1 Overview

ComPDFKit for Flutter is a comprehensive SDK that allows you to quickly add PDF fuctions to any Flutter application, such as viewer, annotations, editing PDFs, forms and signatures.

More information can be found at https://www.compdf.com/

1.1 ComPDFKit for Flutter

ComPDFKit for Flutter consists of two elements.

The two elements for ComPDFKit for Flutter:

- **PDF Core API**

  The Core API can be used independently for document rendering, analysis, text extraction, text search, form filling, password security, annotation creation and manipulation, and much more.

- **PDF View**

  The PDF View 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. The View Control is derived from platform-related viewer classes (e.g. `UIView` on iOS) and allows for extension to accommodate specific user needs.

1.2 Key Features

Viewer
component offers:

- Standard page display modes, including Scrolling, Double Page, Crop Mode, 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.
- Predefine annotations.

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, including Delete pages, Insert pages, Crop 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.
- Undo or redo any change.
- Find and Replace.

Security component offers:

- Encrypt and decrypt PDFs, including Permission setting and Password protected.
- Create, edit, and remove watermark.

Watermark component offers:

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

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 `bundle ID` or `applicationId` 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 to any third party is prohibited.

Online License: We have introduced an online license mechanism for enhanced license management. Through the online approach, you can manage and update your license more flexibly to meet the specific requirements of your project.

Offline License: In scenarios with high security requirements, no internet connectivity, or offline environments, we provide the option of an offline license. The offline license allows authorization and usage of the ComPDFKit PDF SDK when internet connectivity is not available.

2 Get Started

2.1 Requirements

Before starting, please make sure that you have already met the following prerequisites.

2.1.1 Get ComPDFKit License Key

ComPDF offers two types of license keys: a free 30-day trial license and a commercial license.

How to Get Free Trial License

Method 1: Apply Online

If you want to get PDF SDK trials for **Web, Windows, Android, iOS, Flutter, and React Native**, simply apply for a 30-day free trial license online.

You can check features supported by the free trial license on our Pricing page.

Method 2: Contact Sales

For other platforms or features outside of the trial license, feel free to contact our sales team.

How to Get Formal License

ComPDFKit PDF SDK is a commercial SDK that requires a license for application release. Any documents, sample code, or source code distribution from the released package of ComPDFKit to any third party is prohibited.

Method 1: Purchase Online

We allow developers to purchase formal licenses for the PDF SDK for **iOS** from our Pricing page, including common and standard PDF features. For any question about the plans, please contact our sales.

Method 2: Contact Sales

To get formal licenses for additional platforms, advanced features, custom requirements, or quote inquiries, feel free to contact our sales team.

For the Android PDF SDK, the commercial license must be bound to your application's ApplicationId.

2.1.2 Download the PDF SDK

Download the ComPDFKit Android PDF SDK from Github or pub.dev.

2.1.3 System Requirements

Android

Please install the following required packages:

- The latest stable version of Flutter
- The latest stable version of Android Studio
- The Android NDK
- An Android Virtual Device

Operating Environment Requirements:

- A minSdkVersion of `21` or higher.
- A `compileSdkVersion` of `34` or higher.
- A `targetSdkVersion` of `34` or higher.
- Android ABI(s): x86, x86_64, armeabi-v7a, arm64-v8a.

iOS

Please install the following required packages:

- The latest stable version of Flutter
- The latest stable version of Xcode
- The latest stable version of CocoaPods. Follow the CocoaPods installation guide to install it.

Operating Environment Requirements:

- The iOS **12.0** or higher.
- The Xcode 12.0 or newer for Objective-C or Swift.

2.2 Creating a New Project

2.2.1 Android

1. Create a Flutter project called `example` with the `flutter` CLI:

```
flutter create --org com.compdfkit.flutter example
```

2. In the terminal app, change the location of the current working directory to your project:

```
cd example
```

3. open `example/android/app/src/main/AndroidManifest.xml` , add `Internet Permission` and `Storage Permission`:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.compdfkit.flutter.example">

+    <uses-permission android:name="android.permission.INTERNET"/>

    <!-- Required to read and write documents from device storage -->
+    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
+    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <!-- Optional settings -->
+    <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>

    <application
+       android:requestLegacyExternalStorage="true">

    </application>
</manifest>
```

4. Open the app's Gradle build file, `android/app/build.gradle`:

```
open android/app/build.gradle
```

5. Modify the minimum SDK version, All this is done inside the `android` section:

```
 android {
     defaultConfig {
-        minSdkVersion flutter.minSdkVersion
+        minSdkVersion 21
         ...
     }
 }
```

6. Open the project's main activity class, `android/app/src/main/java/com/example/compdfkit/flutter/example/MainActivity.java`, Change the base `Activity` to extend `FlutterFragmentActivity`:

```
- import io.flutter.embedding.android.FlutterActivity;
+ import io.flutter.embedding.android.FlutterFragmentActivity;

- public class MainActivity extends FlutterActivity {
+ public class MainActivity extends FlutterFragmentActivity {
}
```

Alternatively you can update the `AndroidManifest.xml` file to use `FlutterFragmentActivity` as the launcher activity:

```
<activity
-     android:name=".MainActivity"
+     android:name="io.flutter.embedding.android.FlutterFragmentActivity"
      android:exported="true"
      android:hardwareAccelerated="true"
      android:launchMode="singleTop"
      android:theme="@style/LaunchTheme"
      android:windowSoftInputMode="adjustPan">
```

> **Note:** `FlutterFragmentActivity` is not an official part of the Flutter SDK. If you need to use `CPDFReaderWidget` in ComPDFKit for Flutter, you need to use this part of the code. You can skip this step if you don't need to use.

7. Add the ComPDFKit dependency in `pubspec.yaml`

```
 dependencies:
   flutter:
     sdk: flutter
+  compdfkit_flutter: ^2.5.0
```

8. From the terminal app, run the following command to get all the packages:

```
flutter pub get
```

2.2.2 iOS

1. Create a Flutter project called `example` with the `flutter` CLI:

```
flutter create --org com.compdfkit.flutter example
```

2. In the terminal app, change the location of the current working directory to your project:

```
cd example
```

3. Add the ComPDFKit dependency in `pubspec.yaml`

```
 dependencies:
   flutter:
     sdk: flutter
+  compdfkit_flutter: ^2.5.0
```

4. From the terminal app, run the following command to get all the packages:

```
flutter pub get
```

5. Open your project's Podfile in a text editor:

```
open ios/Podfile
```

6. Update the platform to **iOS 12** and add the ComPDFKit Podspec:

```
- platform :ios, '9.0'
+ platform :ios, '12.0'
 ...
 target 'Runner' do
   use_frameworks!
   use_modular_headers!`

   flutter_install_all_ios_pods File.dirname(File.realpath(__FILE__))
+  pod "ComPDFKit",
podspec:'https://file.compdf.com/cocoapods/ios/compdfkit_pdf_sdk/2.5.0/ComPDFKit.podspec'
+  pod "ComPDFKit_Tools",
podspec:'https://file.compdf.com/cocoapods/ios/compdfkit_pdf_sdk/2.5.0/ComPDFKit_Tools.pod
spec'
 end
```

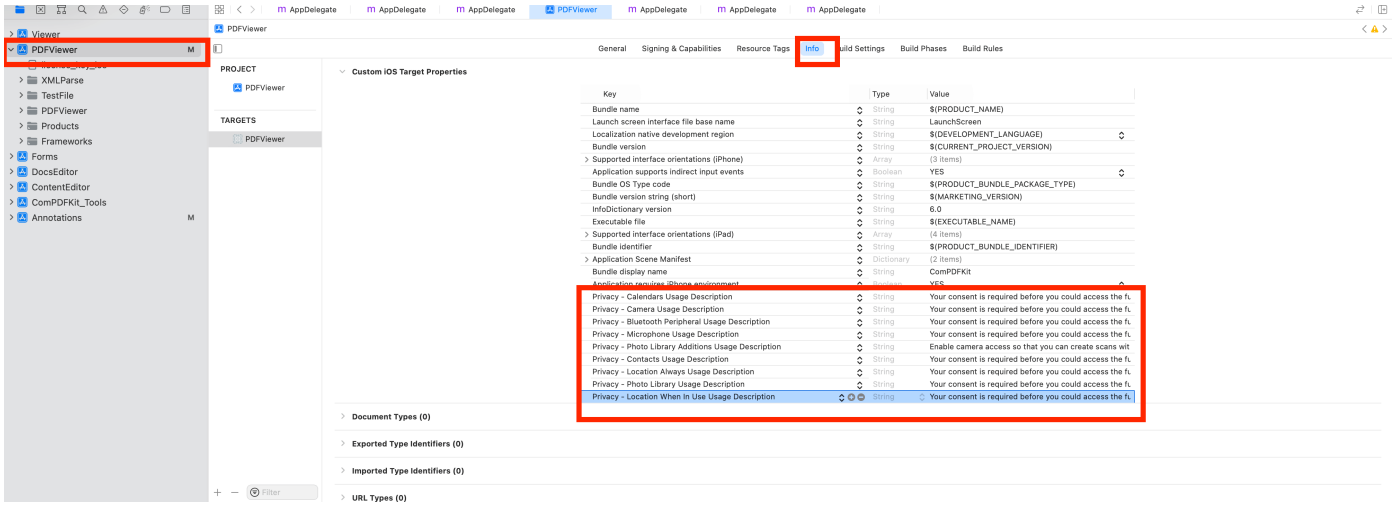
7. Go to the `example/ios` folder and run the `pod install` command:

```
pod install
```

Note: If SSL network requests fail to download the `ComPDFKit` library when you run `pod install`, you can see the processing method in Troubleshooting).

8. To protect user privacy,

To protect user privacy, before accessing the sensitive privacy data, you need to find the "***Info***" configuration in your iOS 10.0 or higher iOS project and configure the relevant privacy terms as shown in the following picture.



```
<key>NSCameraUsageDescription</key>
<string>Your consent is required before you could access the function.</string>

<key>NSMicrophoneUsageDescription</key>
<string>Your consent is required before you could access the function.</string>

<key>NSPhotoLibraryAddUsageDescription</key>
<string>Your consent is required before you could access the function.</string>

<key>NSPhotoLibraryUsageDescription</key>
<string>Your consent is required before you could access the function.</string>

<key>NSAppTransportSecurity</key>
  <dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
  </dict>
```

2.3 Apply the License Key

If you haven't get a license key, please check out how to obtain a license key.

ComPDFKit PDF SDK currently supports two authentication methods to verify license keys: online authentication and offline authentication.

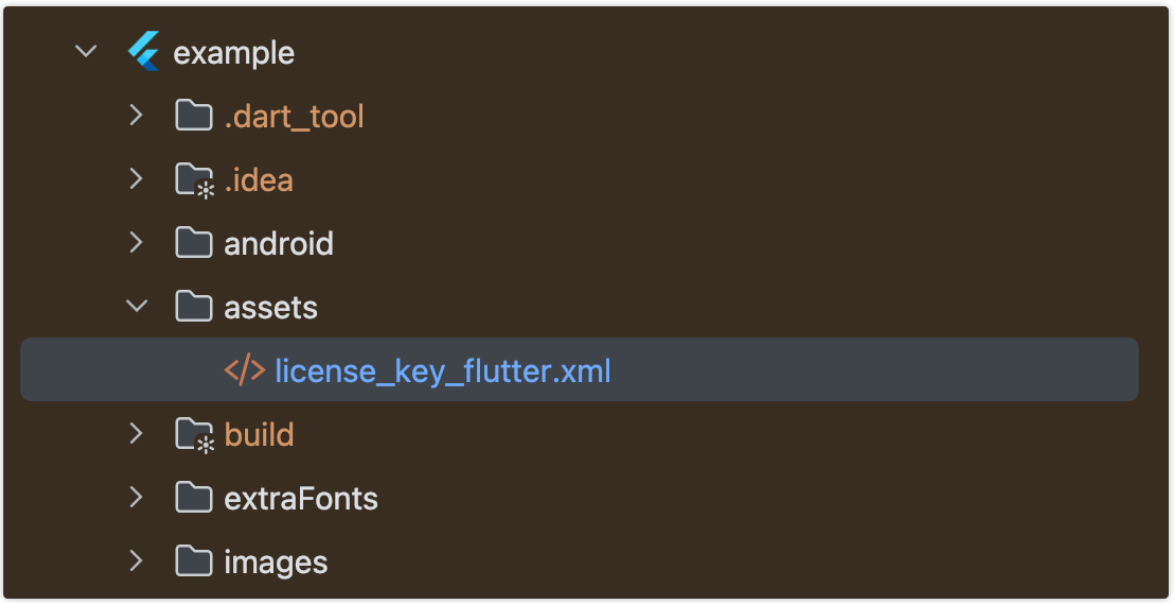
Learn about:

What is the authentication mechanism of ComPDFKit's license?

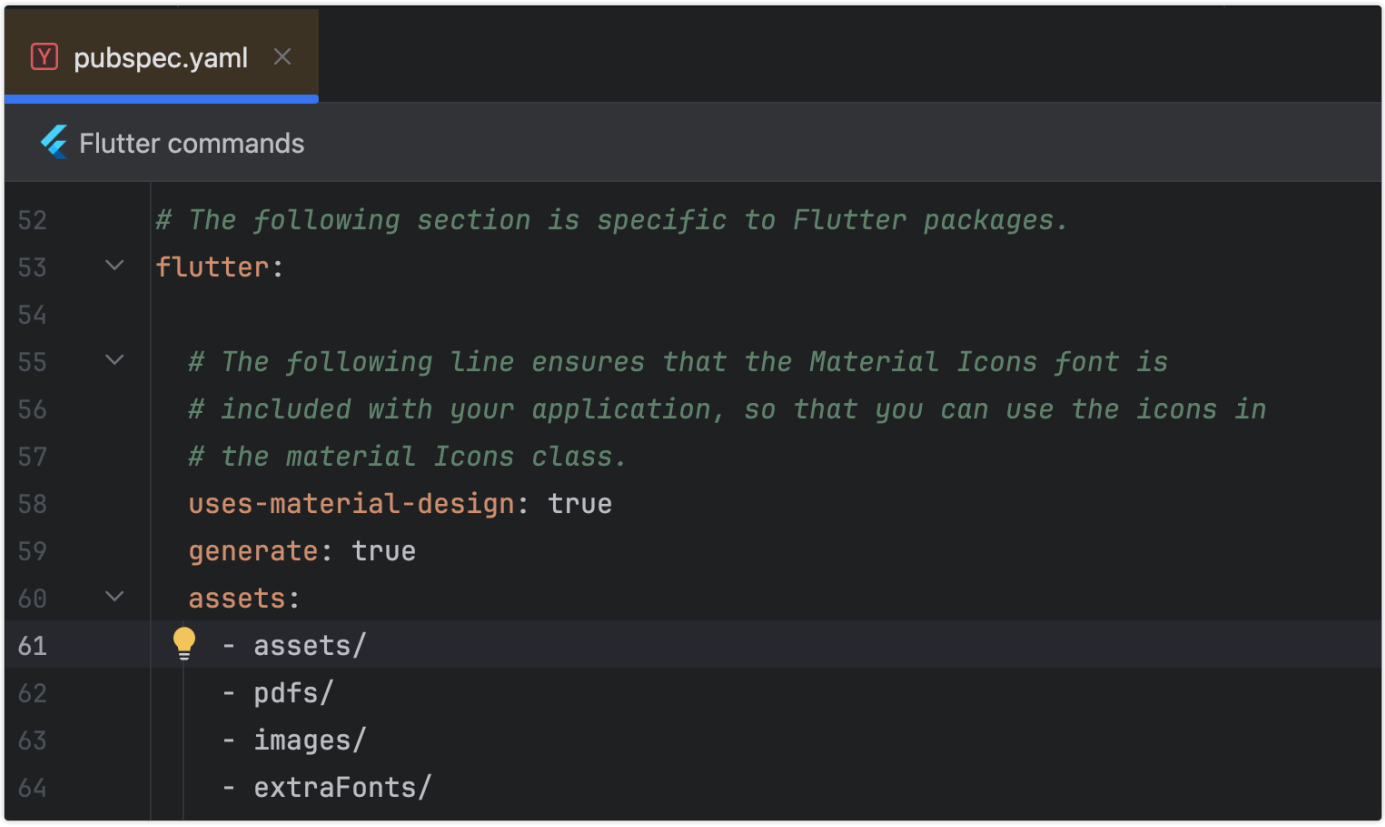
What are the differences between Online Authentication and Offline Authentication?

Accurately obtaining the license key is crucial for the application of the license.

1. In the email you received, locate the `XML` file containing the license key.
2. Copy the License.xml file into the **assets directory** of your Flutter project.



3. Open the pubspec.yaml file of your project and configure the flutter: section to enable the assets directory.



4. Initialize the SDK:

```
// Include the license in Flutter assets and copy to device storage
// Add `license_key_flutter.xml` to your Flutter project's assets directory;
File licenseFile = await CPDFFileUtil.extractAsset(context,
'assets/license_key_flutter.xml');
await ComPDFKit.initWithPath(licenseFile.path);
```

Alternative methods (optional)

```
// Android
// Copy the license_key_flutter.xml file into the `android/app/src/main/assets` directory
of your Android project:
await ComPDFKit.initWithPath('assets://license_key_flutter.xml');

// iOS
// Copy the license_key_flutter.xml file into your iOS project directory (or a readable
location):
await ComPDFKit.initWithPath('license_key_flutter.xml');
```

2.4 Run Project

1. Open `lib/main.dart` and replace the entire content with the following code. And fill in the license provided to you in the `ComPDFKit.init` method.

There are 2 different ways to use ComPDFKit Flutter API:

- Present a document via a plugin.
- Show a ComPDFKit document view via a Widget.

Usage Plugin

Open `lib/main.dart`,replace the entire file with the following:

```
import 'dart:io';

import 'package:compdfkit_flutter/compdfkit.dart';
import 'package:compdfkit_flutter/configuration/cpdf_configuration.dart';
import 'package:compdfkit_flutter/util/cpdf_file_util.dart';
import 'package:flutter/material.dart';

const String _documentPath = 'pdfs/PDF_Document.pdf';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({super.key});

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  void initState() {
    super.initState();
    _init();
  }

  void _init() async {
    File licenseFile = await CPDFFileUtil.extractAsset(context,
'assets/license_key_flutter.xml');
    final result = await ComPDFKit.initWithPath(licenseFile.path);
    debugPrint('ComPDFKit Init Result: $result');
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: SafeArea(
          child: Center(
            child: ElevatedButton(
              onPressed: () async {
                showDocument(context);
```

```
              },
              child: const Text(
                'Open Document',
              )),
          ))),
    );
  }

  void showDocument(BuildContext context) async {
    var pdfFile = await CPDFFileUtil.extractAsset(_documentPath);
    var configuration = CPDFConfiguration();
    // Present a document via a plugin.
    ComPDFKit.openDocument(pdfFile.path,
                           password: '', configuration: configuration);
  }
}
```

Usage Widget

Open `lib/main.dart`,replace the entire file with the following:

```
import 'dart:io';

import 'package:compdfkit_flutter/compdfkit.dart';
import 'package:compdfkit_flutter/configuration/cpdf_configuration.dart';
import 'package:compdfkit_flutter/widgets/cpdf_reader_widget.dart';
import 'package:compdfkit_flutter/util/cpdf_file_util.dart';
import 'package:flutter/material.dart';

const String _documentPath = 'pdfs/PDF_Document.pdf';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({super.key});

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  String? _document;

  @override
  void initState() {
    super.initState();
    _init();
    _getDocumentPath().then((value) {
      setState(() {
        _document = value;
```

```
      });
    });
  }

  void _init() async {
    /// Please replace it with your ComPDFKit license
    File licenseFile = await CPDFFileUtil.extractAsset(context,
'assets/license_key_flutter.xml');
    final result = ComPDFKit.initWithPath(licenseFile.path);
    debugPrint('ComPDFKit Init Result: $result');
  }

  Future<String> _getDocumentPath() async {
    var file = await CPDFFileUtil.extractAsset('pdfs/PDF_Document.pdf');
    return file.path;
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        resizeToAvoidBottomInset: false,
        appBar: AppBar(
          title: const Text('CPDFReaderWidget Example'),
        ),
        body: _document == null
        ? Container()
        : CPDFReaderWidget(
          document: _document!,
          configuration: CPDFConfiguration(),
          onCreated: (_create) => {})));
  }
}
```

2. Add the PDF documents you want to display in the project

- create a `pdf` directory

```
mkdir pdfs
```

- Copy your example document into the newly created `pdfs` directory and name it `PDF_Document.pdf`

3. Specify the `assets` directory in `pubspec.yaml`

```
 flutter:
+  assets:
+    - pdfs/
```

4. Launch your Android, iOS emulator, or connect your device.

