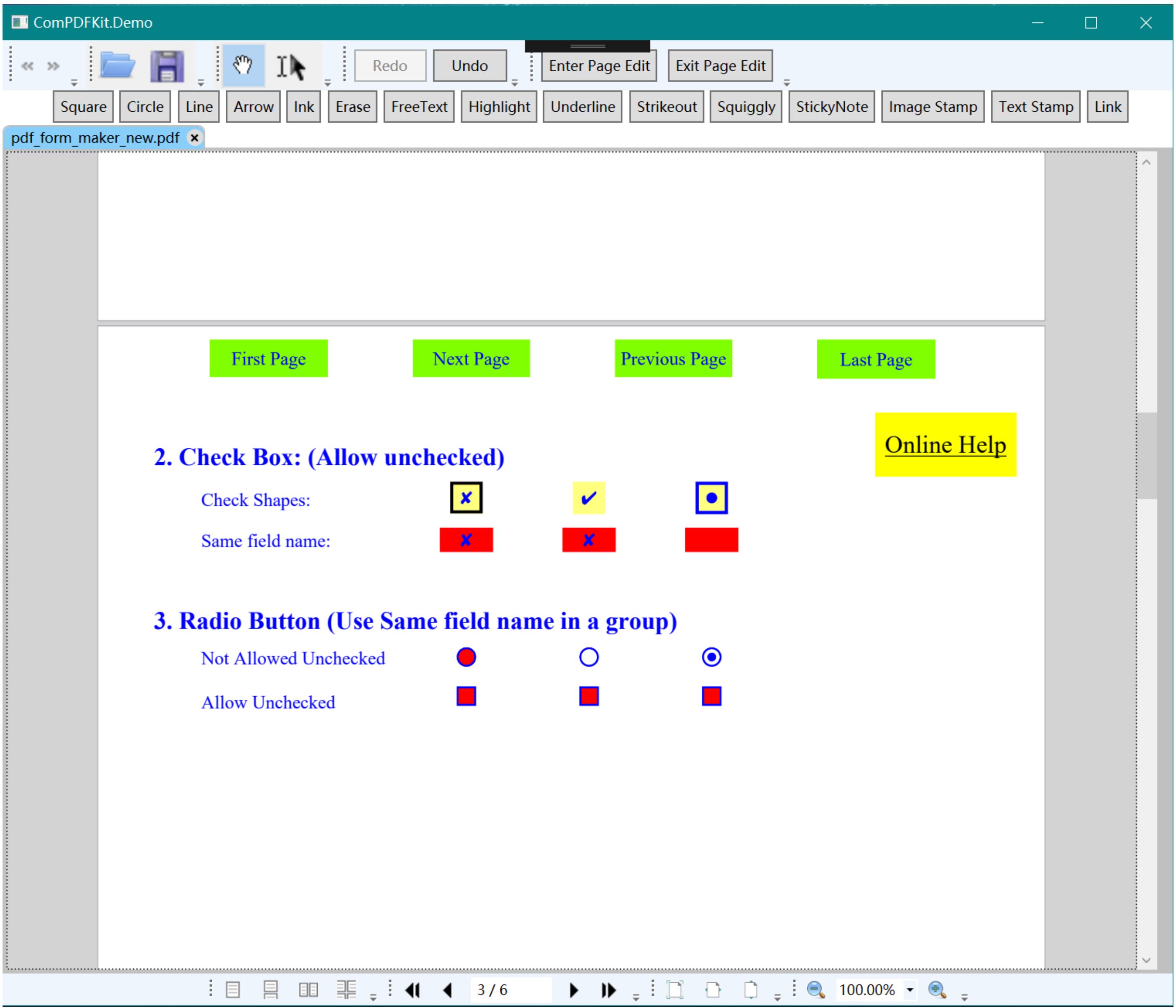
To fill in a text form element, tap it and then type text using either the onscreen keyboard or an attached hardware keyboard. Then tap either the Done button above the keyboard or any blank area on the page to deselect the form element, which will commit the changes.



