Contents

1 Overview

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

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

1.1 ComPDFKit for React Native

ComPDFKit for React Native consists of two elements.

The two elements of ComPDFKit for React Native:

- **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.

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 for React Native 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 the integration, please ensure the following prerequisites are met.

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 ReactNative 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 NPM.

2.1.3 System Requirements

Android

Please install the following required packages:

- A development environment for running React Native projects using the React Native CLI (not the Expo CLI)
- The latest stable version of Android Studio
- The Android NDK
- An Android Virtual Device or a hardware device

Operating Environment Requirements:

- Android `minSdkVersion` of `21` or higher.
- ComPDFKit SDK 2.0.1 or higher.

iOS

Please install the following required packages:

- A development environment for running React Native projects using the React Native CLI (not the Expo CLI)
- The latest stable version of Xcode
- The latest stable version of CocoaPods. Follow the CocoaPods installation guide to install it.

Operating Environment Requirements:

- ComPDFKit SDK 2.0.1 or higher.
- React Native dependency to version 3.0.0 or higher.
- iOS 12.0 or higher.

2.2 Creating a New Project

Let's create a simple app that integrates ComPDFKit for React Native.

1. In the terminal app, change the current working directory to the location you wish to save your project. In this example, we'll use the `~/Documents/` directory:

   ```
   cd ~/Documents
   ```

2. Create the React Native project by running the following command:

   ```
   react-native init MyApp
   ```

3. In the terminal app, change the location of the current working directory inside the newly created project:

   ```
   cd MyApp
   ```

2.3 Installing the ComPDFKit Dependency

You can integrate the SDK in two ways:

- **Through ComPDFKit GitHub repo**:

  In `MyApp` folder, install `@compdfkit_pdf_sdk/react_native` by calling:

  ```
  yarn add github:ComPDFKit/compdfkit-pdf-sdk-react-native
  ```

- **Through ComPDFKit npm package:**

  In `MyApp` folder, install run the following commands:

  ```
  yarn add @compdfkit_pdf_sdk/react_native
  ```

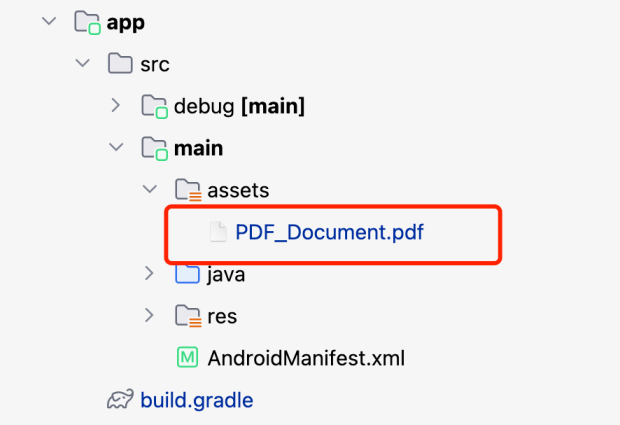
2.3.1 Android

1. Open `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.reactnative.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>
```

2. Copy the sample pdf file to the `assets` directory



2. Replace `App.js` (or `App.tsx`) with what is shown for Usage Example
3. Finally in the root project directory, run `react-native run-android`.

2.3.2 iOS

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

```
open ios/Podfile
```

2. Add the following line to the `target 'MyApp' do ... end` block:

```
target 'MyApp' do
    # ...
+  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
```

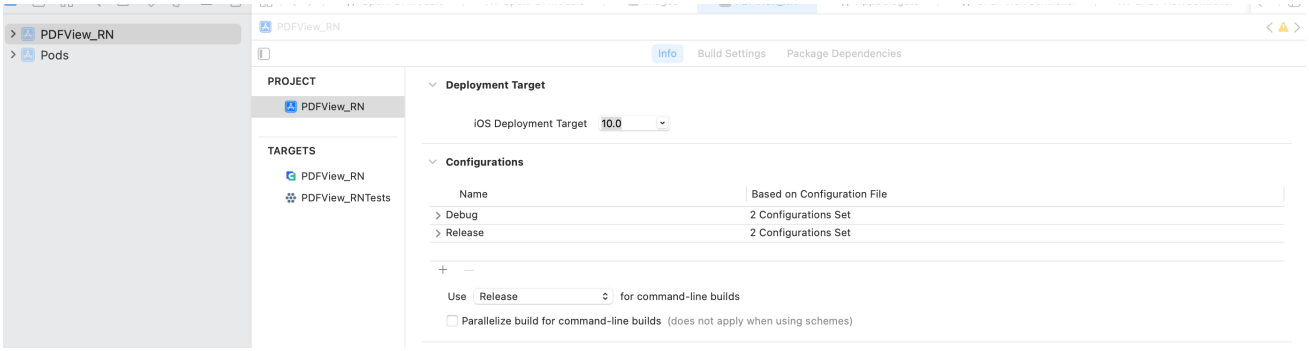
Note: If SSL network requests fail to download the `ComPDFKit` library when you run `pod install`, you can use the following method instead.