```
flutter emulators --launch apple_ios_simulator
```

5. Run the app with:

```
flutter run
```

3 Guides

Welcome to the Developer Guide for ComPDFKit PDF SDK for Flutter. These guides will show you how to add document functionality to your Flutter applications.

If you're new to ComPDFKit, check out our Get Started guide to quickly add PDF viewing, annotating, and editing capabilities to your app.

3.1 Basic Operations

3.1.1 Overview

This section will demonstrate some of the most basic usage examples to help you get started quickly.

Guides

Apply the License Key

Initialize ComPDFKit PDF SDK with the License Key.

Open a Document

How to open a PDF from local storage.

Save Document

How to manually save and set the save callback function.

3.1.2 Apply the License Key

If you haven't get a license key, please check out how to obtain a license key.

ComPDFKit PDF SDK currently supports two authentication methods to verify license keys: online authentication and offline authentication.

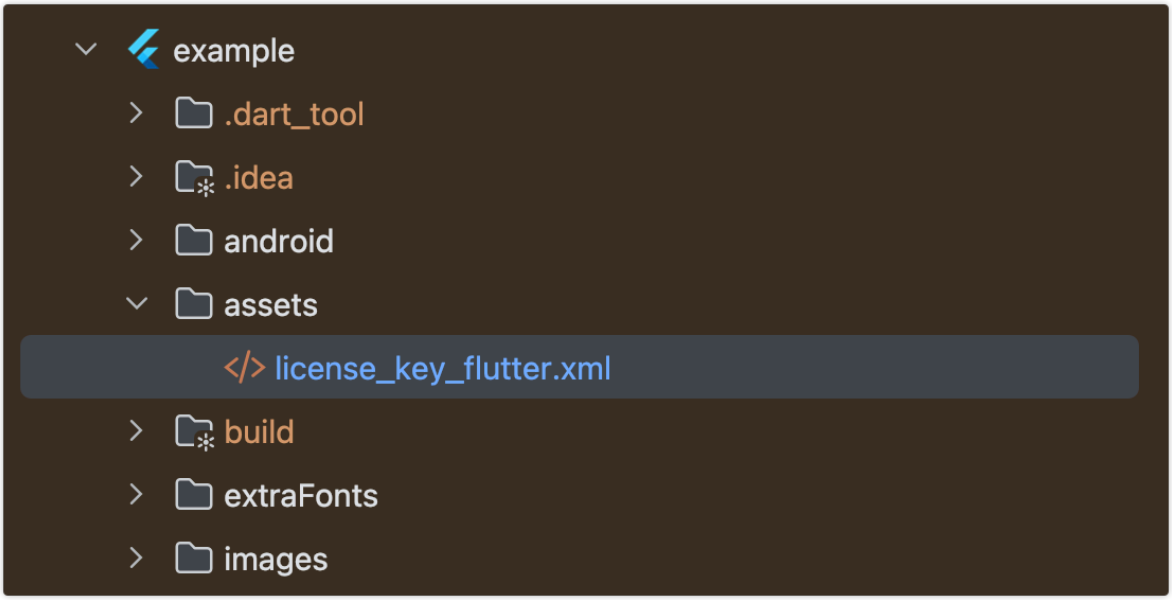
Learn about:

What is the authentication mechanism of ComPDFKit's license?

What are the differences between Online Authentication and Offline Authentication?

Accurately obtaining the license key is crucial for the application of the license.

1. In the email you received, locate the `XML` file containing the license key.
2. Copy the License.xml file into the `assets` directory of your Flutter project.



3. Open the pubspec.yaml file of your project and configure the flutter: section to enable the assets directory.

pubspec.yaml

Flutter commands

```
# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true
  generate: true
  assets:
    - assets/
    - pdfs/
    - images/
    - extraFonts/
```

4. Initialize the SDK:

```
// Include the license in Flutter assets and copy to device storage
// Add `license_key_flutter.xml` to your Flutter project's assets directory;
File licenseFile = await CPDFFileUtil.extractAsset(context,
'assets/license_key_flutter.xml');
await ComPDFKit.initWithPath(licenseFile.path);
```

Alternative methods (optional)

```
// Android
// Copy the license_key_flutter.xml file into the `android/app/src/main/assets` directory
of your Android project:
await ComPDFKit.initWithPath('assets://license_key_flutter.xml');

// iOS
// Copy the license_key_flutter.xml file into your iOS project directory (or a readable
location):
awai tComPDFKit.initWithPath('license_key_flutter.xml');
```

3.1.3 Open a Document

ComPDFKit PDF SDK for Flutter offers multiple ways to open PDF documents depending on your use case:

- Use `CPDFReaderWidget` to display an embedded reader UI.
- Use `ComPDFKit.openDocument()` to quickly launch a full-screen document viewer.
- Use `CPDFDocument` for background processing or non-UI operations.

Using CPDFReaderWidget (Recommended for embedded reader scenarios)

```
const String _documentPath = 'pdfs/PDF_Document.pdf';

File document = await CPDFFileUtil.extractAsset(_documentPath, shouldOverwrite: false);
...
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: document.path,
    configuration: CPDFConfiguration(),
    onCreated: (controller) {
      // Interact with controller if needed
    },
  ),
);
...
```

Using ComPDFKit.openDocument() (Quick full-screen viewer)

This is ideal for quickly launching a standalone document viewer.

```
const String _documentPath = 'pdfs/PDF_Document.pdf';

File document = await CPDFFileUtil.extractAsset(_documentPath);

ComPDFKit.openDocument(
  document.path,
  password: '',
  configuration: CPDFConfiguration(),
);
```

Using CPDFDocument (For background or logic-only use cases)

This is useful for non-UI operations such as annotation management, import/export, etc.

```
const String _documentPath = 'pdfs/PDF_Document.pdf';

File document = await CPDFFileUtil.extractAsset(context, _documentPath);
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
CPDFDocumentError error = await document.open(document.path);

if (error == CPDFDocumentError.success) {
  // Perform operations with the CPDFDocument instance...
}
```

3.1.4 Save Document

3.1.4.1 Manual save

When using the `CPDFReaderWidget` widget, documents are automatically saved during operations such as sharing, flattening, or adding watermarks. Additionally, you can manually save a document at any desired time using the `CPDFReaderWidgetController`. Here's an example:

```
late CPDFReaderWidgetController _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(
    leading: IconButton(
      onPressed: () {
        _save();
        Navigator.pop(context);
      },
      icon: const Icon(Icons.arrow_back)),
  ),
  body: CPDFReaderWidget(
    document: widget.documentPath,
    configuration: CPDFConfiguration(),
```

```
    onCreated: (controller) {
      setState(() {
        _controller = controller;
      });
    },
  ));

void _save() async {
  bool saveResult = await _controller.document.save();
  debugPrint('ComPDFKit-Flutter: saveResult:$saveResult');
}
```

3.1.4.2 Save as

The ComPDFKit SDK for Flutter supports saving a document using the "Save As" feature. After displaying the document with CPDFReaderWidget in Flutter, you can follow these steps to implement the "Save As" functionality:

- Open the document using `CPDFReaderWidget`
- When a custom Flutter button is pressed, call the `saveAs` API on `CPDFReaderWidgetController`

The code for this on Dart looks something like this:

```
late CPDFReaderWidgetController _controller;

@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomInset: false,
    appBar: AppBar(),
    body: Column(children: [
      TextButton(onPressed: () async{
        final tempDir = await ComPDFKit.getTemporaryDirectory();
        String savePath = '${tempDir.path}/temp/test_save_as.pdf';
        bool saveResult = await _controller.document.saveAs(savePath, removeSecurity:
false, fontSubSet: true);
      }, child: const Text('Save As')),
      Expanded(child: CPDFReaderWidget(
        document: widget.documentPath,
        configuration: CPDFConfiguration(),
        onCreated: (controller) {
          setState(() {
            _controller = controller;
          });
        },
      ))
    ],));
}
```

On the Android platform, the `saveAs()` method also supports saving to a specified directory via a Uri. As shown in the following example, the document will be saved to the `Downloads` folder in the device's public directory:

```
String? uri = await ComPDFKit.createUri('save_as.pdf');
if(uri != null){
  bool saveResult = await _controller.document.saveAs(uri, removeSecurity: false,
fontSubSet: true);
}
```

Parameter details:

Name	Type	Description
savePath	string	The target path for the "Save As" operation.
removeSecurity	bool	Whether to remove the document's password **Default**: false
fontSubSet	bool	Whether to include a font subset in the document. This parameter affects how the PDF displays in other software. If the required fonts are unavailable in other software, the text may not display properly. **Default**: true

3.1.4.3 Save Callback

When creating a `CPDFReaderWidget`, you can set a save listener to handle save events. For example:

```
CPDFReaderWidget(
  document: widget.documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {

  },
  onSaveCallback: (){
    debugPrint('CPDFDocument: save success');
  },
)
```

3.2 Viewer

3.2.1 Overview

ComPDFKit for Flutter provides a high-quality, fast, accurate, and feature-rich PDF viewer. It offers developers a quick and configurable way to integrate a PDF viewer into Flutter applications on Android and iOS platforms.

You can configure the page scrolling direction, scrolling mode, supported usage modes, and more through the CPDFConfiguration object.

Guides

ViewMode

How to set the default mode displayed when opening a PDF view, and configure switchable modes.

Display Modes

How to configure single page, double page, book, flip, or scroll direction, as well as cropping and viewing modes?

Zooming

How to set the initial zoom factor.

Themes

How to set a preset theme.

Page Navigation

How to navigate to a specified page number and get the current page number.

Highlight Form Fields and Hyperlinks

How to highlight form fields and hyperlink annotations.

Text Search and Selection

How to search for specific text in a PDF document using the API.

Render Image

How to use the API to render a specified PDF page as an image.

3.2.2 ViewMode

When you open a document using `CPDFReaderWidget` or `ComPDFKit.openDocument()`, you can set the default display mode according to your product's needs. For example, the default `Viewer Mode` allows viewing PDF documents and filling out forms but does not allow editing annotations, text, etc.

Setting the Default Mode

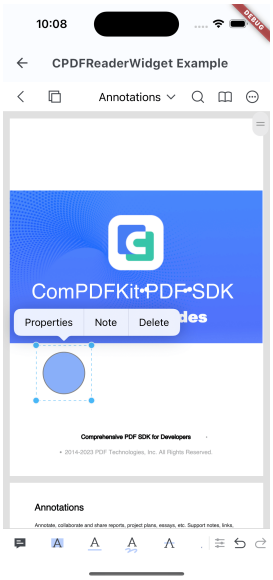
The following example demonstrates how to set Annotation Mode as the default display mode. In Annotation Mode, users can add, delete, and modify annotations.

```
// CPDFReaderWidget Sample
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(
    title: const Text('CPDFReaderWidget Example'),
  ),
  body: CPDFReaderWidget(
    document: documentPath,
```

```
    configuration: CPDFConfiguration(
      modeConfig: const ModeConfig(initialViewMode: CPreviewMode.annotations)
    ),
    onCreated: (controller) {
    },
  ));


// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(documentPath, '', CPDFConfiguration(
  modeConfig: const ModeConfig(initialViewMode: CPreviewMode.annotations)
));
```

The results are as follows:

Android	iOS
	

Setting the Mode List

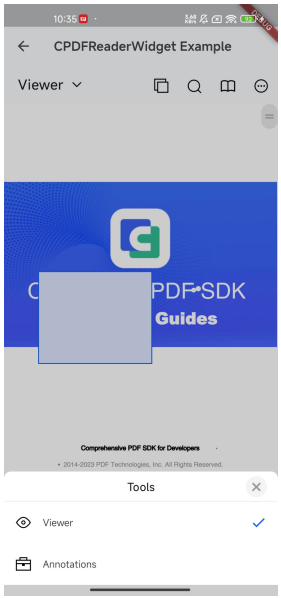
You can switch modes by clicking the top title. Depending on your needs, you can configure the required modes and their display order in `CPDFConfiguration`. The following example shows how to configure only Viewer Mode and Annotation Mode:

```
// CPDFReaderWidget Sample
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(
    title: const Text('CPDFReaderWidget Example'),
  ),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      modeConfig: const ModeConfig(availableViewModes: [
        CPreviewMode.viewer,
        CPreviewMode.annotations
```

```
      ])
    ),
    onCreated: (controller) {
    },
  ));


// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(documentPath, '', CPDFConfiguration(
  modeConfig: const ModeConfig(availableViewModes: [
    CPreviewMode.viewer,
    CPreviewMode.annotations
  ])
));
```

The results are as follows:

Android	iOS
	

Switch the View Mode:

When displaying a PDF using `CPDFReaderWidget`, you can switch the current view mode programmatically through the API provided by `CPDFReaderWidgetController`.

```
// Switch to annotation mode
controller.setPreviewMode(CPDFViewMode.annotations);

// Get the current view mode
CPDFViewMode viewMode = await controller.getPreviewMode();
```

3.2.3 Display Modes

Scroll direction

The page scroll direction can be either `horizontal` or `vertical`.

If `verticalMode` is `true`, it indicates `vertical` scrolling; otherwise, it's `horizontal` scrolling.

```
CPDFReaderWidgetController? _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      readerViewConfig: const ReaderViewConfig(verticalMode: true)),
    onCreated: (controller) {
      setState(() {
        this._controller = controller;
      });
    },
  ));
```

You can also set the scroll direction via `CPDFReaderWidgetController`:

```
_controller.setVerticalMode(true)
```

Display Mode

The page display mode can be `singlePage`, `doublePage`, or `coverPage`.

```
CPDFReaderWidgetController? _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      readerViewConfig: const ReaderViewConfig(displayMode: CPDFDisplayMode.doublePage)),
    onCreated: (controller) {
      setState(() {
        this._controller = controller;
      });
    },
  ));
```

Set the display mode through `CPDFReaderWidgetController`:

- singlePage

```
_controller.setDoublePageMode(false);
```

- doublePage

```
_controller.setDoublePageMode(true);
```

- coverPage

```
_controller.setCoverPageMode(true);
```

Scrolling Mode

The scrolling mode can be set to continuous scrolling or page flipping mode. When `continueMode` is `true`, it represents continuous scrolling; otherwise, it's page flipping scrolling.

```
CPDFReaderWidgetController? _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      readerViewConfig: const ReaderViewConfig(continueMode: true)),
    onCreated: (controller) {
      setState(() {
        this._controller = controller;
      });
    },
  ));
```

Setting scroll mode via `CPDFReaderWidgetController`:

```
_controller.setContinueMode(true);
```

Crop Mode

To display the document after cropping the blank areas around the PDF, when `cropMode` is `true`, it indicates enabling cropping mode; otherwise, it's not cropped.

```
CPDFReaderWidgetController? _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      readerViewConfig: const ReaderViewConfig(cropMode: true)),
```

```
    onCreated: (controller) {
      setState(() {
        this._controller = controller;
      });
    },
  ));
```

Set the crop mode through `CPDFReaderWidgetController`:

```
_controller.setCropMode(true);
```

3.2.4 Zooming

Set the default zoom factor for displaying the PDF document, which defaults to **1.0** and is limited to the range of **1.0** to **5.0**.

```
CPDFReaderWidgetController? _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      readerViewConfig: const ReaderViewConfig(pageScale: 1.0)),
    onCreated: (controller) {
      setState(() {
        this._controller = controller;
      });
    },
  ));
```

Setting zoom via `CPDFReaderWidgetController`:

```
_controller.setScale(1.0);
```

3.2.5 Themes

Theme refers to using different background colors to render PDF document pages when displaying PDF files to adapt to user preferences and scene needs, enhancing the reading experience.

When modifying the theme, only the visual effects during document display are altered, and it does not modify the PDF document data on the disk. The theme settings are not saved within the PDF document data.

This example shows how to set the dark theme:

```
CPDFReaderWidgetController? _controller;
```

```
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: CPDFConfiguration(
      readerViewConfig: const ReaderViewConfig(themes: CPDFThemes.dark)),
    onCreated: (controller) {
      setState(() {
        this._controller = controller;
      });
    },
  ));
```

Setting the background color through `CPDFReaderWidgetController`:

```
await _controller.setReadBackgroundColor(CPDFThemes.light);
```

Explanation of Themes

Mode	Description	Option Values
light	Uses a white background and black text, suitable for reading in well-lit environments.	CPDFThemes.light
dark	Uses a dark background and light text, suitable for reading in low-light environments.	CPDFThemes.dark
sepia	Use a beige background for users who are used to reading on paper.	CPDFThemes.sepia
reseda	Soft light green background reduces discomfort from high brightness and strong contrast when reading, effectively relieving visual fatigue.	CPDFThemes.reseda

3.2.6 Page Navigation

When using the `CPDFReaderWidget` component to display a PDF, the `CPDFReaderWidgetController` can be used to perform the following actions:

Navigate to a Specific Page

```
await _controller.setDisplayPageIndex(pageIndex);
```

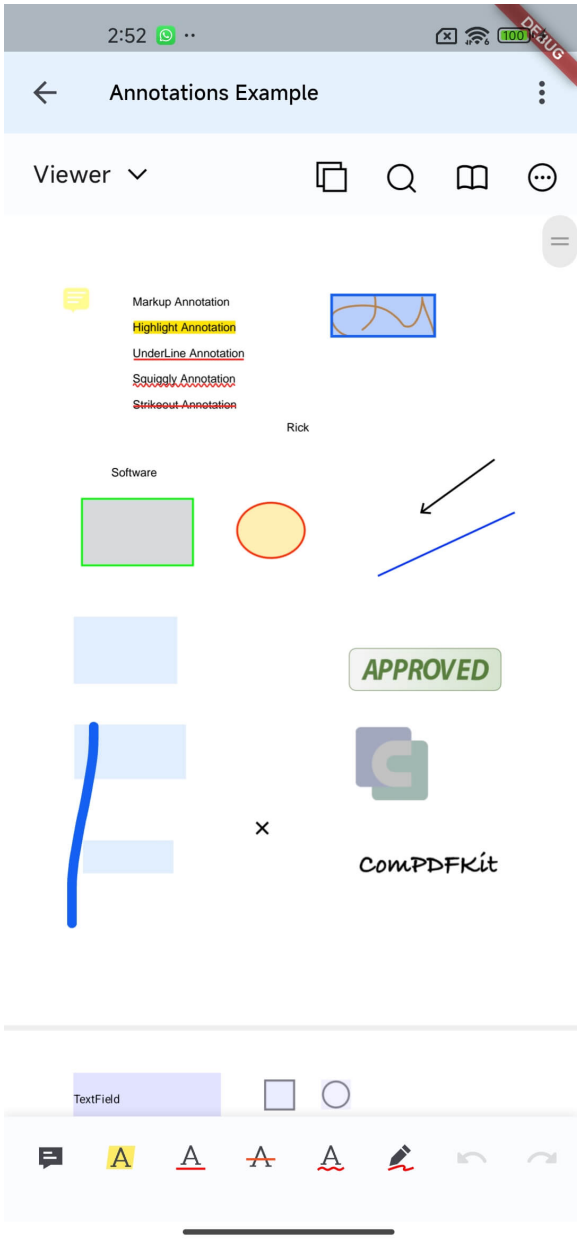
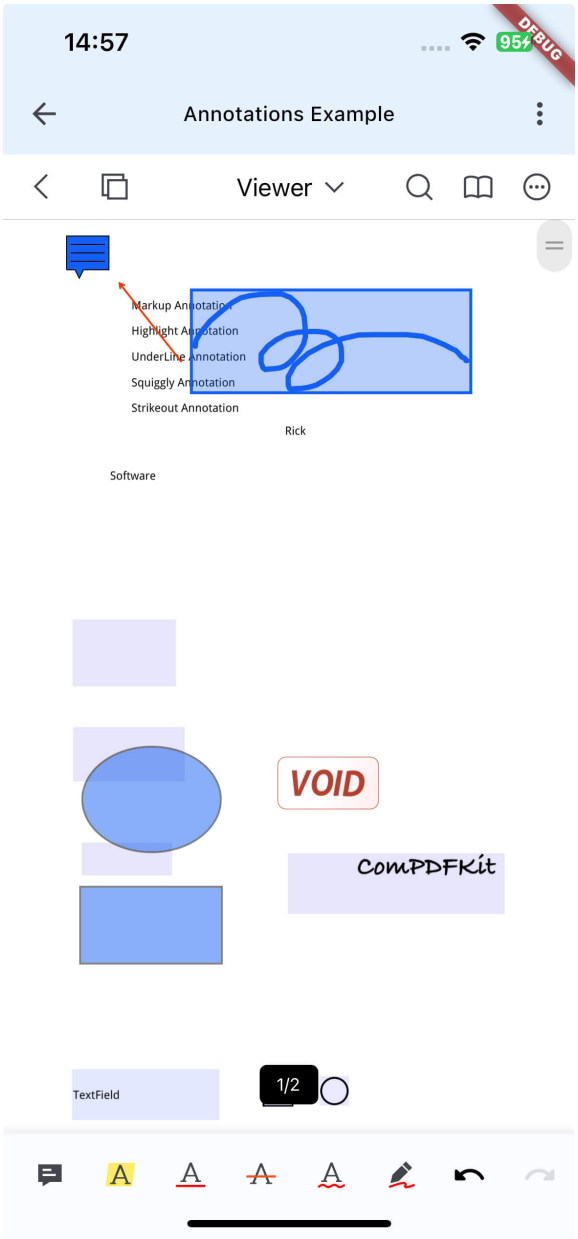
When you need to navigate to a specific page and simultaneously highlight certain content (for example, highlighting an annotation when clicked), you can use the rectList parameter of setDisplayPageIndex to draw rectangles on the page.

The following example shows how to jump to the page containing an annotation and highlight the annotation area with a rectangle:

```
final pageIndex = 0;
CPDFPage page = controller.document.pageAtIndex(pageIndex);
final pageAnnotations = await page.getAnnotations();
final annotation = pageAnnotations[0];
await controller.setDisplayPageIndex(pageIndex: annotation.page, rectList:
[annotation.rect]);

// Clear highlight
await controller.clearDisplayRect();
```

Example Result:

Android	iOS
	

Get the Current Page Index

```
int currentPageIndex = await _controller.getCurrentPageIndex();
```

You can also set a page number listener when using `CPDFReaderWidget` to get the current sliding page number in real time.