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

Delete form fields works the same as deleting annotations, and check delete 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 ranges 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 ranges 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 function `CPDFDocument.RemovePages(int[] pageIndexs)`.
- To insert a blank page into a PDF document, use function `CPDFDocument.InsertPage(int pageIndex, double width, double height, string imagePath="")`.
- To insert an image as an entire page into a PDF document, use function `CPDFDocument.InsertPage(int pageIndex, double width, double height, string imagePath)`.

- To insert specific page from one document to another, use function `CPDFDocument.ImportPagesAtIndex(CPDFDocument otherDocument, string pageRange,int pageIndex)`.
- To move a page to a new location, use function `CPDFDocument.MovePage(int startIndex, int endIndex)`.
- To exchange the location of two document pages, use function `CPDFDocument.ExchangePage(int firstIndex, int secondIndex)`.
- To replace original document pages with new pages from a different document, use 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 PDF's 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 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 watermark, and more. For controlling over document security in ComPDFKit PDF SDK, security handlers perform user authorization and sets various permissions over PDF documents.

3.6.1 PDF Permission

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

A user password (also known as an open password) requires a user to type a password to open the PDF. If you want to open a document with a user password programmatically, you can use the `CPDFDocument.UnlockWithPassword(string password)` API.

An owner password (also known as a permissions password) requires a password to change permission settings. When an owner password is set, you can configure a set of permissions. For example, you can configure an 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.

The PDF specification defines the permissions 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 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.

  To create a password-protected document:

  ```
  CPDFDocument document = CPDFDocument.InitWithFilePath("filePath");
  CPDFPermissionsInfo permission = new CPDFPermissionsInfo();
  document.Encrypt("UserPassword", "ownerPassword", permission);
  document.WriteToFilePath("savePath");
  ```

- 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 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.Descrypt(string filePath)` method:

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

3.6.2 Watermark

Adding a non-removable watermark to documents can discourage viewers from sharing your content or taking screenshots.

- To add a watermark, use function `CPDFDocument.CreateWatermark(C_Watermark_Type type)`.
- To remove all the watermarks in 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.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.CreateWatermark();

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

3.6.3 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();
```

3.7 Conversion

3.7.1 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");
```

3.8 PDF Editing

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

- Edit text and images
- Remove specific content from existing pages
- Insert or append new content to existing pages
- Undo or redo any change

3.8.1 Initialize PDF Editing

The following code demonstrates how to do PDF editing initialization.

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

3.8.2 How to Edit Text and 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 using 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 });
}
```

3.8.3 How to Set Text and Image Properties

You can use `CPDFViewer`'s `PDFEditActiveHandler` event to set the text and image properties. The following code will show you how to do this.

```
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 = 14;
        e.TextAlign = TextAlignType.AlignJustify;
        e.UpdatePDFEditByEventArgs();
    }

    //Image properties.
    if (e.EditType == CPDFEditType.EditImage)
    {
        e.VerticalMirror = true;
```

```
        e.HorizontalMirror = true;
        e.ClipImage = true;
        e.Rotate = 90;
        e.ReplaceImagePath = "***";
        e.UpdatePDFEditByEventArgs();
    }
}
```

3.8.4 How to Insert Text and Image

You can use `CPDFViewer`'s `SetPDFEditCreateType` method to set 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);
```

3.8.5 How to Redo and Undo

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

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

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

3.9 Document Comparison

ComPDFKit provides two methods to compare documents:

- Overlay Comparison
- Content Comparison

3.9.1 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.

3.9.2 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 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 of 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.

4 Support

4.1 Reporting Problems

Thank you for your interest in ComPDFKit PDF SDK, the only 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

Home Link:

https://www.compdf.com

Support & General Contact:

Email: support@compdf.com

Thanks,
The ComPDFKit Team