```
target 'MyApp' do
    # ...
+  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'
    # ...
end
```

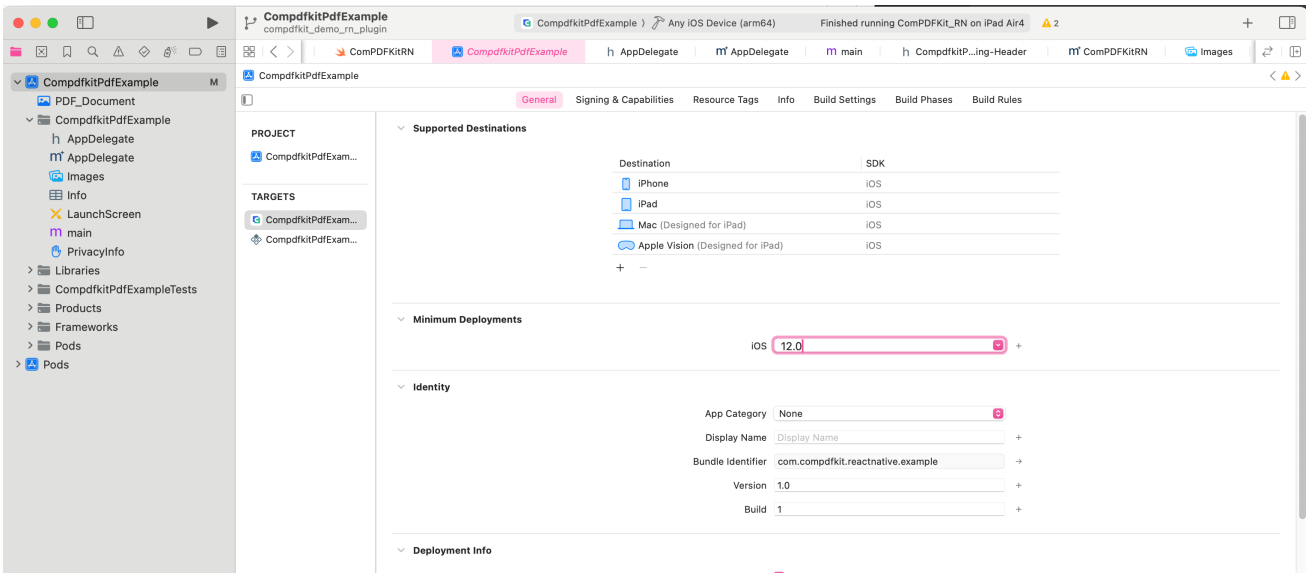
3. In the `ios` folder, run `pod install`.
4. Open your project's Workspace in Xcode:

   ```
   open ios/MyApp.xcworkspace
   ```

   Make sure the deployment target is set to 10.0 or higher:



5. Add the PDF document you want to display to your application by dragging it into your project. On the dialog that's displayed, select Finish to accept the default integration options. You can use **"PDF_Document.pdf"** as an example.