```
CPDFReaderWidget(
          document: widget.documentPath,
          configuration: CPDFConfiguration(),
          onCreated: (controller) {

          },
          onPageChanged: (pageIndex){
            debugPrint('pageIndex:${pageIndex}');
          },
        )
```

3.2.7 Highlight Form Fields and Hyperlinks

You can use `CPDFConfiguration` or `CPDFReaderWidgetController` to set the form and hyperlink annotations in the current document to be highlighted.

The highlight feature for PDF form fields helps users quickly locate and fill out forms, significantly enhancing efficiency in scenarios involving extensive form completion.

The highlight feature for hyperlinks allows users to add hyperlinks to crucial information within a PDF document, facilitating other users in swiftly locating and comprehending information. This not only improves the readability of the PDF document but also enhances its interactivity.

This example shows how to highlight form fields and hyperlinks:

```
late CPDFReaderWidgetController _controller;

/// CPDFConfiguration
CPDFReaderWidget(
  document: widget.documentPath,
  configuration: CPDFConfiguration(
    readerViewConfig: CPDFReaderViewConfig(linkHighlight: true,formFieldHighlight: true)),
  onCreated: (controller) {
    setState(() {
      _controller = controller;
    });
  },
);

/// Enable hyperlink highlighting.
await _controller.setLinkHighlight(true);
/// Enable form field highlighting.
await _controller.setFormFieldHighlight(true);
```

3.2.8 Text Search and Selection

In the Flutter SDK, ComPDF provides the CPDFTextSearcher API to search for specific text in PDF documents, highlight results, and retrieve contextual text snippets.

1. Get a `CPDFTextSearcher` Instance

To perform a search, obtain a `CPDFTextSearcher` object from the `CPDFDocument`:

```
late CPDFReaderWidgetController _controller;

Scaffold(
  resizeToAvoidBottomInset: false,
  body: CPDFReaderWidget(
    document: widget.documentPath,
    configuration: CPDFConfiguration(),
    onCreated: (controller) {
      setState(() {
        _controller = controller;
      });
    },
  ));


CPDFTextSearcher? textSearcher = _controller?.document.getTextSearcher();
```

2. Text Search

Use the `searchText` method to search for keywords in the PDF document. You can specify search options such as case sensitivity or whole-word matching.

```
// Search for keywords in the PDF document
List<CPDFTextRange> results = await searcher.searchText(
  'keywords',
  searchOptions: CPDFSearchOptions.caseInsensitive, // Search option
);
```

> The search results are returned as a list of `CPDFTextRange` objects, each representing the position of the keyword in the document.

Explanation of Search Settings

Enum Value	Description
CPDFSearchOptions.caseInsensitive	Case-insensitive search (default)
CPDFSearchOptions.caseSensitive	Case sensitive search
CPDFSearchOptions.matchWholeWord	Matches a whole word only

3. Select a Search Result

You can highlight a specific result in the document using the `selection` method:

```
// Select the first search result
if (results.isNotEmpty) {
  CPDFTextRange range = results[0];
  await searcher.selection(range);
}
```

4. Clear Search Results

To reset the search state and clear results:

```
await searcher.clearSearch();
```

5. Retrieve Contextual Text

With `CPDFPage.getText`, you can extract a snippet of text for a given range and optionally expand the range to include surrounding context.

```
// Assume we already have a search result
CPDFTextRange range = results[0];

// Expand the range to include 20 characters before and after
final expandedRange = range.expanded(before: 20, after: 20);

// Retrieve contextual text from the corresponding page
final contextText = await page.getText(expandedRange);
print("Text with context: $contextText");
```

Set Search Highlight Style

In PDF search, the SDK allows customizing how search results are highlighted via CPDFConfiguration.

Developers can configure:

- **Normal match results (normalKeyword)**: Displays all candidate matches. *Only supported on Android.*
- **Current focused result (focusKeyword)**: Highlights the search result the user is currently navigating to.

Example Code

```
CPDFConfiguration(globalConfig: const CPDFGlobalConfig(
  search: CPDFSearchConfig(
    normalKeyword: CPDFKeywordConfig(
      borderColor: Colors.transparent,
      fillColor: Color(0x77FFFF00),
    ),
    focusKeyword: CPDFKeywordConfig(
      borderColor: Colors.transparent,
      fillColor: Color(0xCCFD7338),
    ),
  )
))
```

If the top toolbar in CPDFReaderWidget is hidden, you can show or hide the search view via the API:

```
// Show search view
await controller.showTextSearchView();

// Hide search view
await controller.hideTextSearchView();
```

3.2.9 Render Image

This API allows rendering a specified PDF page as an image. It is useful for generating previews or exporting page content.

The returned data type is Uint8List, which can be used directly for display or saving as an image file.

Method Declaration

```
Future<Uint8List> renderPage({
  required int pageIndex,
  required int width,
  required int height,
  Color backgroundColor = Colors.white,
  bool drawAnnot = true,
  bool drawForm = true
})
```

Parameters

Parameter	Type	Required	Description
pageIndex	int	Yes	The index of the page to render, starting from 0
width	int	Yes	Width of the rendered image in pixels
height	int	Yes	Height of the rendered image in pixels
backgroundColor	Color	No	Page background color, default is white, Android only
drawAnnot	bool	No	Whether to render annotations, default true, Android only
drawForm	bool	No	Whether to render form fields, default true, Android only

Example Usage

```
final pageImage = await controller.renderPage(
  pageIndex: 0,
  width: 1080,
  height: 1920,
  backgroundColor: Colors.white,
  drawAnnot: true,
  drawForm: true,
);

// Display the image
Image.memory(pageImage);
```

3.3 Annotations

3.3.1 Overview

The ComPDFKit Flutter SDK offers functions for importing and exporting annotations, allowing you to manage annotations in your documents. We are continually enhancing these features to cater to a wider range of use cases.

Guides

Access annotations

Get a list of annotations and individual annotation objects within a document.

Create Annotation

Create various types of annotations through `CPDFReaderWidget`.

Import and Export

Managing comments in XFDF files.

Delete Annotations

How to delete all comments in the currently displayed document.

Flatten Annotations

Permanently save the existing annotations in the document as images.

Annotation History Management

Used to manage the editing history of annotations in the current document, providing undo and redo functions.

3.3.2 Access Annotations

To retrieve all annotations from a PDF document, follow the steps below:

Basic Steps:

1. Obtain a document object (either from a CPDFReaderWidget controller or a CPDFDocument instance).
2. Get the total number of pages to iterate through each one.
3. Retrieve each page object (CPDFPage).
4. Retrieve the list of annotations from each page.
5. Iterate through and collect all annotations across all pages.

Using `CPDFReaderWidget` to Access Annotations

This is suitable for scenarios where you're integrating with Flutter's UI component. First, obtain a CPDFReaderWidgetController via the onCreated callback, then access the document object to read pages and annotations.

```
CPDFReaderWidgetController? _controller;

// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: widget.documentPath,
  password: widget.password,
  configuration: widget.configuration,
  onCreated: (controller) {
    setState(() {
      _controller = controller;
    });
  },
);

// Retrieve the annotation list
List<CPDFAnnotation> annotations = [];

if (_controller != null) {
  int pageCount = await _controller!.document.getPageCount();

  for (int i = 0; i < pageCount; i++) {
```

```
    CPDFPage page = _controller!.document.pageAtIndex(i);
    List<CPDFAnnotation> pageAnnotations = await page.getAnnotations();
    annotations.addAll(pageAnnotations);
  }
}
```

Using `CPDFDocument` to Access Annotations

This is ideal for background or logic-only use cases where no UI component is needed.

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
CPDFDocumentError error = await document.open(pdfFilePath);

if (error == CPDFDocumentError.success) {
  List<CPDFAnnotation> annotations = [];

  int pageCount = await document.getPageCount();

  for (int i = 0; i < pageCount; i++) {
    CPDFPage page = document.pageAtIndex(i);
    List<CPDFAnnotation> pageAnnotations = await page.getAnnotations();
    annotations.addAll(pageAnnotations);
  }

  // 'annotations' now contains all annotations from all pages in the document
}
```

3.3.3 Create Annotations

ComPDFKit supports a wide variety of annotation types, including text notes, links, shapes, highlights, stamps, freehand drawings, and audio annotations, fully meeting diverse annotation requirements.

Creating Annotations

With CPDFReaderView, users can enter annotation mode and create annotations via touch interaction. The annotation type can be set using the setAnnotationMode() method.

```
CPDFReaderWidgetController? _controller;

// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: widget.documentPath,
  password: widget.password,
  configuration: widget.configuration,
  onCreated: (controller) {
    setState(() {
      _controller = controller;
    });
  },
);
```

```
await _controller?.setAnnotationMode(CPDFAnnotationType.highlight);
```

The CPDFAnnotationType enum supports the following annotation types:

Annotation Type	Enum Value
Text Note	note
Highlight	highlight
Underline	underline
Squiggly	squiggly
Strikeout	strikeout
Ink	ink
Ink Eraser	ink_eraser
Pencil	pencil
Circle	circle
Rectangle	square
Arrow	arrow
Line	line
Free Text	freetext
Signature	signature
Stamp	stamp
Link	link
Sound	sound
Exit Drawing Mode	unknown

Exiting Annotation Mode

To exit annotation mode after finishing the operation, call:

```
await _controller?.setAnnotationMode(CPDFAnnotationType.unknown);
```

Get Current Annotation Type

You can retrieve the current annotation type to check whether you are in annotation mode or determine the selected tool:

```
CPDFAnnotationType currentType = await _controller?.getAnnotationMode();
```

3.3.4 Import and Export

XFDF is an XML-based standard derived from Adobe XFDF, used for encoding annotations and form field values. It is compatible with Adobe Acrobat and various third-party frameworks.

The ComPDFKit Flutter SDK supports reading and writing annotations in XFDF format, enabling import and export of annotation data. The following guide demonstrates how to use XFDF to import and export annotations.

Importing Annotations

You can import annotations from an XFDF file into a PDF document by calling the `importAnnotations(xfdfFile)` method. This allows you to apply XFDF-formatted annotation data to an existing PDF.

- **xfdfFile**: The path of the XFDF file to import.

  On Android, xfdfFile can be:

  - A file URI (e.g., content:// format)
  - A file path (e.g., /storage/emulated/0/Download/annotations.xfdf)
  - A file located in the assets directory

Example:

Using `CPDFReaderWidget`:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
// Import annotations
bool result = await _controller.document.importAnnotations(xfdfFile);
```

Using `CPDFDocument`:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Import annotations
  bool result = await document.importAnnotations(xfdfFile);
}
```

Exporting Annotations

To export annotations from a PDF document in XFDF format, use the exportAnnotations() method. This is useful for saving annotation data or sharing it with other systems.

- This method does not require any path parameters. The system will automatically generate and return the file path of the exported XFDF file.

Example:

Using `CPDFReaderWidget`:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
// Export annotations
String xfdfFilePath = await _controller.document.exportAnnotations();
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
CPDFDocumentError error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Export annotations
  String xfdfFilePath = await document.exportAnnotations();
}
```

3.3.5 Delete Annotations

The ComPDFKit Flutter SDK supports deleting selected annotations through APIs. The steps to delete annotations are as follows:

1. Obtain the document object (either the controller of `CPDFReaderWidget` or an instance of `CPDFDocument`).

2. Get the page object that contains the annotation to delete.
3. Retrieve the list of annotations on that page.
4. Locate the annotation you want to delete from the list.
5. Delete the annotation.

Example:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
// Delete the first annotation on the first page
CPDFPage page = _controller.document.pageAtIndex(0);
var pageAnnotations = await page.getAnnotations();
await page.removeAnnotation(pageAnnotations[0]);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Delete the first annotation on the first page
  CPDFPage page = document.pageAtIndex(0);
  var pageAnnotations = await page.getAnnotations();
  await page.removeAnnotation(pageAnnotations[0]);
}
```

Alternatively, you can delete **all annotations** in the current document by calling the removeAllAnnotations() method.

- The return value is a boolean indicating whether the operation was successful.

Example:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
// Delete all annotations
bool deleteResult = await _controller.document.removeAllAnnotations();
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Delete all annotations
  bool deleteResult = await document.removeAllAnnotations();
}
```

> **Note**: This method does not remove link annotations.

3.3.6 Flatten Annotations

Annotation flattening refers to the process of converting editable annotations into non-editable, unmodifiable static images or plain text. When flattening annotations, **all editable elements** in the document (including annotations and form fields) are flattened, so annotation flattening is also known as **document flattening**.

Below is an example of how to flatten annotations:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
```

```
final savePath = 'file:///storage/emulated/0/Download/flatten.pdf';
// Or use a Uri on Android:
// final savePath = await ComPDFKit.createUri('flatten_test.pdf', 'compdfkit',
'application/pdf');
final fontSubset = true;
final result = await _controller.document.flattenAllPages(savePath, fontSubset);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  final savePath = 'file:///storage/emulated/0/Download/flatten.pdf';
  // Or use a Uri on Android:
  // final savePath = await ComPDFKit.createUri('flatten_test.pdf', 'compdfkit',
'application/pdf');
  final fontSubset = true;
  final result = await document.flattenAllPages(savePath, fontSubset);
}
```

What is Document Flattening

Document flattening refers to the process of converting editable elements, such as annotations, form fields, or layers, in a PDF document into non-editable, static images, or pure text. The purpose of this process is to lock the final state of the document, eliminating editable elements.

Document flattening is typically applied in the following contexts:

1. **Content Protection:** Flattening can be used to protect document content, ensuring that the document remains unaltered during distribution or sharing. This is crucial for maintaining document integrity and confidentiality.
2. **Form Submission:** In form processing, flattening can convert user-filled form fields and annotations into static images for easy transmission, archiving, or printing, while preventing modifications to the form content in subsequent stages.
3. **Compatibility and Display:** Some PDF readers or browsers may encounter issues with displaying and interacting with PDF documents that contain numerous annotations or layers. Document flattening helps address these compatibility issues, enhancing the visual representation of documents in various environments.
4. **File Size Reduction:** Flattened documents typically have reduced file sizes since editable elements are converted into static images or text, eliminating the need to store additional data for editing information.

3.3.7 Annotation History Management

The CPDFAnnotationHistoryManager class manages the annotation editing history within the current document. It provides undo and redo functionality, allowing users to revert or restore actions such as adding, modifying, or deleting annotations.

Accessing the Annotation History Manager

You can obtain an instance of CPDFAnnotationHistoryManager through the CPDFReaderWidgetController:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
);

CPDFAnnotationHistoryManager historyManager = _controller.annotationHistoryManager;
```

Setting a History Change Listener

Use setOnHistoryChangedListener() to set a callback that will be triggered whenever the annotation history state changes (e.g., after a new annotation is added or an undo action is performed):

```
historyManager.setOnHistoryChangedListener((canUndo, canRedo) {
  print('Can Undo: $canUndo, Can Redo: $canRedo');
  // Update undo/redo button states accordingly
});
```

Checking Whether Undo or Redo Is Available

You can check whether there are actions that can be undone or redone using the following methods:

```
bool canUndo = await historyManager.canUndo();
bool canRedo = await historyManager.canRedo();
```

These methods return boolean values indicating whether corresponding history actions are available.

Performing Undo and Redo Operations

If there are valid undo or redo actions available, you can perform them as follows:

```
await historyManager.undo(); // Undo the last annotation action
await historyManager.redo(); // Redo the last undone annotation action
```

3.4 Forms

3.4.1 Overview

The ComPDFKit Flutter SDK supports managing form-related data via API.

Guide

Access Forms

How to retrieve form data from a PDF document.

Create Form Fields

How to switch to the form creation mode to create form fields.

Delete Form Fields

Delete specific form fields.

Import and Export

Export form data as XFDF format files for management.

3.4.2 Supported Form Field Types

ComPDFKit supports various form field types compliant with the PDF standard, which are readable and writable by various programs, including Adobe Acrobat and other PDF processors that adhere to the standard. The supported form field types are as follows:

Type	Description	Class Name
Check Box	Select one or more options from predefined choices	CPDFCheckBoxWidget
Radio Button	Select a single option from a predefined set	CPDFRadioButtonWidget
Push Button	Create a custom button that performs an action	CPDFPushButtonWidget
List Box	Select one or more options from a list	CPDFListboxWidget
Combo Box	Select one option from a dropdown list of text values	CPDFComboBoxWidget
Text Field	Enter text such as name, address, or email	CPDFTextWidget
Signature Field	Digitally or electronically sign the PDF document	CPDFSignatureWidget

3.4.3 Access Forms

The steps to retrieve the form field list and form field objects are as follows:

1. Obtain the document object (either from the `CPDFReaderWidget` controller or a `CPDFDocument` instance).
2. Get the page object.
3. Retrieve the list of form fields from each page.

Here's the corresponding code:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
```

```
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
// Retrieve all form fields in the document
List<CPDFWidget> widgets = [];
int pageCount = await _controller.document.getPageCount();
for (int i = 0; i < pageCount; i++) {
  CPDFPage page = _controller.document.pageAtIndex(i);
  var pageWidgets = await page.getWidgets();
  widgets.addAll(pageWidgets);
}
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Retrieve all form fields in the document
  List<CPDFWidget> widgets = [];
  int pageCount = await document.getPageCount();
  for (int i = 0; i < pageCount; i++) {
    CPDFPage page = document.pageAtIndex(i);
    var pageWidgets = await page.getWidgets();
    widgets.addAll(pageWidgets);
  }
}
```

3.4.4 Create Form Fields

Create Form Fields

Using `CPDFReaderWidget`, users can enter form mode and add form fields via touch interactions.

```
CPDFReaderWidgetController? _controller;

// Initialize CPDFReaderWidget and obtain the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(
    modeConfig: const CPDFModeConfig(
      initialViewMode: CPDFViewMode.forms)),
  onCreated: (controller) {

  });

await controller.setFormCreationMode(CPDFFormType.textField);
```

Exit Creation Mode

```
await controller.exitFormCreationMode();
```

Supported Form Field Types

Type
textField
checkBox
radioButton
listBox
comboBox
signaturesFields
pushButton

3.4.5 Delete Form Fields

The ComPDFKit Flutter SDK supports deleting selected form fields via API. The steps to remove a form field are as follows:

1. Obtain the document object (either from the `CPDFReaderWidget` controller or a `CPDFDocument` instance).
2. Get the page object where the form field to be deleted is located.
3. Retrieve the list of form fields on that page.
4. Locate the form field you want to delete from the list.
5. Delete the form field.

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)
// Remove the first form field on the first page
CPDFPage page = _controller.document.pageAtIndex(0);
var pageWidgets = await page.getWidgets();
await page.removeWidget(pageWidgets[0]);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Remove the first form field on the first page
  CPDFPage page = document.pageAtIndex(0);
  var pageWidgets = await page.getWidgets();
  await page.removeWidget(pageWidgets[0]);
}
```

3.4.6 Import and Export

The methods for importing and exporting XFDF form data allow users to save and restore PDF form data, making it easier to fill out and manage form inputs.

Import Form

Here is the code to import XFDF form data:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
```

```
)

// Import from file path
bool importResult = await
_controller.document.importWidgets('data/your_package_name/files/xxx.xfdf');

// Or use a content URI on Android
const xfdfFile = 'content://media/external/file/1000045118';
const importResult = await _controller.document.importWidgets(xfdfFile);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  // Import from file path
  bool importResult = await
document.importWidgets('data/your_package_name/files/xxx.xfdf');

  // Or use a content URI on Android
  const xfdfFile = 'content://media/external/file/1000045118';
  const importResult = await document.importWidgets(xfdfFile);
}
```

Export Form

Here is the code to export form data:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)

// Export to file path
const exportPath = await _controller.document.exportWidgets();
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  const exportPath = await document.exportWidgets();
}
```

What is XFDF

XFDF (XML Forms Data Format) is an XML-based format used to describe and transfer PDF form data. It is commonly used alongside PDF files to store and share values, states, and actions of form fields.

An XFDF file contains the data for the corresponding PDF form, including field names, values, options, and formats.

Note: XFDF is a data-only format—it does not contain the actual PDF file itself. It is used solely to represent and transmit form data for integration and interoperability across different systems and applications.

3.5 Document Editor

3.5.1 Overview

The document editing feature provides a range of page manipulation capabilities, allowing users to control the structure of the document and adjust its layout and formatting. This ensures the content is presented in a logical and precise manner.

Guide

Insert Pages

Insert blank pages or pages from another document into the target document.

Split Pages

Split a portion of a multi-page document into a separate standalone document.

3.5.2 Insert Pages

You can insert blank pages or pages from another PDF into the target document.

Insert a Blank Pages

Here is an example of inserting a blank page:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
```

```
    setState(() {
      this._controller = controller;
    });
  },
)

CPDFPageSize pageSize = CPDFPageSize.a4;
// Or use a custom page size:
// CPDFPageSize.custom(500, 800);
bool insertResult = await _controller.document.insertBlankPage(pageIndex: 0, pageSize =
pageSize);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  CPDFPageSize pageSize = CPDFPageSize.a4;
  // Or use a custom page size:
  // CPDFPageSize.custom(500, 800);
  bool insertResult = await document.insertBlankPage(pageIndex: 0, pageSize = pageSize);
}
```

Insert Pages from Another PDF

Here is an example of inserting pages from another PDF document:

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)

final filePath = '/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';
final pages = [0, 1, 2]; // Pages to import from the source document
final insertPosition = 0; // Insert at the beginning of the document
final result = await _controller.document.importDocument(
  filePath: filePath,
  pages: pages,
  insertPosition: insertPosition,
);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  final filePath =
'/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';
  final pages = [0, 1, 2]; // Pages to import from the source document
  final insertPosition = 0; // Insert at the beginning of the document
  final result = await document.importDocument(
    filePath: filePath,
    pages: pages,
    insertPosition: insertPosition,
  );
}
```

3.5.3 Split Pages

Page splitting allows you to extract specific pages from a document and reorganize them into a new PDF. The steps are as follows:

1. Specify the page indices to split.
2. Set the output path for the new PDF document.

Using CPDFReaderWidget:

```
CPDFReaderWidgetController? _controller;
// Initialize CPDFReaderWidget and get the controller in the onCreated callback
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(),
  onCreated: (controller) {
    setState(() {
      this._controller = controller;
    });
  },
)

var pages = [0, 1, 2];
var savePath = '/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';
// On Android, you can use ComPDFKit.createUri() to generate a Uri
// var savePath = ComPDFKit.createUri('split_document.pdf');
bool splitResult = await _controller.document.splitDocumentPages(savePath, pages);
```

Using CPDFDocument:

```
// Create and open the document
CPDFDocument document = await CPDFDocument.createInstance();
var error = await document.open(pdfFilePath);
if (error == CPDFDocumentError.success) {
  var pages = [0, 1, 2];
  var savePath = '/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';
  // On Android, you can use ComPDFKit.createUri() to generate a Uri
  // var savePath = ComPDFKit.createUri('split_document.pdf');
  bool splitResult = await document.splitDocumentPages(savePath, pages);
}
```

3.6 Security

3.6.1 Overview

The security module provides features including password protection, permission settings, Bates coding, background, and page header and footer functionalities. Document security is guaranteed by managing document passwords and permissions, and by adding logos and copyright information.

Key Capabilities

- **Password Management:** Set, modify, or delete document passwords and permissions.
- **Encryption Standards:** Support for standard PDF security handlers (40-bit and 128-bit RC4 encryption), as well as 128-bit and 256-bit AES (Advanced Encryption Standard) encryption.

Guides

Set Password

By managing document passwords and permission settings, unauthorized access is prevented, and user operations on the document can be controlled.

Remove Password

How to remove a document's password and permissions via the API.

3.6.2 Set Password

The PDF permission module ensures document security by providing encryption, document permissions, decryption, and password removal functionalities, allowing users to securely control and manage documents.

Encrypt

Encrypt function consists of two parts: User Password and Owner Password.

The User Password is utilized to open the document, ensuring that only authorized users can access its content. When a user password is set, it typically restricts certain document permissions such as modification, copying, or printing. On the other hand, the Owner Password not only opens the document but also unlocks all restricted permissions, allowing users to modify, copy, or print the document. The dual-

password system aims to provide a more flexible and secure approach to document access and management.

ComPDFKit offers a variety of encryption algorithms and permission settings. Depending on your requirements, you can use the appropriate algorithm and configure custom permissions to safeguard your document.

The steps to encrypt are as follows:

1. Set distinct user passwords and owner passwords.
2. Create a permissions information class.
3. Specify the encryption algorithm.
4. Encrypt the document using the user password, owner password, permission information, and the chosen algorithm.
5. Save the document

This sample shows how to encrypt a document:

```
CPDFReaderWidgetController? _controller;

@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomInset: false,
    appBar: AppBar(),
    body: Column(children: [
      TextButton(onPressed: () async{
        // set password
        bool setPasswordResult = await _controller.document.setPassword(
          userPassword: '1234',
          ownerPassword: '12345',
          allowsPrinting: false,
          allowsCopying: false,
          encryptAlgo: CPDFDocumentEncryptAlgo.aes256);
        debugPrint('ComPDFKit-Flutter: setPasswordResult:$setPasswordResult');
        // save pdf
        await _controller.document.save();
      }, child: const Text('Encrypt PDF')),
      Expanded(child: CPDFReaderWidget(
        document: widget.documentPath,
        configuration: CPDFConfiguration(),
        onCreated: (controller) {
          setState(() {
            _controller = controller;
          });
        },
      ))
    ],));
}
```

Different Encryption Algorithms:

Algorithm	Description	Enum Value
No Encrypt Algo	No encryption	CPDFDocumentEncryptAlgo.noEncryptAlgo
RC4	Encrypts plain text using key-based XOR	CPDFDocumentEncryptAlgo.rc4
AES-128	AES encryption using a 128-bit key	CPDFDocumentEncryptAlgo.aes128
AES-256	AES encryption using a 256-bit key	CPDFDocumentEncryptAlgo.aes256

3.6.3 Remove Password

Removing the password refers to the process where a user with owner permissions removes both the owner and user passwords, allowing the document to be opened without a password and defaulting to full access without restrictions.

Steps to remove a password:

1. Open the document using the `CPDFReaderWidget` component.
2. Unlock the document and gain full permissions.
3. Remove the password from the unlocked document.

Here is an example of how to remove a password:

```
late CPDFReaderWidgetController _controller;

@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomInset: false,
    appBar: AppBar(),
    body: Column(children: [
      TextButton(onPressed: () async{

        bool removePasswordResult = await _controller.document.removePassword();

      }, child: const Text('Encrypt PDF')),
      Expanded(child: CPDFReaderWidget(
        document: widget.documentPath,
        configuration: CPDFConfiguration(),
        onCreated: (controller) {
          setState(() {
            _controller = controller;
          });
        },
      ))
    ],));
}
```

For easy customization of UI for setting and removing passwords in Flutter, the following APIs are available:

Check if Document is Encrypted:

```
bool isEncrypted = await controller.isEncrypted();
```

Get Document Permissions Status:

```
CPDFDocumentPermissions permissions = await controller.getPermissions();
```

Different Permission States:

Permission State	Description	Enum Value
None	No permissions are applied to the document	`CPDFDocumentPermissions.none`
User	Document opened with user password; may have printing and copying restrictions	`CPDFDocumentPermissions.user`
Owner	Document opened with owner password; no restrictions	`CPDFDocumentPermissions.owner`

Check Owner Permissions:

```
// Check if the current document is unlocked with owner permissions
bool unlocked = await controller.checkOwnerUnlocked();

// Check if the owner password is correct
bool result = await controller.checkOwnerPassword('owner_password');
```

3.7 Watermark

3.7.1 Overview

A watermark is a mark placed on a PDF document, usually in the form of semi-transparent text or image elements. By adding a watermark, you can highlight ownership or other relevant information without interfering with the readability of the document.

Advantages of ComPDFKit Watermarks

- **Copyright Marking**: Display document source and copyright information through watermarks to prevent screenshots or misuse.
- **Branding**: Customize watermark appearance to showcase your brand's logo or other preset content.
- **Quick UI Integration**: Easily integrate and customize watermarks using scalable UI components.

ComPDFKit Watermark Features

- Add Text Watermark

  Use the API to add a custom text watermark, or set the text watermark interactively via the popup.
- Add Image Watermark

Use the API to add an image watermark, or select and configure the image watermark interactively via the popup.

- **Remove Watermark**

  Remove watermarks from the PDF document.

3.7.2 Add Text Watermark

ComPDFKit supports **two ways** to add text watermarks:

1. **Programmatic Approach (API Call)**

   Use document.createWatermark to create a text watermark. This method is suitable for scenarios where watermarks need to be generated automatically in code.

```
late CPDFReaderWidgetController _controller;

@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomInset: false,
    appBar: AppBar(),
    body: Column(children: [
      TextButton(onPressed: () async{
        await _controller.document.createWatermark(CPDFWatermark.text(
          textContent: 'ComPDFKit',
          scale: 1.0,
          fontSize: 60,
          rotation: 0,
          horizontalAlignment: CPDFWatermarkHorizontalAlignment.center,
          verticalAlignment: CPDFWatermarkVerticalAlignment.center,
          textColor: Colors.red,
          pages: [0, 1, 2, 3]));
      }, child: const Text('Add Text Watermark')),
      Expanded(child: CPDFReaderWidget(
        document: widget.documentPath,
        configuration: CPDFConfiguration(),
        onCreated: (controller) {
          setState(() {
            _controller = controller;
          });
        },
      ))
    ],));
}
```

2. **UI Popup Approach**

- Configure default watermark properties in CPDFConfiguration, such as default text, color, font size.
- Call controller.showAddWatermarkView() to open the interactive watermark popup.

```
// Set default text watermark in configuration
final configuration = CPDFConfiguration(
  globalConfig: const CPDFGlobalConfig(
    watermark: CPDFWatermarkConfig(
      text: 'ComPDFKit-Flutter',
      textColor: Colors.red,
      textSize: 36,
      opacity: 120,
      rotation: -45
    )
  )
);

// Open watermark popup
controller.showAddWatermarkView(
  config: CPDFGlobalConfig(
    watermark: CPDFWatermarkConfig(
      text: 'ComPDFKit-Flutter',
      textColor: Colors.red,
      textSize: 36,
      opacity: 120,
      rotation: -45
    )
  )
);
```

3.7.3 Add Image Watermark

ComPDFKit supports **two ways** to add image watermarks:

1. **Programmatic Approach (API Call)**

```
late CPDFReaderWidgetController _controller;

@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomInset: false,
    appBar: AppBar(),
    body: Column(children: [
      TextButton(onPressed: () async{
        // image file path
        // android platform support Uri format.
        // content://xxxx
        File imageFile = await extractAsset(context, 'images/logo.png');
        await controller.document.createWatermark(CPDFWatermark.image(
          imagePath: imageFile.path,
          opacity: 1,
          rotation: 45,
          pages: [0, 1, 2, 3],
```

```
          horizontalAlignment: CPDFWatermarkHorizontalAlignment.center,
          verticalAlignment: CPDFWatermarkVerticalAlignment.center,
        ));
      }, child: const Text('Add Image Watermark')),
      Expanded(child: CPDFReaderWidget(
        document: widget.documentPath,
        configuration: CPDFConfiguration(),
        onCreated: (controller) {
          setState(() {
            _controller = controller;
          });
        },
      ))
    ],));
}
```

2. **UI Popup Approach**

- Configure default watermark properties in CPDFConfiguration, such as image, opacity, and rotation.
- Call controller.showAddWatermarkView(CPDFWatermarkConfig config) to open the interactive watermark popup.

```
// Set default image watermark in configuration
final configuration = CPDFConfiguration(
  globalConfig: const CPDFGlobalConfig(
    watermark: CPDFWatermarkConfig(
      image: 'ic_logo',
      opacity: 120,
      rotation: -45
    )
  )
);

// Open watermark popup
controller.showAddWatermarkView(
  config: CPDFGlobalConfig(
    watermark: CPDFWatermarkConfig(
      image: 'ic_logo',
      opacity: 120,
      rotation: -45
    )
  )
);
```

Customizing Default Images

You need to handle default images differently for Android and iOS:

Android

1. Add the image to the resources folder:

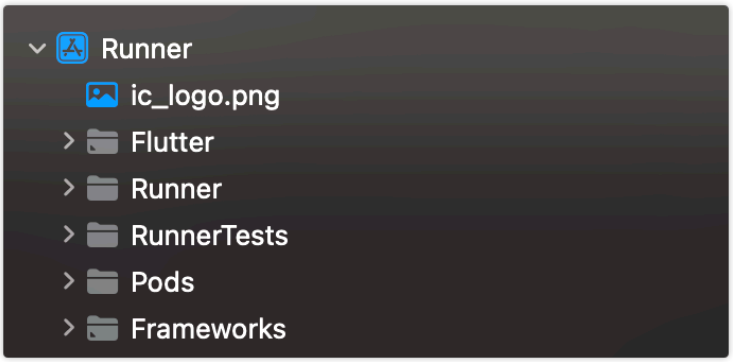
```
android/app/src/main/res/drawable/ic_logo.png
```

1. Reference it in code using the file name without extension:

```
image: 'ic_logo'
```

iOS

1. Import the image into the project via Xcode:



1. Reference it in code using the file name without extension:

```
image: 'ic_logo'
```

Flutter Cross-Platform

You can also import the image into the Flutter project, copy it to device storage at runtime, and pass the local file path:

```
final imagePath = await extractAsset(context, 'images/ic_logo.png');
await controller.showAddWatermarkView(
  config: CPDFWatermarkConfig(
    image: imagePath.path,
    rotation: -45,
    opacity: 200,
    scale: 2.0
  ),
);
```

3.7.4 Remove Watermark

To remove watermarks, use the following method:

```
// Remove all watermarks from the document
document.removeAllWatermarks();
```

3.8 UI Customization

3.8.1 Overview

Custom PDF Viewer in Flutter

ComPDFKit PDF SDK for Flutter allows you to customize the user interface of your application by hiding buttons or toolbars.

For detailed configurable options, please refer to CONFIGURATION.md.

Key Capabilities

- ToolBars - Hide or add function buttons.
- Open the page to edit, add watermarks, configure security settings, capture screenshots, and other feature views.
- Internationalization and Localization

Guides for Customizing the UI

Main Toolbar

How to show or hide toolbar buttons

Annotation Toolbar

How to customize the annotation toolbar

ContentEditor Toolbar

How to customize the content editor toolbar

Form Toolbar

How to customize the form toolbar

Signature Toolbar

How to customize the signature toolbar.

Localization

How to change the language in our ComPDFKit PDF SDK for Flutter

Tools

The toolset includes APIs for page editing dialogs, watermark addition dialogs, security settings, etc. Users can directly invoke and open the corresponding features via `CPDFReaderWidgetController` to implement these functionalities in Flutter.

Context Menu

Used to configure the context menu options that pop up when a comment, text, image, etc. is selected.

UI Visibility

How to show or hide the user interface.

BOTA Configuration

How to configure the BOTA interface to show or hide features and menu options.

Page Editor Menu

How to configure the bottom menu options in the page editing interface.

3.8.2 Main Toolbar

The main toolbar in ComPDFKit is designed to be flexible and highly configurable. This guide shows how to customize it.

Default Toobar

The default toolbar contains the following tools.

Android	iOS
	

Customizing the Toolbar Buttons

You can customize the main toolbar buttons when displaying a PDF using the `iosLeftBarAvailableActions` or `iosRightBarAvailableActions` properties on iOS and the `androidAvailableActions` property on Android. The following example shows how to hide the menu button in the navigation bar (main toolbar) on iOS and how to customize the Android toolbar menu items:

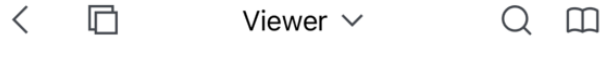
```
CPDFConfiguration configuration = CPDFConfiguration(toolbarConfig: const
CPDFToolbarConfig(
  mainToolbarVisible : true,
  androidAvailableActions: [
    ToolbarAction.back,
    ToolbarAction.search,
    ToolbarAction.thumbnail,
    ToolbarAction.bota,
    ToolbarAction.menu,
  ],
  iosLeftBarAvailableActions: [
    ToolbarAction.back,
    ToolbarAction.thumbnail
  ],
  iosRightBarAvailableActions: [
    ToolbarAction.search,
    ToolbarAction.bota,
  ],
));

// CPDFReaderWidget Sample
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
```

```
    document: documentPath,
    configuration: configuration,
    onCreated: (controller) {},
  ));

// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(documentPath, '', configuration)
```

The customized toolbar will look like what's shown below.

Android	iOS
	

Available Toolbar Customization Options

Toolbar Button Item	Description
back	Shows close button item.
thumbnail	Shows thumbnails button item.
search	Shows search button item.
bota	Shows outline, bookmarks, annotation list button item.
menu	Shows menu button item.

> Note: Please refer to `ToolbarAction` for the relevant options.

Available menu Customization options

If you configure `ToolbarAction.menu`, you can access more functional buttons in the menu. For configurable options, please refer to the following list.

Menu Button Item	Description
viewSettings	Open the settings view and set the scrolling direction, display mode, theme color and other related settings for reading PDF.
documentEditor	Open the document thumbnail list, and you can delete, rotate, and add document pages in the view.
documentInfo	Open the document information view to display basic document information and permission information.
watermark	Open the watermark editing view to add text and image watermarks and save them as a new document.
security	Open the security settings view, set the document opening password and set the permission password
flattened	Flatten the annotations in the document, and the annotations will not be editable.
save	save pdf document.
share	Turn on system sharing function.
openDocument	Open the system file selector and open a new pdf document.

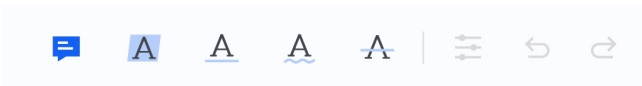
> Note: Please refer to `ToolbarMenuAction` for the relevant options.

3.8.3 Annotation Toolbar

The annotation toolbar in ComPDFKit can be flexibly configured to enable annotation types and tools. This section explains how to customize the annotation toolbar.

Default Annotation ToolBar

The default annotation toolbar contains the following annotation types and tools:

Android	iOS
	

Customizing the Annotation Toolbar Buttons

You can enable or hide specific annotation types by setting the `availableTypes` property in the `annotationsConfig` object. The following example demonstrates how to only display the note and highlight annotation types.

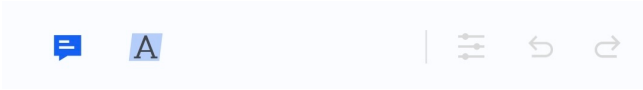
```
CPDFConfiguration configuration = CPDFConfiguration(
  annotationsConfig: const CPDFAnnotationsConfig(availableTypes: [
```

```
    CPDFAnnotationType.note,
    CPDFAnnotationType.highlight
  ], availableTools: [
    CPDFConfigTool.setting,
    CPDFConfigTool.undo,
    CPDFConfigTool.redo,
  ]));

// CPDFReaderWidget Sample
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: configuration,
    onCreated: (controller) {},
  ));

// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(documentPath, '', configuration)
```

The customized toolbar will look like what's shown below.

Android	iOS
	

Available Annotation Toolbar Customization Options

Type					
note	highlight	underline	squiggly	strikeout	ink
pencil	circle	square	arrow	line	freetext
signature	stamp	pictures	link	sound	

> Note: Please refer to `CPDFAnnotationType` for the relevant options. `pencil` is only available on iOS.

Available Annotation Toolbar Tool Customization Options

Tool	Description
setting	Set button, corresponding to open the selected annotation, text or picture property panel.
undo	Undo annotation, content editing, form operations.
redo	Redo an undone action

> Note: Please refer to `CPDFConfigTool` for the relevant options.

Show or Hide Toolbar

You can control the visibility of the bottom toolbar in annotation mode by configuring `CPDFConfiguration`.

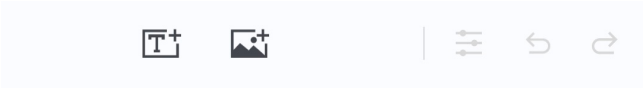
```
CPDFConfiguration configuration = CPDFConfiguration(
  // Set whether the annotation mode bottom toolbar is visible
  toolbarConfig: const CPDFToolbarConfig(annotationToolbarVisible: false),
);
```

3.8.4 Content Editor Toolbar

The content editing toolbar in ComPDFKit can be flexibly configured to enable content editing types and tools. This section describes how to customize the content editing toolbar.

Default Content Editor ToolBar

The default content editor toolbar contains the following editor types and tools:

Android	iOS
	

Customizing the Content Editor Toolbar Buttons

You can enable or hide specific edit types by setting the `availableTypes` property in the `contentEditorConfig` object. The following example shows how to adjust the order of edit types and hide the settings button for the content editing tool.

```
CPDFConfiguration configuration = CPDFConfiguration(
  contentEditorConfig: const CPDFContentEditorConfig(availableTypes: [
    CPDFContentEditorType.editorImage,
    CPDFContentEditorType.editorText
  ], availableTools: [
    CPDFConfigTool.undo,
    CPDFConfigTool.redo,
  ]));

// CPDFReaderWidget Sample
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: configuration,
    onCreated: (controller) {},
  ));

// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(documentPath, '', configuration)
```

The customized toolbar will look like what's shown below.

Android	iOS
	

Available Content Editor Toolbar Customization Options

Type
CPDFContentEditorType.editorText
CPDFContentEditorType.editorImage

> Note: Please refer to `CPDFContentEditorType` for the relevant options.

Available Content Editor Toolbar Tool Customization Options

Tool	Description
setting	Set button, corresponding to open the selected annotation, text or picture property panel.
undo	Undo annotation, content editing, form operations.
redo	Redo an undone action

> Note: Please refer to `CPDFConfigTool` for the relevant options.

Show or Hide Toolbar

You can control the visibility of the bottom toolbar in **content editing mode** by configuring `CPDFConfiguration`.

```
CPDFConfiguration configuration = CPDFConfiguration(
  // Set whether the content editor mode bottom toolbar is visible
  toolbarConfig: const CPDFToolbarConfig(contentEditorToolbarVisible: false),
);
```

3.8.5 Forms Toolbar

The form toolbar in ComPDFKit can be flexibly configured to enable form types and tools. This section explains how to customize the form toolbar.

Default Form ToolBar

The default form toolbar contains the following form types and tools:

Android	iOS
	

Customizing the Form Toolbar Buttons

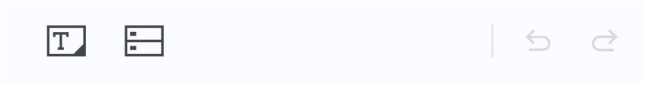
You can enable or hide specific editing types by setting the `availableTypes` property in the `formsConfig` object. The following example demonstrates how to adjust the form types to only enable text fields and list boxes.

```
CPDFConfiguration configuration = CPDFConfiguration(
  formsConfig: const CPDFFormsConfig(availableTypes: [
    CPDFFormType.textField,
    CPDFFormType.listBox
  ],availableTools: [
    CPDFFormConfigTool.undo,
    CPDFFormConfigTool.redo
  ]));

// CPDFReaderWidget Sample
Scaffold(
  resizeToAvoidBottomInset: false,
  appBar: AppBar(),
  body: CPDFReaderWidget(
    document: documentPath,
    configuration: configuration,
    onCreated: (controller) {},
  ));

// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(documentPath, '', configuration)
```

The customized toolbar will look like what's shown below.

Android	iOS
	

Available Form Toolbar Customization Options

Type
CPDFFormType.textField
CPDFFormType.checkBox
CPDFFormType.radioButton
CPDFFormType.listBox
CPDFFormType.comboBox
CPDFFormType.signaturesFields
CPDFFormType.pushButton

> Note: Please refer to `CPDFFormType` for the relevant options.

Available Form Toolbar Tool Customization Options

Tool	Description
SETTING	Set button, corresponding to open the selected annotation, text or picture property panel.
UNDO	Undo annotation, content editing, form operations.
REDO	Redo an undone action

> Note: Please refer to `CPDFFormConfigTool` for the relevant options.

Show or Hide Toolbar

You can control the visibility of the bottom toolbar in **form mode** by configuring `CPDFConfiguration`.

```
CPDFConfiguration configuration = CPDFConfiguration(
  // Set whether the form mode bottom toolbar is visible
  toolbarConfig: const CPDFToolbarConfig(formToolbarVisible: false),
);
```

3.8.6 Signature Toolbar

Show or Hide Toolbar

You can control the visibility of the bottom toolbar in **signature mode** by configuring `CPDFConfiguration`.

```
CPDFConfiguration configuration = CPDFConfiguration(
  // Set whether the signature mode bottom toolbar is visible
  toolbarConfig: const CPDFToolbarConfig(signatureToolbarVisible: false),
);
```

3.8.7 Localization

The ComPDFKit SDK for Flutter comes with the following built-in languages:

- English(en)
- Chinese Simplified (zh-Hans)
- Spanish(es)

Adding Additional Localization to ComPDFKit

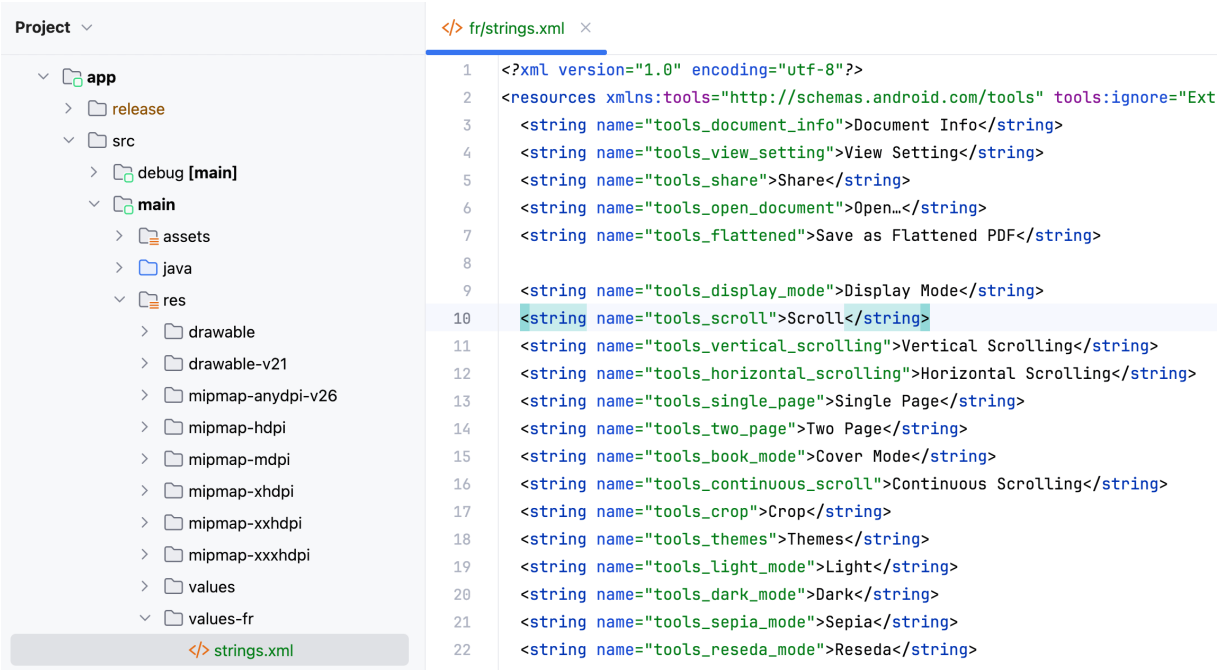
- **Android Platform**

You can add additional translations by putting them into the `res/strings-XX` directory of your app. Android will automatically merge all string resources at build time. You can also override ComPDFKit strings of existing languages by putting them into the respective string folders.

> Tip: To view a list of all available ComPDFKit string resources, copy them to your project from the following URL:
>
> strings.xml

Here's an example of adding French language support:

1. Create a `app/src/main/res/values-fr` directory in your project, and create a `strings.xml` file within it.
2. Copy the content from the link above into your `strings.xml` file.



3. Translate the strings into French. For example:

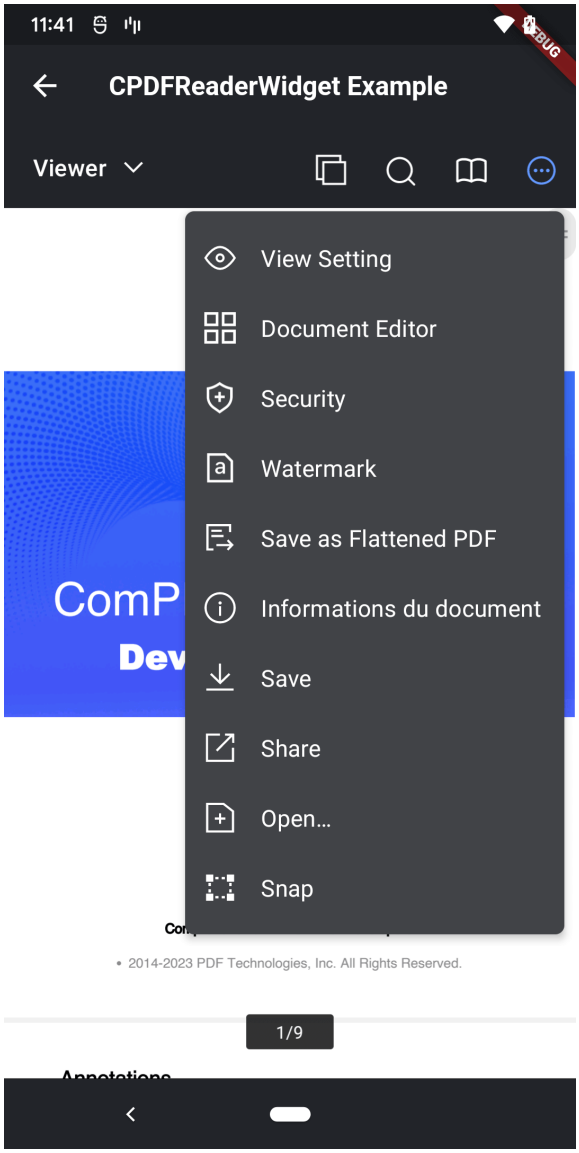
Original content:

```
<string name="tools_document_info">Document Info</string>
```

Translate it to French and replace the original content:

```
<string name="tools_document_info">Informations du document</string>
```

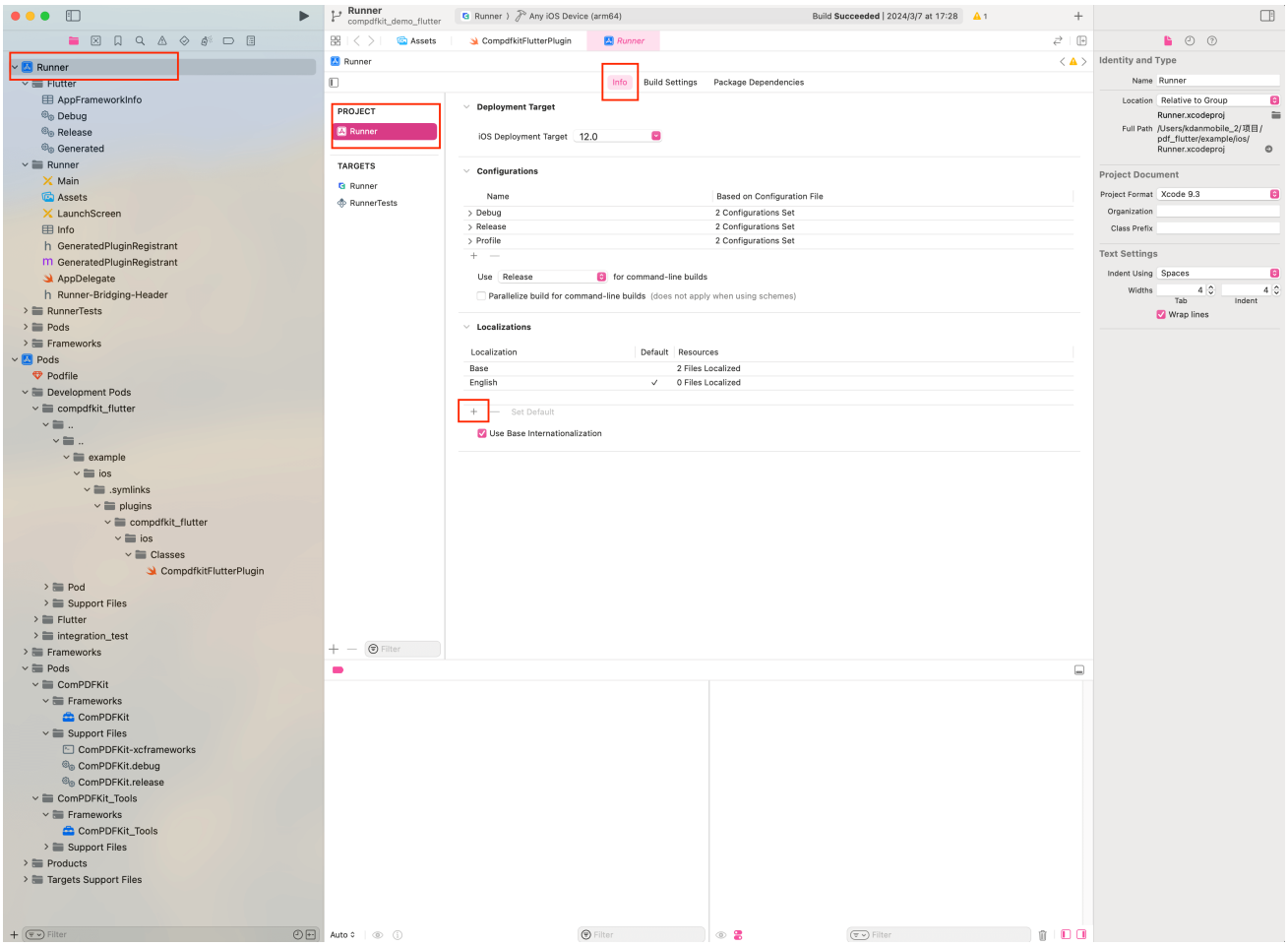
Switch your device to French, and after running your project, you will see the translated content:



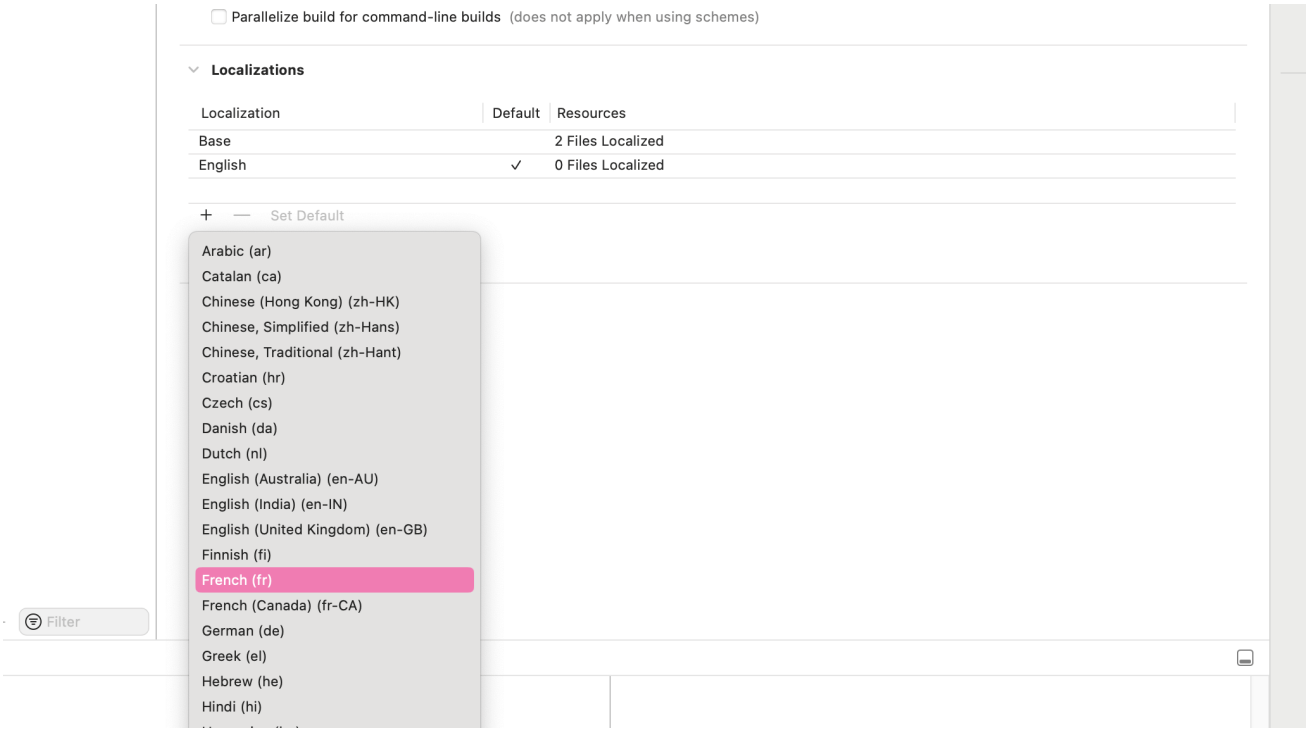
- **iOS Platform**

Configure the languages that require internationalization

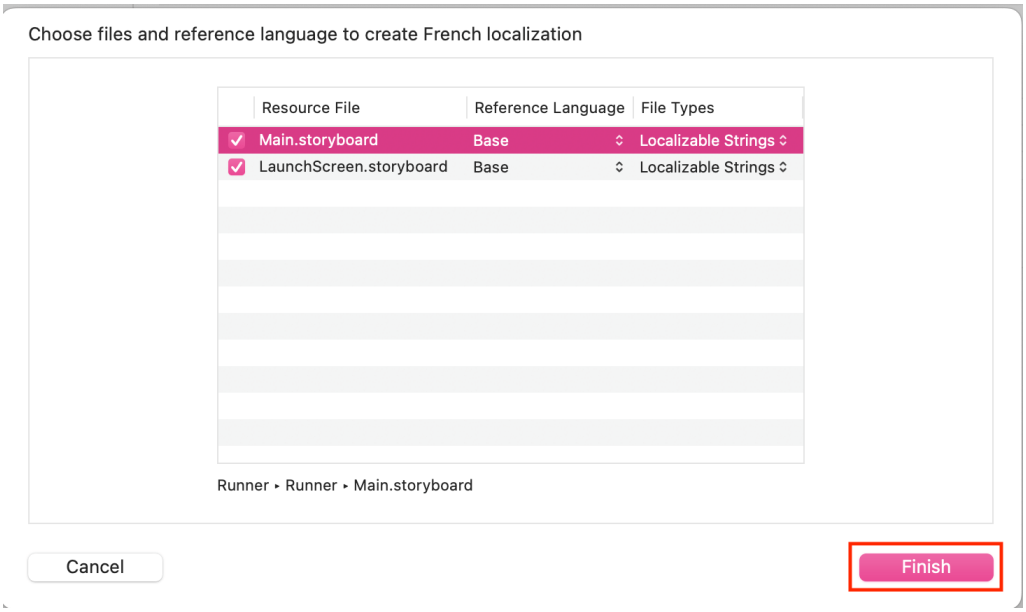
1. Open your Flutter project's iOS project using Xcode.Then, Select project -> Info -> Localizations, then click '+', add the desired language for internationalization/localization, as shown below (Make sure to check 'Use Base Internationalization' by default):



2. Here, we'll use adding French as an example, as shown below:



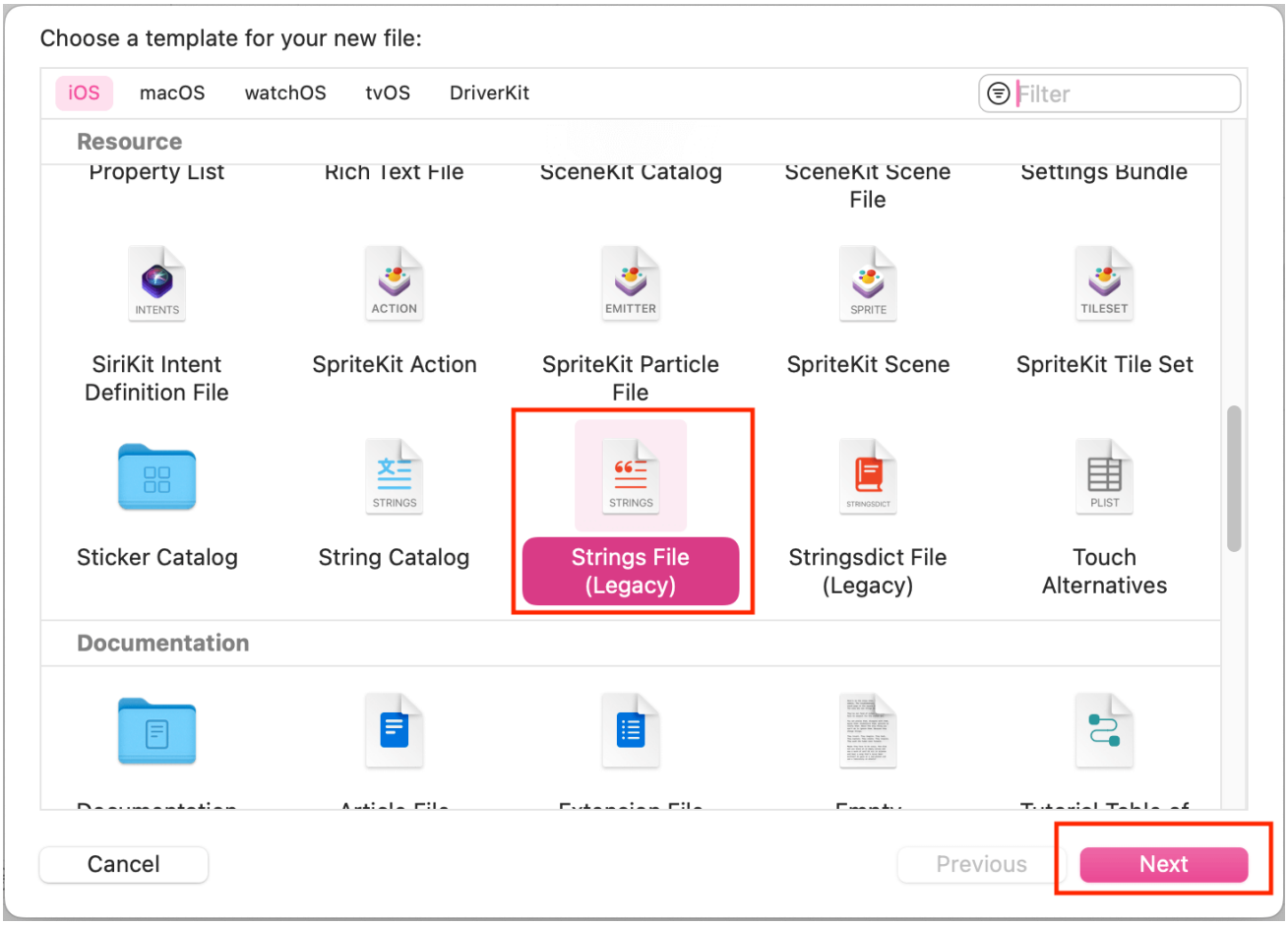
3. Pop up the following dialog, and click 'Finish' directly, as shown below:



Internationalizing the application name

Internationalizing the application name refers to displaying different names for the same app in various language environments (i.e., the language settings on the mobile device).

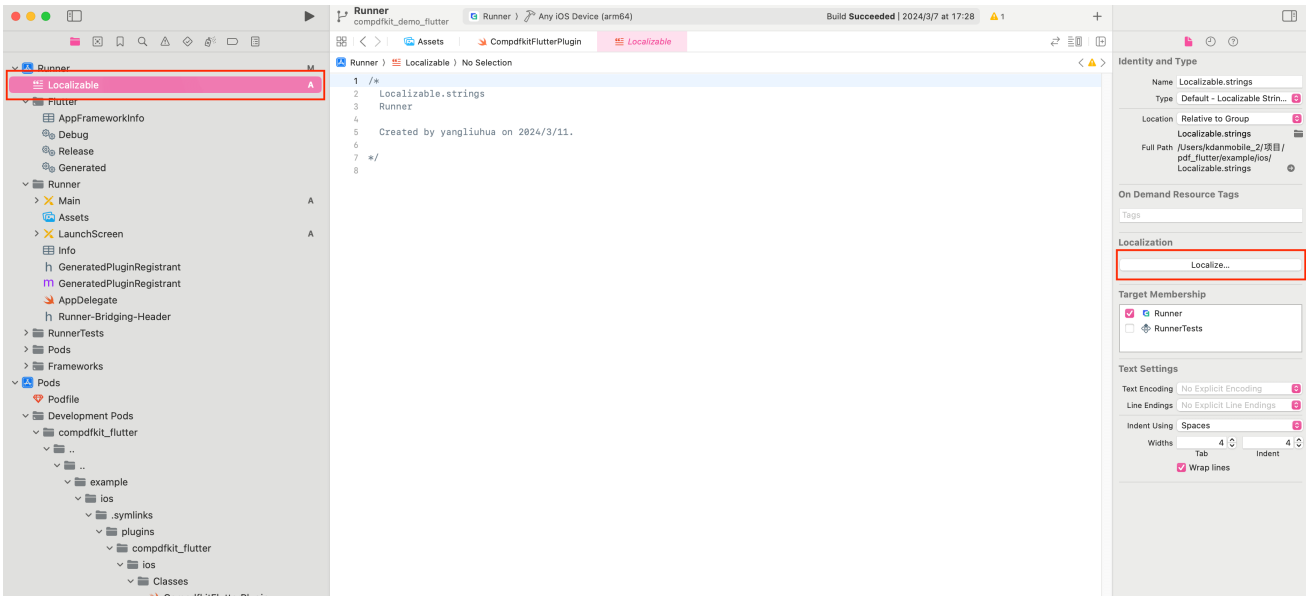
1. Creating `Localizable.strings` file



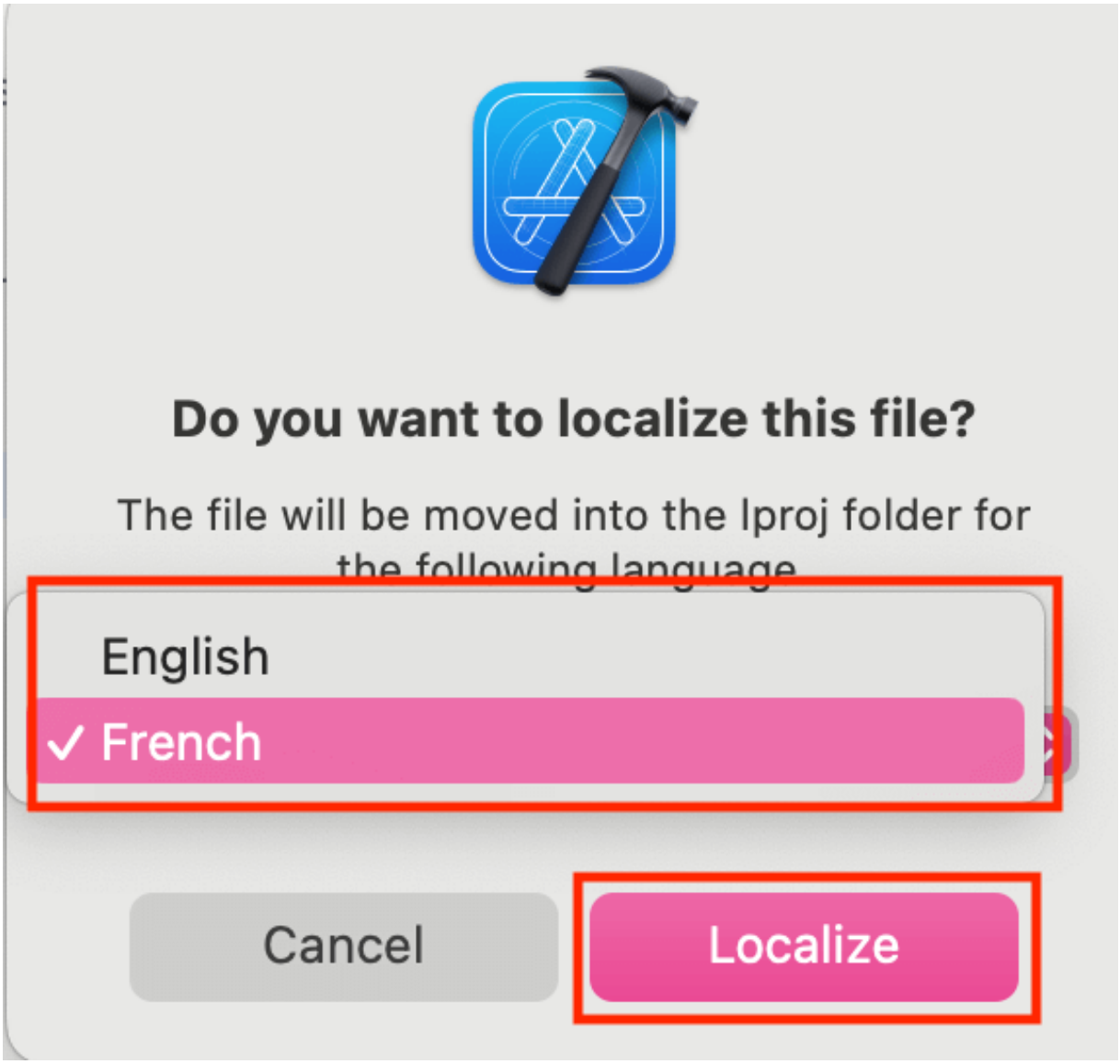
2. Select `Localizable.strings` , click on Localize in the File inspection (right-side file inspector) in Xcode. The purpose is to choose the language we want to localize, as shown below:

   Note: Before clicking on Localize, make sure that we have already added the languages for localization. This is the step where we configure the languages for internationalization (Steps: project-

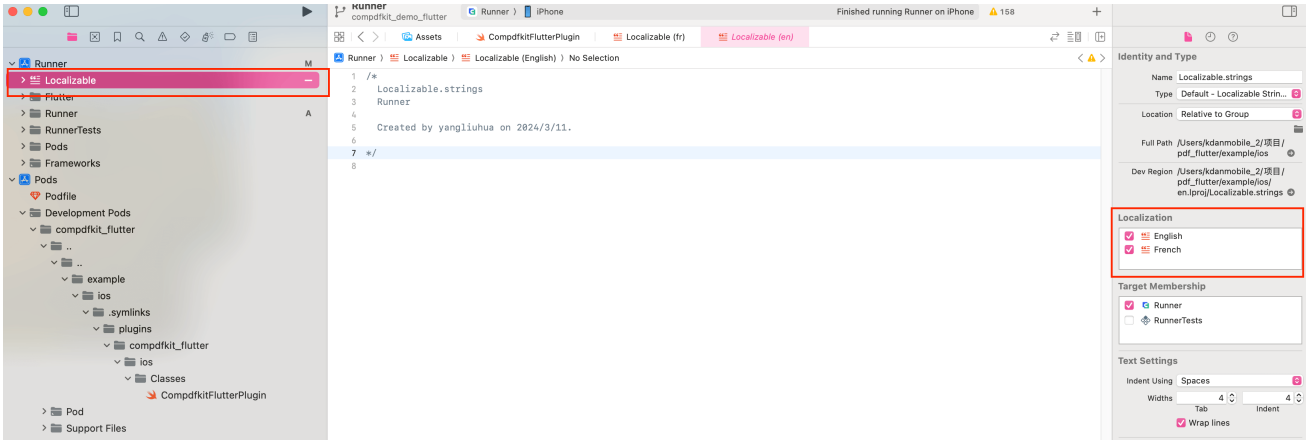
>Info->Localizations, then click '+', add the desired languages for internationalization/localization).



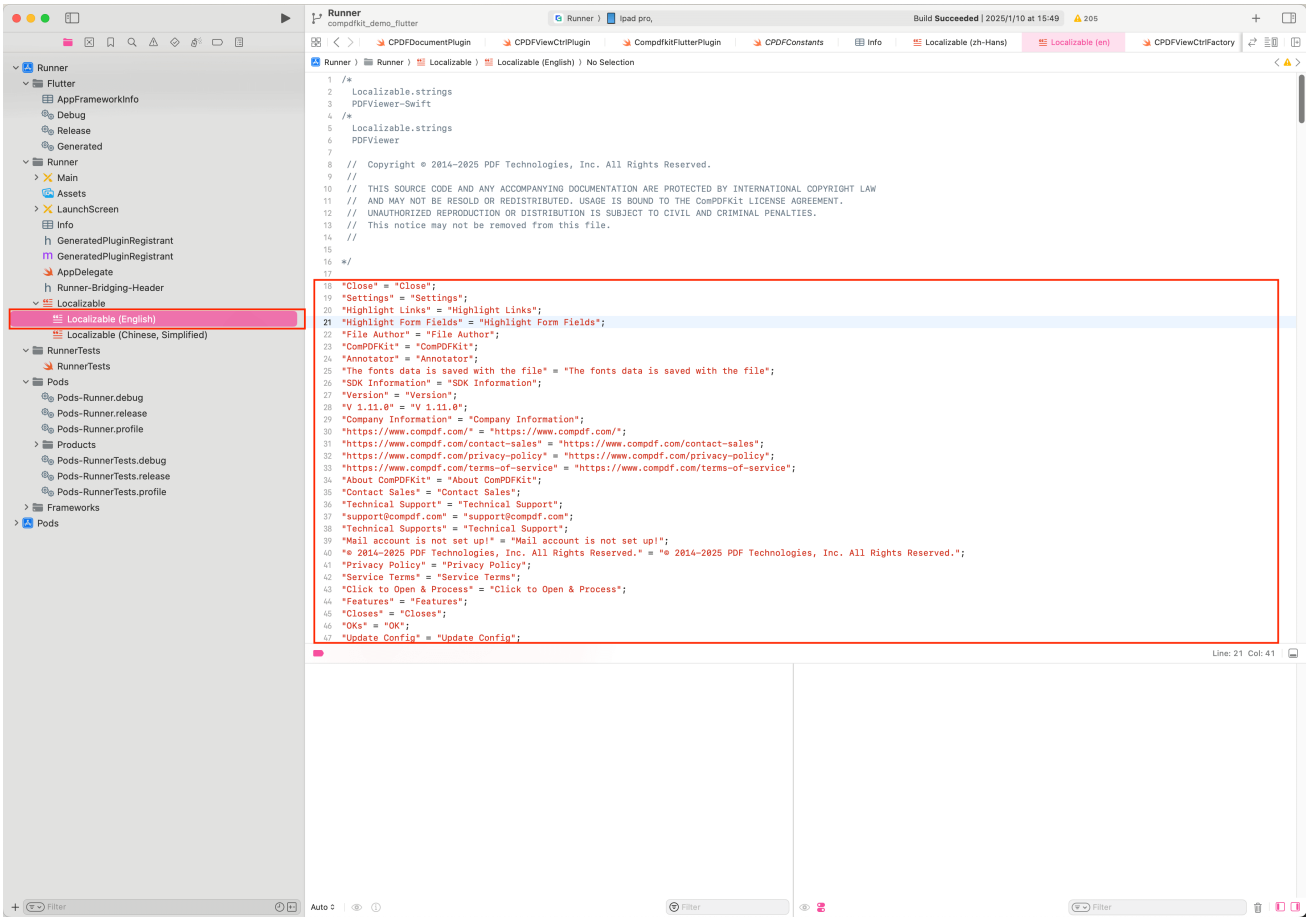
3. After clicking on Localize, a dialog will appear. Expand the dialog list, and you will find that the dropdown list shows the languages we configured for internationalization above. Choose the language we want to localize, then click the 'Localize' button in the dialog, as shown below:



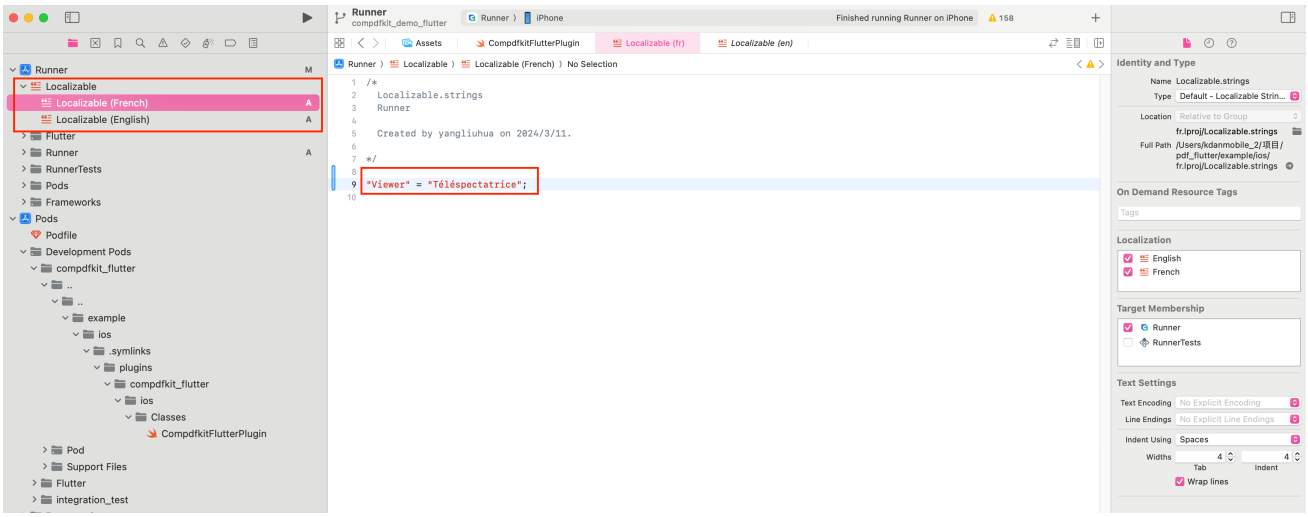
4. Next, check French and English, as shown below:



5. Then open the ComPDFKit Flutter iOS Demo, find the `Localizable (English)` file, select all the text, copy it, and paste it into the `Localizable (English)` file you created.



6. Finally, select the `Localizable (French)` file and configure the corresponding French translations according to the `Localizable (English)` file, ensuring that the number of text segments in both files is the same. For example, if the `Localizable (English)` file has `"Viewer" = "Viewer";`, the `Localizable (French)` file should have the corresponding `"Viewer" = "Téléspectatrice";`.



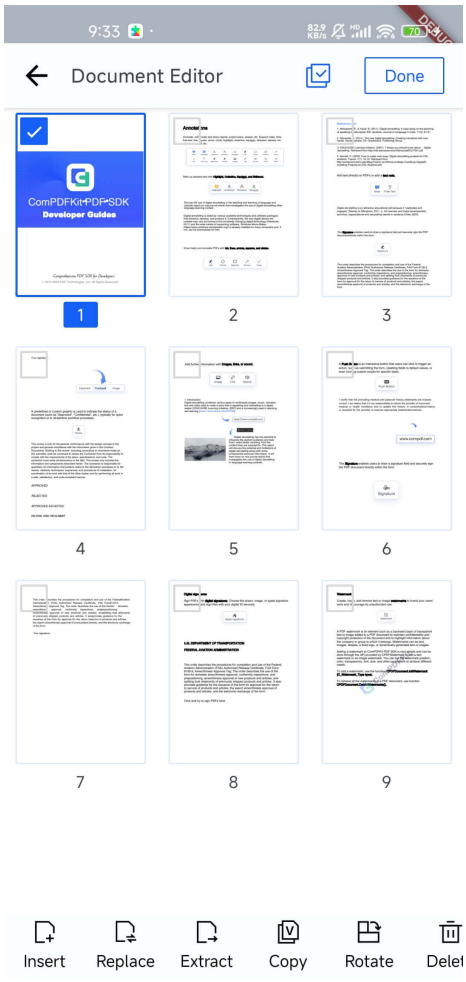
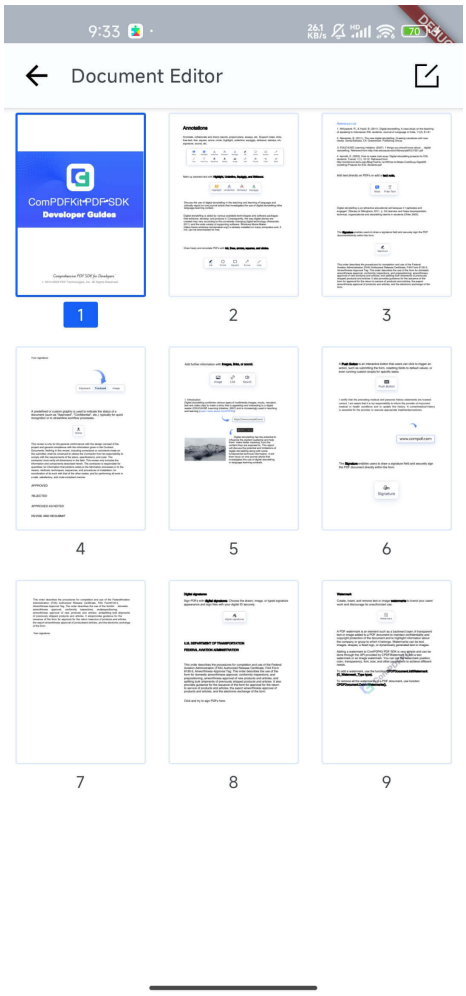
3.8.8 Tools

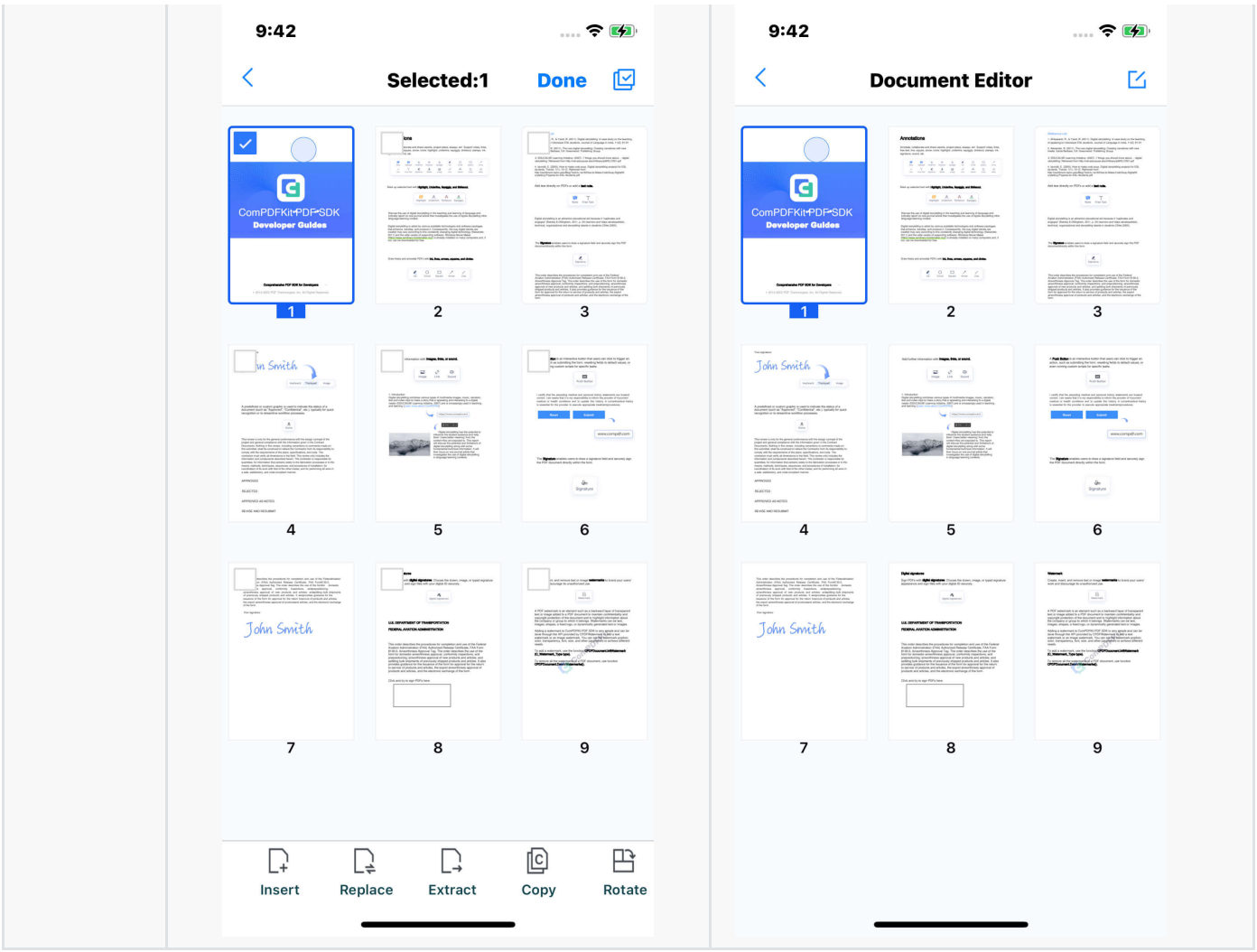
This section introduces how to use the API provided by `CPDFReaderWidgetController` to directly access features such as the page editing view, watermark addition view, security settings view, etc., while displaying PDF documents with `CPDFReaderWidget` in Flutter, providing a more flexible user experience.

Open Page Editing View

```
// Control whether to enter page editing mode; when false, the thumbnail list is shown
bool editMode = true;
controller.showThumbnailView(editMode);
```

	editMode:true	editMode:false

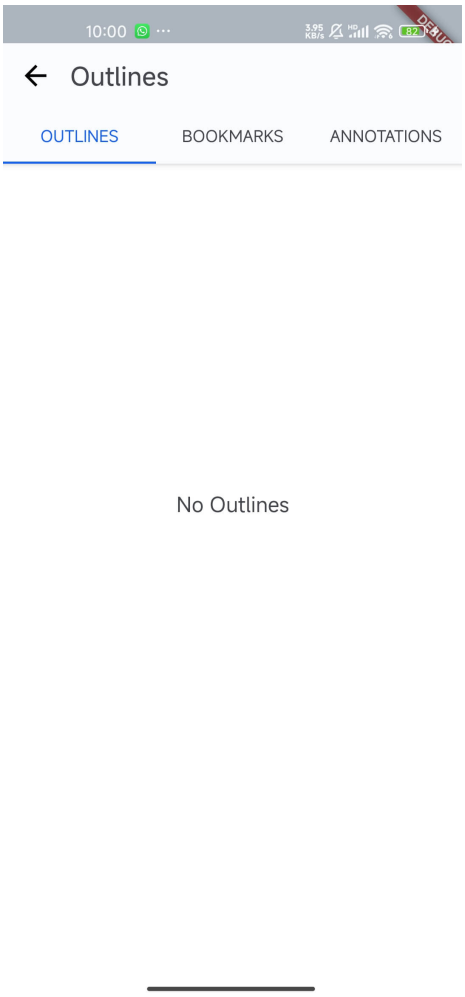
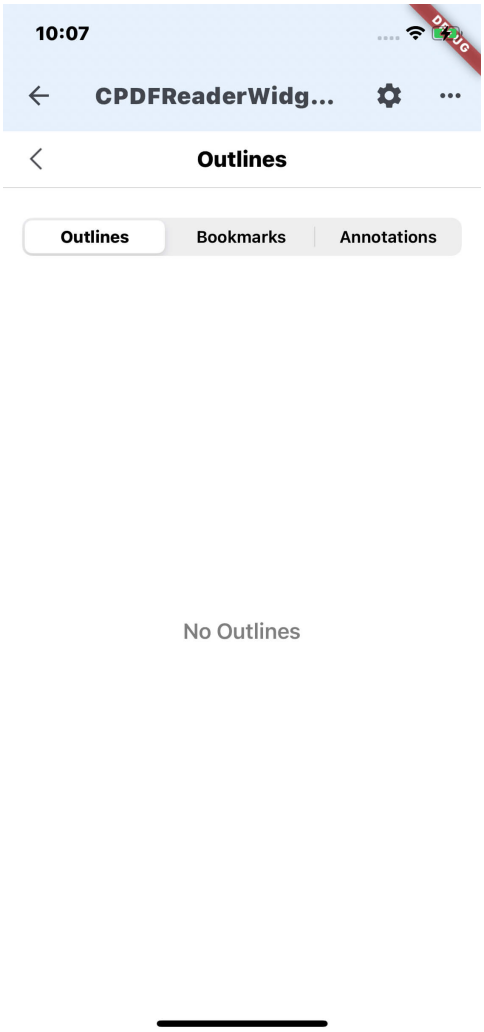
Android	 Document Editor Done Insert Replace Extract Copy Rotate Delet	 Document Editor
iOS		



Open BOTA View

This view displays the document outline, bookmarks, and annotations lists.

```
await controller.showBotaView();
```

Android	iOS
	

Open Add Watermark Dialog

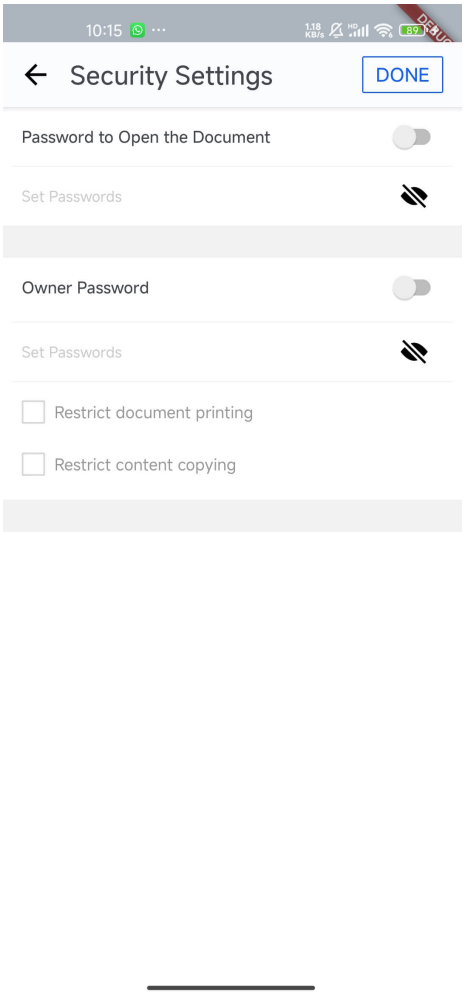
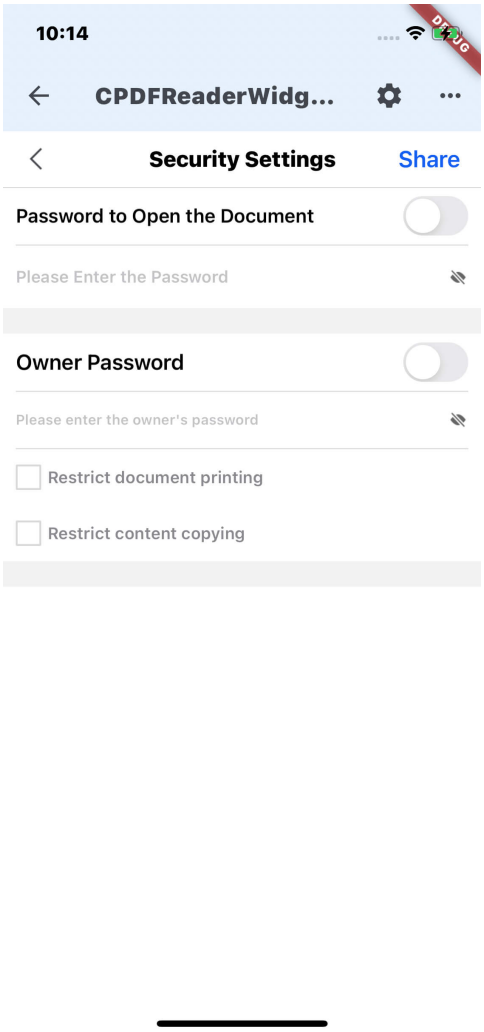
```
await controller.showAddWatermarkView();
```

Android	iOS
	

Open Security Settings View

In the security settings view, you can configure the document password, owner permissions password, and encryption method.

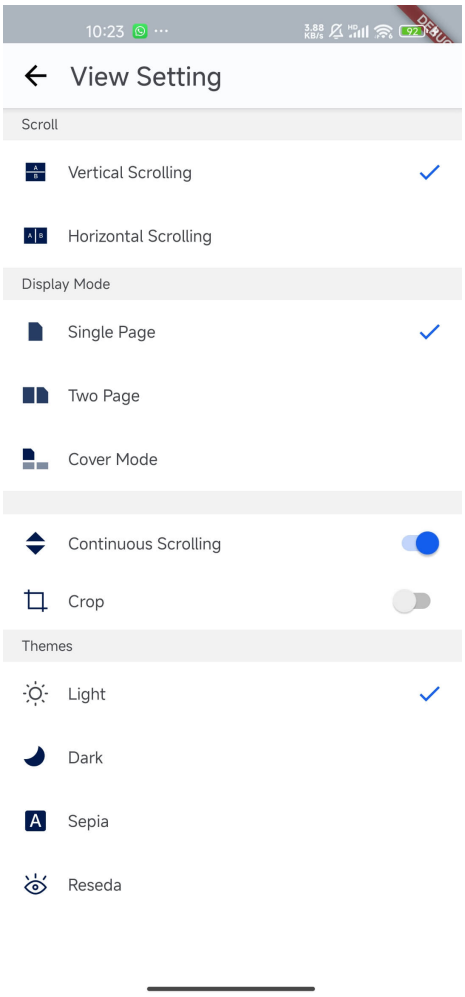
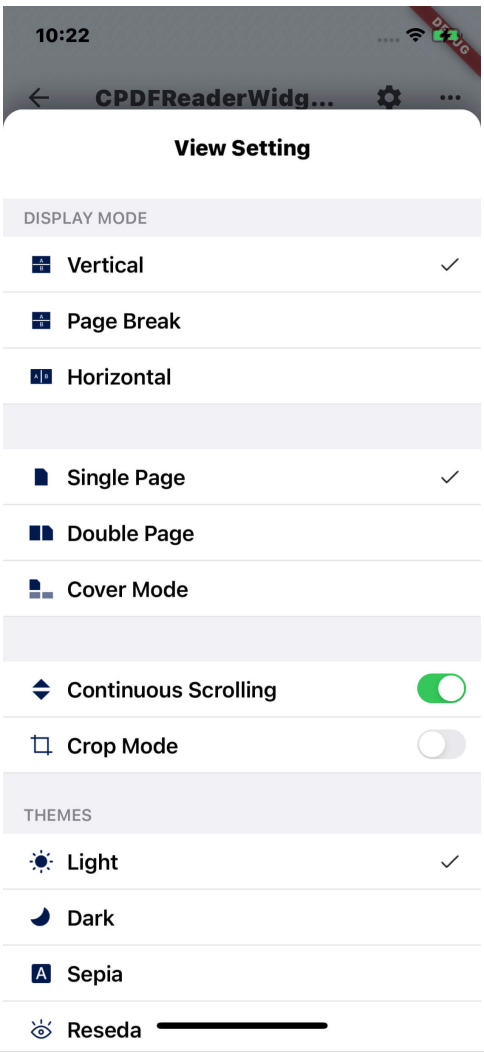
```
await controller.showSecurityView();
```

Android	iOS
	

Open Display Settings View

The display settings view allows you to configure the scroll direction, scroll mode, theme color, and other options.

```
await controller.showDisplaySettingView();
```

Android	iOS
	

Snipping Function

You can enter snipping mode when displaying a PDF document via the API to capture a specific area.

```
// Enter snipping mode
await controller.enterSnipMode();
// Exit snipping mode
await controller.exitSnipMode();
```

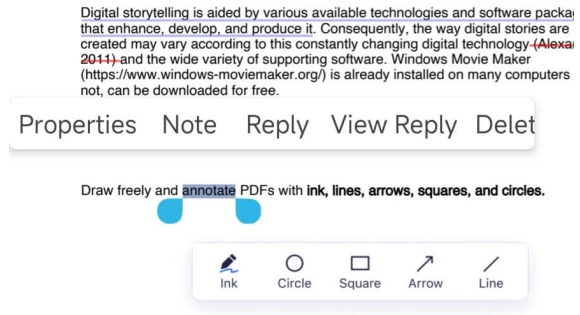
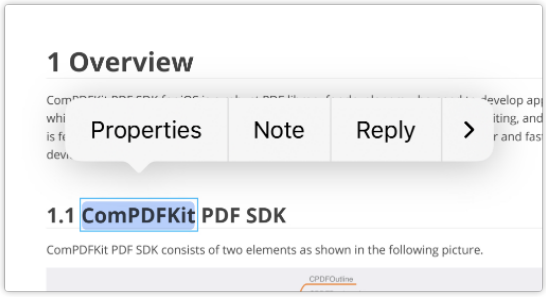
Android	iOS
	

3.8.9 Context Menu

When creating a CPDFReaderWidget, you can customize the context menus that appear when annotations, text, images, or form fields are selected. This is done via the CPDFConfiguration object using the contextMenuConfig field.

Default Context Menu

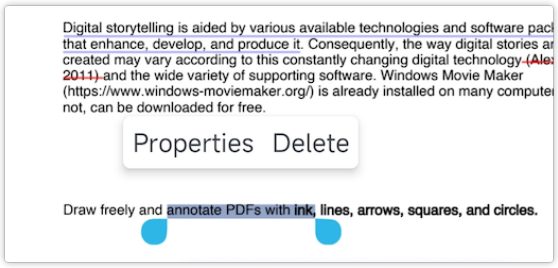
By default, when a user selects a **highlight annotation**, ComPDFKit displays a context menu with common options such as “Note”, “Delete”, and “Properties”:

Android	iOS
	

Customizing Menu Items

You can customize the context menu by specifying which menu items to include. The following example only keeps the **Properties** and **Delete** options for highlight annotations:

```
CPDFReaderWidget(
  document: widget.documentPath,
  password: widget.password,
  configuration: CPDFConfiguration(
    contextMenuConfig: const CPDFContextMenuConfig(
      annotationMode: CPDFAnnotationModeContextMenu(
        markup: [
          CPDFContextMenuItem(CPDFAnnotationMarkupMenuKey.properties),
          CPDFContextMenuItem(CPDFAnnotationMarkupMenuKey.delete),
        ]
      )
    )
  ),
  onCreated: (controller) {
  },
);
```

Android	iOS
	

> For full customization details, refer to the official documentation: CONFIGURATION.md - contextMenuConfig

3.8.10 UI Visibility

The ComPDFKit Flutter SDK provides several ways to show or hide the user interface (UI). The following table summarizes the supported options:

Option	Description
automatic	Toolbars and other UI elements automatically show or hide when the page is tapped
always	UI is always visible
never	UI is always hidden

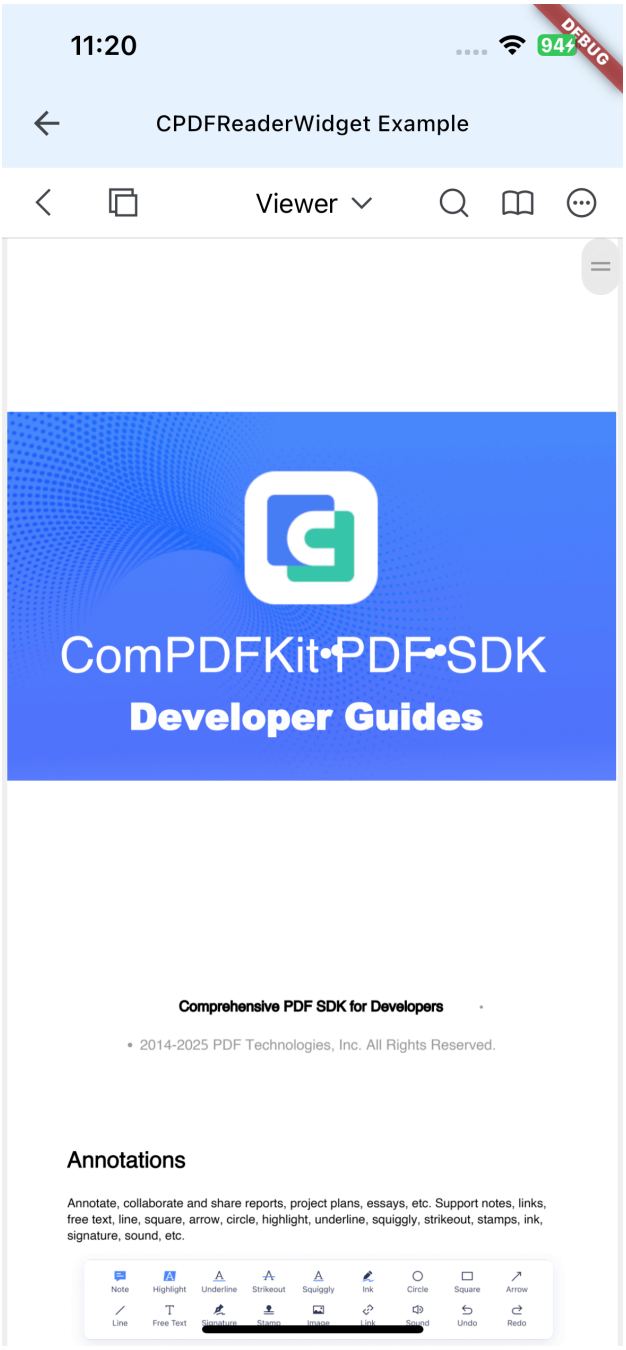
You can change the UI visibility mode using the `uiVisibilityMode` option. The example below shows how to use **automatic** mode:

```
CPDFConfiguration(
  modeConfig: const CPDFModeConfig(
    uiVisibilityMode: CPDFUIVisibilityMode.automatic,
  ),
);
```

Automatic mode behavior:

Android	iOS
	

Always mode behavior:

Android	iOS
	

Never mode behavior:

Android	iOS
	

3.8.11 BOTA Configuration

Overview

The BOTA interface displays **Bookmarks**, **Outline**, and **Annotations** for a PDF document.

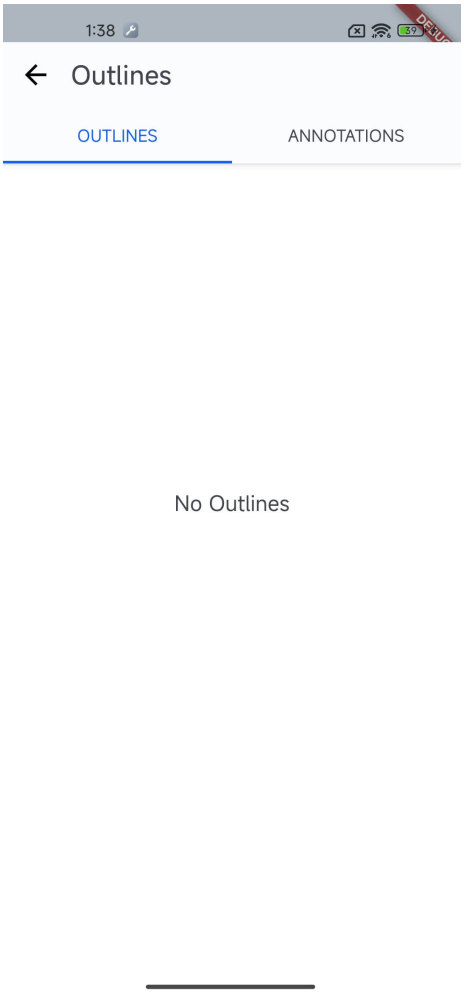
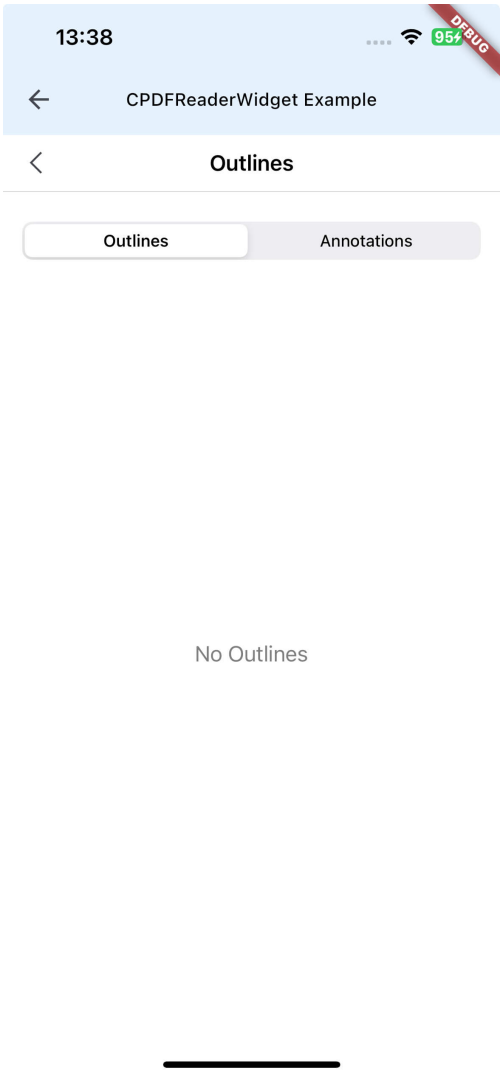
By configuring BOTA, you can control which sections are displayed and customize menu options within each section.

Enable Specific Tabs

Use `CPDFBotaConfig.tabs` to configure the tabs to show:

```
CPDFConfiguration config = CPDFConfiguration(
  globalConfig: const CPDFGlobalConfig(
    bota: CPDFBotaConfig(
      tabs: [
        CPDFBotaTabs.outline,
        CPDFBotaTabs.annotations
      ]
    )
  )
);
```

Example:

Android	iOS
	

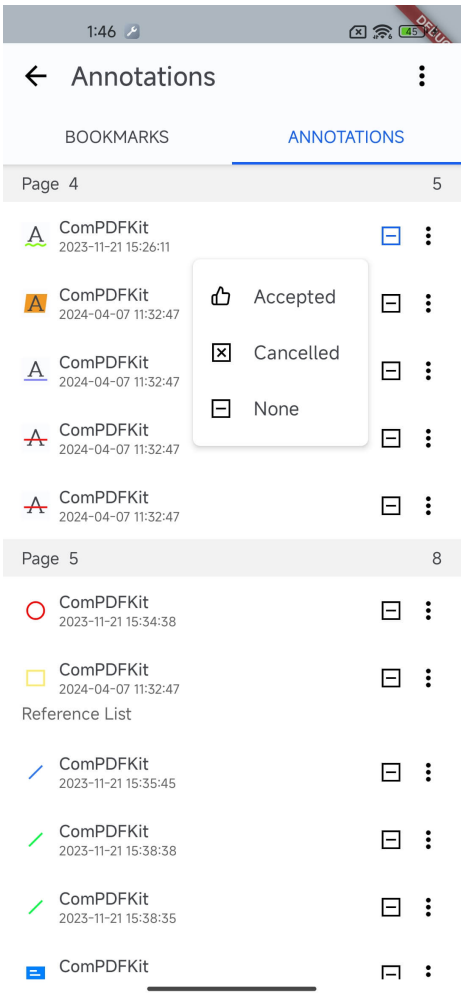
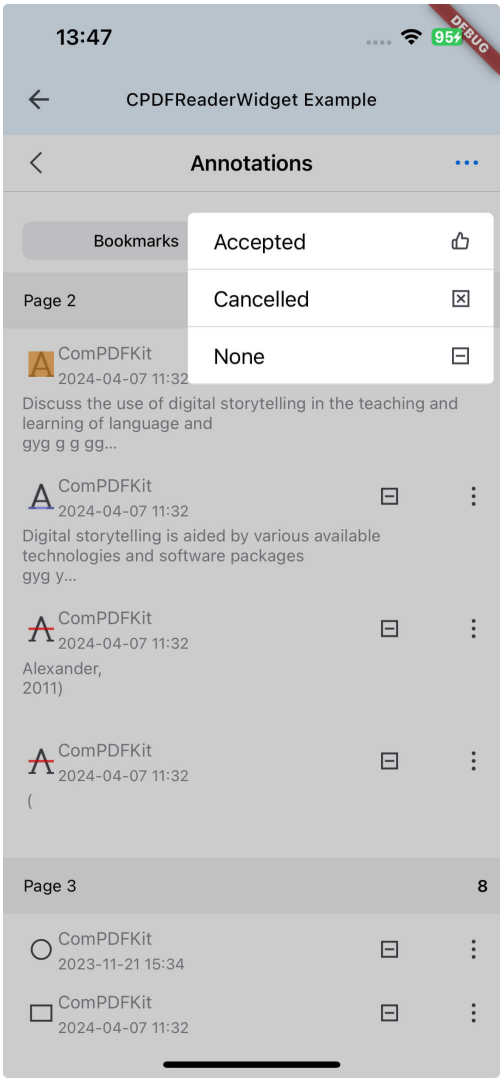
Configure Annotation Menus

BOTA supports both **global menus** and **per-annotation menus**:

```
CPDFConfiguration(
  globalConfig: const CPDFGlobalConfig(
    bota: CPDFBotaConfig(
```

```
      tabs: [
        CPDFBotaTabs.bookmark,
        CPDFBotaTabs.annotations,
      ],
      menus: CPDFBotaMenuConfig(
        annotations: CPDFBotaAnnotationMenuConfig(
          global: [
            CPDFBotaMenuItem(id: CPDFBotaAnnotGlobalMenu.importAnnotation),
            CPDFBotaMenuItem(id: CPDFBotaAnnotGlobalMenu.exportAnnotation),
            CPDFBotaMenuItem(id: CPDFBotaAnnotGlobalMenu.removeAllAnnotation),
          ],
          item: [
            CPDFBotaMenuItem(id: CPDFBotaAnnotItemMenu.reviewStatus, subMenus: [
              CPDFReviewState.accepted,
              CPDFReviewState.cancelled,
              CPDFReviewState.none,
            ]),
            CPDFBotaMenuItem(id: CPDFBotaAnnotItemMenu.more, subMenus: [
              CPDFBotaAnnotMoreMenu.delete
            ]),
          ]
        )
      )
    )
  )
)
```

Example:

Android	iOS
	

Annotation Global Menu Options

Option	Description
CPDFBotaAnnotGlobalMenu.importAnnotation	Import annotations
CPDFBotaAnnotGlobalMenu.exportAnnotation	Export annotations
CPDFBotaAnnotGlobalMenu.removeAllAnnotation	Remove all annotations
CPDFBotaAnnotGlobalMenu.removeAllReply	Remove all annotation replies

Annotation Review Status Submenu

Option	Description
accepted	Accepted
rejected	Rejected
cancelled	Cancelled
completed	Completed
none	None

More Menu Options

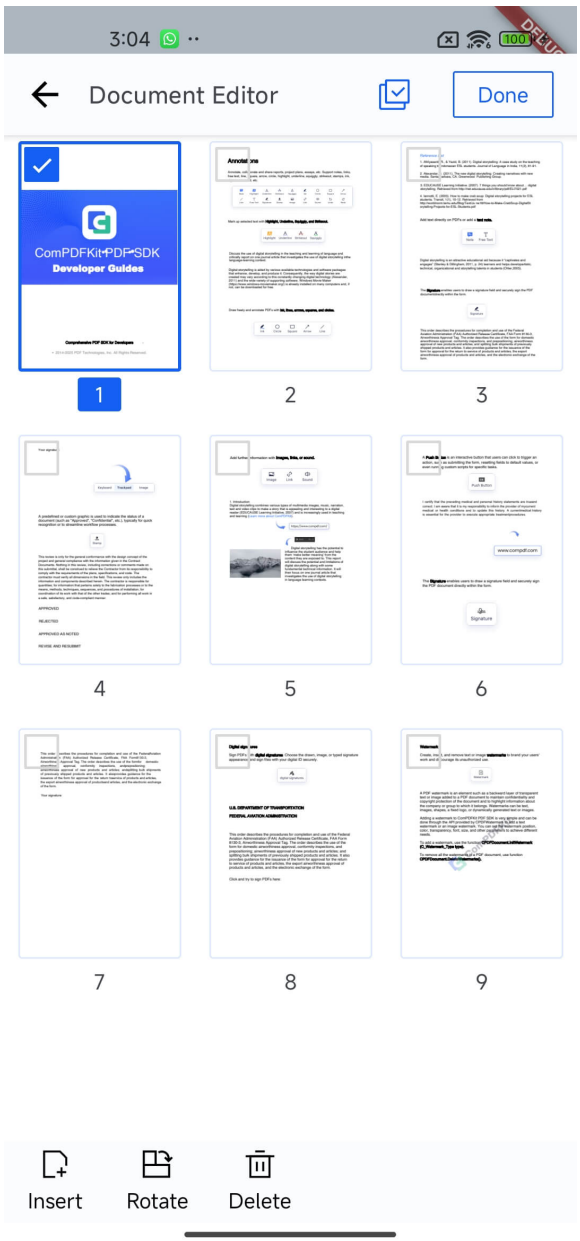
Option	Description
addReply	Add annotation reply
viewReply	View annotation reply
delete	Delete annotation

3.8.12 Page Editor Menu

The ComPDFKit Flutter SDK allows you to configure the bottom menu in **Page Editing** mode via `CPDFConfiguration`.

```
CPDFConfiguration(
  globalConfig: const CPDFGlobalConfig(
    pageEditor: CPDFPageEditorConfig(
      menus: [
        CPDFPageEditorMenus.insertPage,
        CPDFPageEditorMenus.rotatePage,
        CPDFPageEditorMenus.deletePage,
      ]
    )
  )
)
```

Example:

Android	iOS
	

3.9 Content Editor

3.9.1 Overview

Content editor provides capabilities to edit **text, images, and paths**, allowing users to insert, adjust, or delete elements in a PDF. This makes the PDF behave like a rich text editor, enabling direct creation and modification within the document.

Guides

- Content Editor Mode
  Learn how to switch to content editor mode.
- Setting Edit Types
  Use API or the toolbar to select which edit types are enabled.
- Undo and Redo

Undo or redo changes using either the toolbar or API calls.

3.9.2 Content Editor Mode

Before editing content in a `CPDFReaderWidget`, you need to switch to **`CPDFViewMode.contentEditor`** mode.

Switching methods include:

1. **UI Switch**: Select from the mode switch menu in the top-right corner of `CPDFReaderWidget`.
2. **Set at Initialization**: Set the initial mode in `CPDFConfiguration`.
3. **Switch at Runtime**: Call the corresponding API on `CPDFReaderWidgetController`.

```
// Set content editing mode at initialization
CPDFReaderWidget(
  document: documentPath,
  configuration: CPDFConfiguration(
    modeConfig: const CPDFModeConfig(
      initialViewMode: CPDFViewMode.contentEditor
    )
  ),
  onCreated: (controller) {},
);

// Switch to content editing mode at runtime
await controller.setPreviewMode(CPDFViewMode.contentEditor);
```

3.9.3 Setting Edit Types

Once in content editing mode, developers can control which elements users are allowed to edit:

- **Text Editing**: Add, delete, or modify text content in the PDF.
- **Image Editing**: Insert, replace, or delete images in the PDF.
- **Path Editing**: Modify vector paths such as lines or shapes.

Toolbar Switch

Users can switch edit types using the **bottom toolbar**.

API Switch

If the toolbar is hidden, you can use the API:

```
// Enable only text editing
controller.editManager.changeEditType([CPDFEditType.text]);

// Enable both text and image editing
controller.editManager.changeEditType([CPDFEditType.text, CPDFEditType.image]);

// Disable all editing (read-only)
controller.editManager.changeEditType([CPDFEditType.none]);
```

3.9.4 Undo and Redo

Content editing supports **Undo** and **Redo**, allowing users to quickly revert or restore actions during editing.

Toolbar Actions

Users can directly tap the **Undo/Redo** buttons on the bottom toolbar.

API Calls

Developers can also use `historyManager` to implement custom logic:

Set Callback

```
// Set a listener to track history changes
controller.editManager.historyManager.setOnHistoryChangedListener(
  (pageIndex, canUndo, canRedo) {
    debugPrint(
      'History changed: page=$pageIndex, canUndo=$canUndo, canRedo=$canRedo',
    );
  },
);

// Undo
if (await controller.editManager.historyManager.canUndo()) {
  await controller.editManager.historyManager.undo();
}

// Redo
if (await controller.editManager.historyManager.canRedo()) {
  await controller.editManager.historyManager.redo();
}
```

4 Troubleshooting

1. SSL network request to download 'ComPDFKit' library failed when cocopods downloaded iOS third-party library

If SSL network requests fail to download the `ComPDFKit` library when you run `pod install`, replace the third-party platform download address link of the ComPDFKit library and execute `pod install`

```
require_relative '../node_modules/react-native/scripts/react_native_pods'
require_relative '../node_modules/@react-native-community/cli-platform-ios/native_modules'

- platform :ios, '10.0'
+ platform :ios, '12.0'
install! 'cocoapods', :deterministic_uuids => false

target 'PDFView_RN' do
  config = use_native_modules!

  # Flags change depending on the env values.
```

```
  flags = get_default_flags()

  use_react_native!(
    :path => config[:reactNativePath],
    # to enable hermes on iOS, change `false` to `true` and then install pods
    :hermes_enabled => flags[:hermes_enabled],
    :fabric_enabled => flags[:fabric_enabled],
    # An absolute path to your application root.
    :app_path => "#{Pod::Config.instance.installation_root}/.."
  )

  target 'PDFView_RNTests' do
    inherit! :complete
    # Pods for testing
  end

+  pod 'ComPDFKit', :git => 'https://github.com/ComPDFKit/compdfkit-pdf-sdk-ios-
swift.git', :tag => '2.5.0'
+  pod 'ComPDFKit_Tools', :git => 'https://github.com/ComPDFKit/compdfkit-pdf-sdk-ios-
swift.git', :tag => '2.5.0'

  # Enables Flipper.
  #
  # Note that if you have use_frameworks! enabled, Flipper will not work and
  # you should disable the next line.
  use_flipper!()

  post_install do |installer|
    react_native_post_install(installer)
    __apply_Xcode_12_5_M1_post_install_workaround(installer)
  end
end
```

5 Support

5.1 Reporting Problems

Thank you for your interest in ComPDFKit, 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 Flutter PDF Library, 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.

5.2 Contact Information

Home Link:

https://www.compdf.com

Support & General Contact:

Email: support@compdf.com

Thanks,
The ComPDFKit Team