```
<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>
```

3. Replace `App.js` (or `App.tsx`) with what is shown for Usage Example
4. Finally in the root project directory, run `react-native run-ios`.

2.4 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:

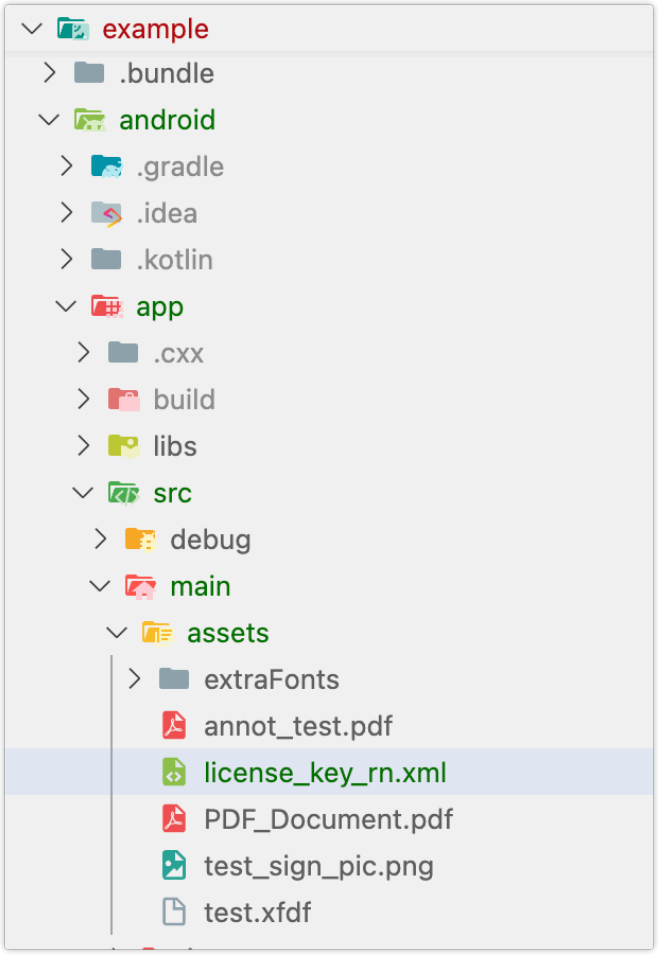
What is the authentication mechanism of ComPDFKit's license?

What are the differences between Online Authentication and Offline Authentication?

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

Android

1. In the email you received, locate the `XML` file containing the license key.
2. Copy the `license_key_rn.xml` file into the following directory: `android/app/src/main/assets/`

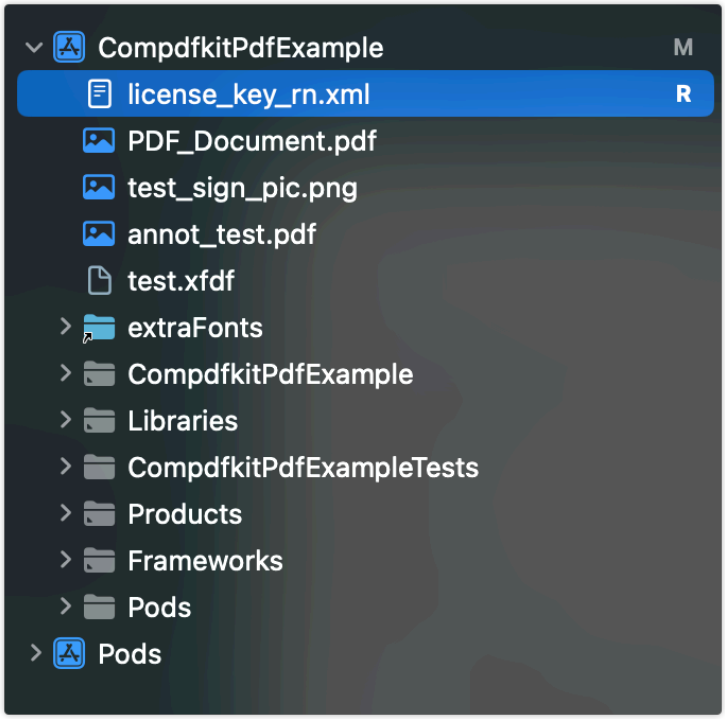


3. Initialize the SDK:

```
ComPDFKit.initWithPath('assets://license_key_rn.xml');
```

iOS

1. Use Xcode to copy the `license_key_rn.xml` file into your project's `ios/` directory.



2. Initialize the SDK:

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

Alternative Method

You can also store the License file in the device's local storage and initialize the SDK using its file path:

```
// Obtain the License file through the local storage path of the device for initialization
ComPDFKit.initWithPath('/data/data/0/your_packages/files/license_key_rn.xml');
```

2.5 Run Project

After installing from NPM or GitHub, replace `App.tsx` with the following code.

Make sure to follow the above steps to copy the sample document into your Android or iOS project.

Here is the sample code for `App.tsx`:

```
import React, { Component } from 'react';
import { SafeAreaView } from 'react-native';
import { ComPDFKit, CPDFReaderView } from '@compdfkit_pdf_sdk/react_native';
import { Platform } from 'react-native';

type Props = {};

export default class App extends Component<Props> {
```

```
  constructor(props: Props) {
    super(props)
    this.initialize()
  }

  async initialize() {
    var result = await ComPDFKit.initWithPath(Platform.OS == "android" ?
"assets://license_key_rn.xml" : "license_key_rn.xml")
    console.log("ComPDFKitRN", "init_:", result)
  }

  samplePDF = Platform.OS === 'android'
  ? 'file:///android_asset/PDF_Document.pdf'
  : 'PDF_Document.pdf';


  render() {
    return (
      <CPDFReaderView
        document={this.samplePDF}
        configuration={ComPDFKit.getDefaultConfig({})}
        style={{ flex: 1 }}
        />
    );
  }
}
```

- (Android) For local storage file path:

```
document = '/storage/emulated/0/Download/PDF_document.pdf'
```

- (Android) For content Uri:

```
document = 'content://...'
```

- (Android) For assets path:

```
document = "file:///android_asset/..."
```

- (iOS) For app bundle file path:

```
document = "document.pdf"
```

- (iOS) for URL path:

```
document = "file://xxxx/document.pdf"
```

The app is now ready to launch! Go back to the terminal app and run:

```
npx react-native run-android
npx react-native run-ios
```

3 Guides

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

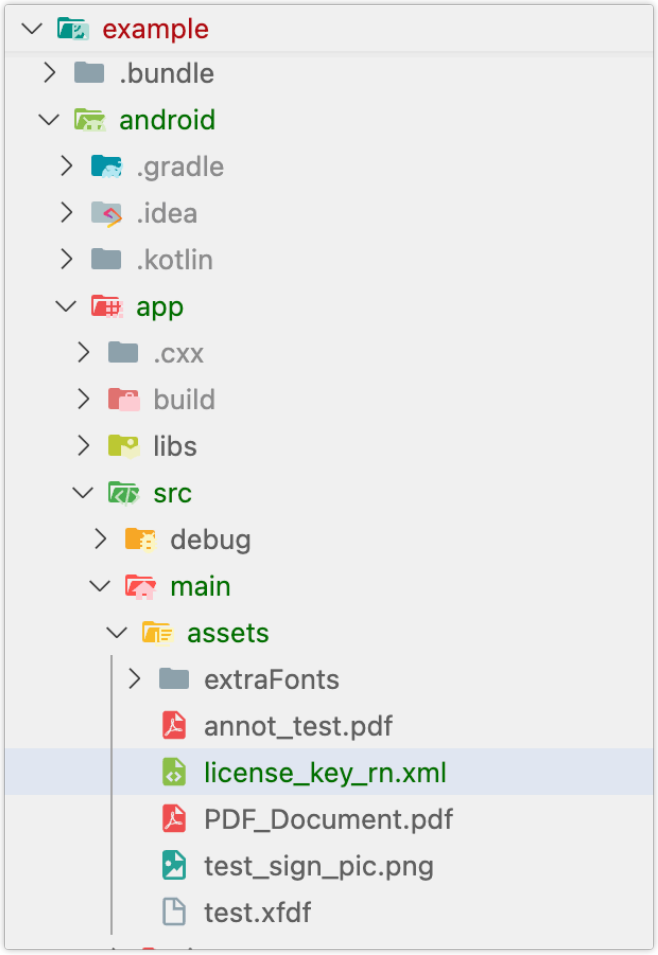
What is the authentication mechanism of ComPDFKit's license?

What are the differences between Online Authentication and Offline Authentication?

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

Android

1. In the email you received, locate the `XML` file containing the license key.
2. Copy the `license_key_rn.xml` file into the following directory: `android/app/src/main/assets/`

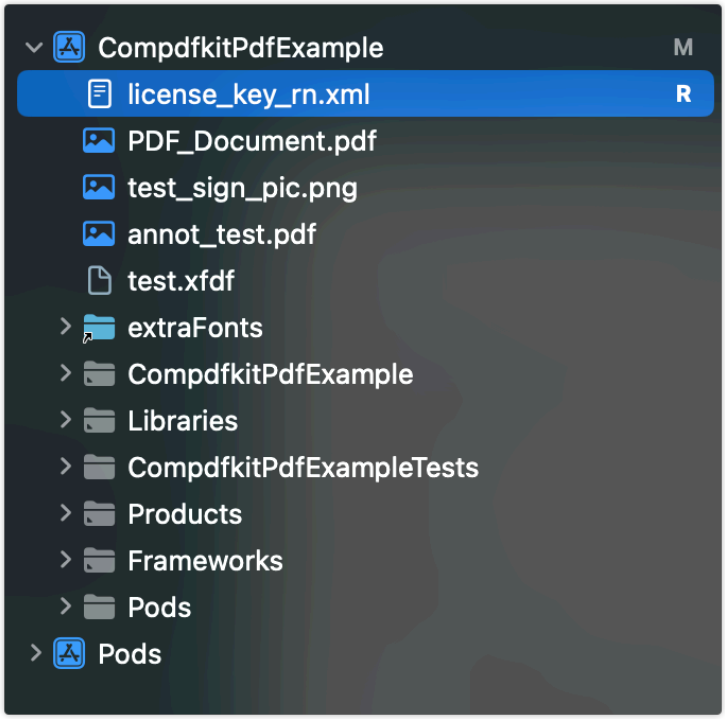


3. Initialize the SDK:

```
ComPDFKit.initWithPath('assets://license_key_rn.xml');
```

iOS

1. Use Xcode to copy the `license_key_rn.xml` file into your project's `ios/` directory.



2. Initialize the SDK:

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

Alternative Method

You can also store the License file in the device's local storage and initialize the SDK using its file path:

```
// Obtain the License file through the local storage path of the device for initialization
ComPDFKit.initWithPath('/data/data/0/your_packages/files/license_key_rn.xml');
```

3.1.3 Open a Document

ComPDFKit for React Native allows you to open documents in two ways: using a native UI component, and using a native module. This is a step-by-step guide to get you started quickly.

Opening a PDF

Here's how you can open a PDF document using the native module API:

```
import { Component } from "react";
import { ComPDFKit } from "@compdfkit_pdf_sdk/react_native";
import { Platform } from "react-native";

const _document: string = Platform.OS == 'android' ?
'file:///android_asset/PDF_Document.pdf' : 'PDF_Document.pdf';

type Props = {};
export default class ExampleApp extends Component<Props> {
```

```
  componentDidMount(): void {
    // Before using other ComPDFKit APIs, please initialize the ComPDFKit SDK
    var result = await ComPDFKit.initWithPath(Platform.OS == "android" ?
"assets://license_key_rn.xml" : "license_key_rn.xml")
    console.log("ComPDFKitRN", "init_:", result)
    ComPDFKit.openDocument(_document, '', ComPDFKit.getDefaultConfig({}));
  }

  render() {
    return null;
  }
}
AppRegistry.registerComponent('App', () => ExampleApp);
```

And here's how you can open a PDF document using the native component API:

```
import React, { Component } from 'react';
import { SafeAreaView } from 'react-native';
import { ComPDFKit, CPDFReaderView } from '@compdfkit_pdf_sdk/react_native';
import { Platform } from 'react-native';

type Props = {};

export default class App extends Component<Props> {

  constructor(props: Props) {
    super(props)
    this.initialize()
  }

  async initialize() {
    var result = await ComPDFKit.initWithPath(Platform.OS == "android" ?
"assets://license_key_rn.xml" : "license_key_rn.xml")
    console.log("ComPDFKitRN", "init_:", result)
  }

  samplePDF = Platform.OS === 'android'
  ? 'file:///android_asset/PDF_Document.pdf'
  : 'PDF_Document.pdf';

  render() {
    return (
        <CPDFReaderView
          document={this.samplePDF}
          configuration={ComPDFKit.getDefaultConfig({})}
          style={{ flex: 1 }}
        />
    );
  }
}
```

After opening a document with the `CPDFReaderView` UI component, you can reopen another PDF document within the same component via the API:

```
await pdfReaderRef.current?._pdfDocument.open(document, 'password');
```

3.1.4 Save Document

3.1.4.1 Manual Save

When using the `CPDFReaderView` component, the document is automatically saved during operations such as sharing, flattening, or adding watermarks. Additionally, you can manually save the document at any time. For example:

```
const CPDFReaderViewExampleScreen = () => {

  const pdfReaderRef = useRef<CPDFReaderView>(null);

  return (
    <View style={{flex : 1}}>
      <Button title='Save As' onPress={async ()=>{
          const success = await pdfReaderRef.current?.save();
        }}></Button>
      <CPDFReaderView
        ref={pdfReaderRef}
        document={samplePDF}
        configuration={ComPDFKit.getDefaultConfig({})}
        />
    </View>
  );
};
```

3.1.4.2 SaveAs

The CPDFReaderView component provides the saveAs method, allowing users to save the currently opened PDF document to a specified directory.

Example Usage

```
// Target file path (Android / iOS)
const savePath = '/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';

// URI format path supported on Android, e.g.:
const savePath = 'content://media/external/file/1000045118';

// Configuration options
const removeSecurity = false; // Whether to remove PDF document password
const fontSubset = true; // Whether to embed the fonts used in the PDF file

// Execute save operation
const result = await pdfReaderRef.current?._pdfDocument.saveAs(savePath, removeSecurity,
fontSubset);
```

3.1.4.3 Save Callback

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

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

const saveDocument = () => {
    console.log('ComPDFKitRN saveDocument');
}

<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={ComPDFKit.getDefaultConfig({
    })}/>
```

3.2 Viewer

3.2.1 Overview

ComPDFKit for React Native includes a fast, precise, and feature-rich high-quality PDF viewer. It provides developers with a quick and configurable way to integrate a PDF viewer into React Native applications on both Android and iOS platforms.

You can configure ComPDFKit page translation, scroll direction, and scroll mode in its `CPDFConfiguration` object.

To get started quickly, you can use `ComPDFKit.getDefaultConfig({})` to get the default configuration provided for you and then make changes

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 specific page 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.

PDF Page to 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 `ComPDFKit.openDocument()` or `CPDFReaderView` UI component, 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.

```
// ComPDFKit.openDocument Sample
var config = ComPDFKit.getDefaultConfig({
  modeConfig : {
    initialViewMode: CPDFViewMode.ANNOTATIONS
  }
})
ComPDFKit.openDocument(samplePDF, '', config)

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

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:

```
var config = ComPDFKit.getDefaultConfig({
  modeConfig : {
    availableViewModes: [
      CPDFViewMode.VIEWER,
      CPDFViewMode.ANNOTATIONS
    ]
  }
})
// ComPDFKit.openDocument Sample
ComPDFKit.openDocument(samplePDF, '', config)

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

The results are as follows:

Android	iOS
	

Switching View Modes:

When using the `CPDFReaderView` to display a PDF, you can programmatically switch the current view mode using the provided API.

```
// Switch to annotation mode
await pdfReaderRef.current?.setPreviewMode(CPDFViewMode.VIEWER);

// Get the current view mode
const mode = await pdfReaderRef.current?.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.

```
let config = ComPDFKit.getDefaultConfig({
  readerViewConfig: {
    verticalMode: true
  }
});
ComPDFKit.openDocument(samplePDF, '', config);

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
```

```
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

You can also set the scroll direction via the API:

```
await pdfReaderRef.current?.setVerticalMode(true);
```

Display Mode

The page display mode can be `SINGLE_PAGE`, `DOUBLE_PAGE`, or `COVER_PAGE`.

```
let config = ComPDFKit.getDefaultConfig({
  readerViewConfig: {
    displayMode: CPDFDisplayMode.DOUBLE_PAGE
  }
});
ComPDFKit.openDocument(samplePDF, '', config);

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

Set the display mode via the API:

- **singlePage**

```
await pdfReaderRef.current?.setDoublePageMode(false);
```

- **doublePage**

```
await pdfReaderRef.current?.setDoublePageMode(true);
```

- **coverPage**

```
await pdfReaderRef.current?.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.

```
let config = ComPDFKit.getDefaultConfig({
  readerViewConfig: {
```

```
    continueMode: true
  }
});
ComPDFKit.openDocument(samplePDF, '', config);

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

Set the scroll mode via the API:

```
await pdfReaderRef.current?.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.

```
let config = ComPDFKit.getDefaultConfig({
  readerViewConfig: {
    cropMode: true
  }
});
ComPDFKit.openDocument(samplePDF, '', config);

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

Set the crop mode via the API:

```
await pdfReaderRef.current?.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**.

```
let config = ComPDFKit.getDefaultConfig({
  readerViewConfig: {
    pageScale: 1.0
```

```
  }
});
ComPDFKit.openDocument(samplePDF, '', config);

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

Set the zoom level via the API:

```
await pdfReaderRef.current?.setScale(2.0);
```

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:

```
let config = ComPDFKit.getDefaultConfig({
  readerViewConfig: {
    themes: CPDFThemes.DARK
  }
});
ComPDFKit.openDocument(samplePDF, '', config);

// CPDFReaderView Sample
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
    ref={pdfReaderRef}
    document={samplePDF}
    saveDocument={saveDocument}
    configuration={config}/>
```

Set the background color via the API:

```
await pdfReaderRef.current?.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 displaying a PDF using the `CPDFReaderView` component, you can perform the following actions via the API:

Jump to a specific page

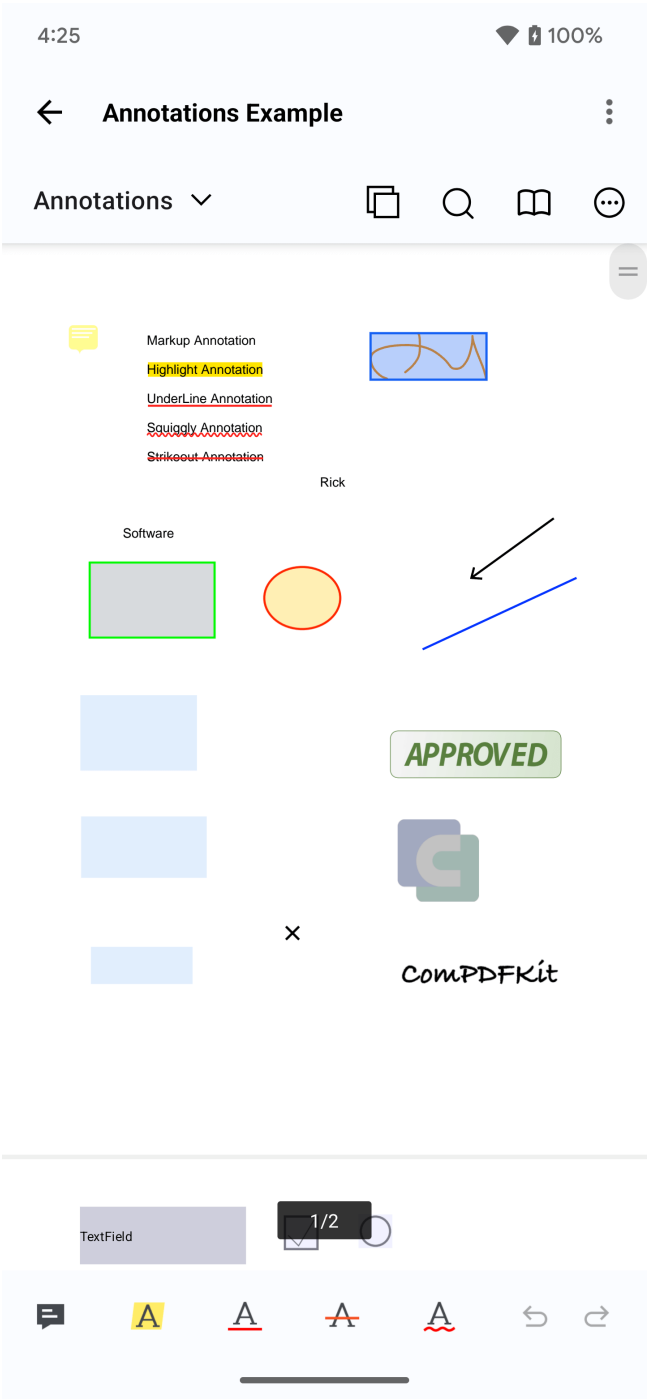
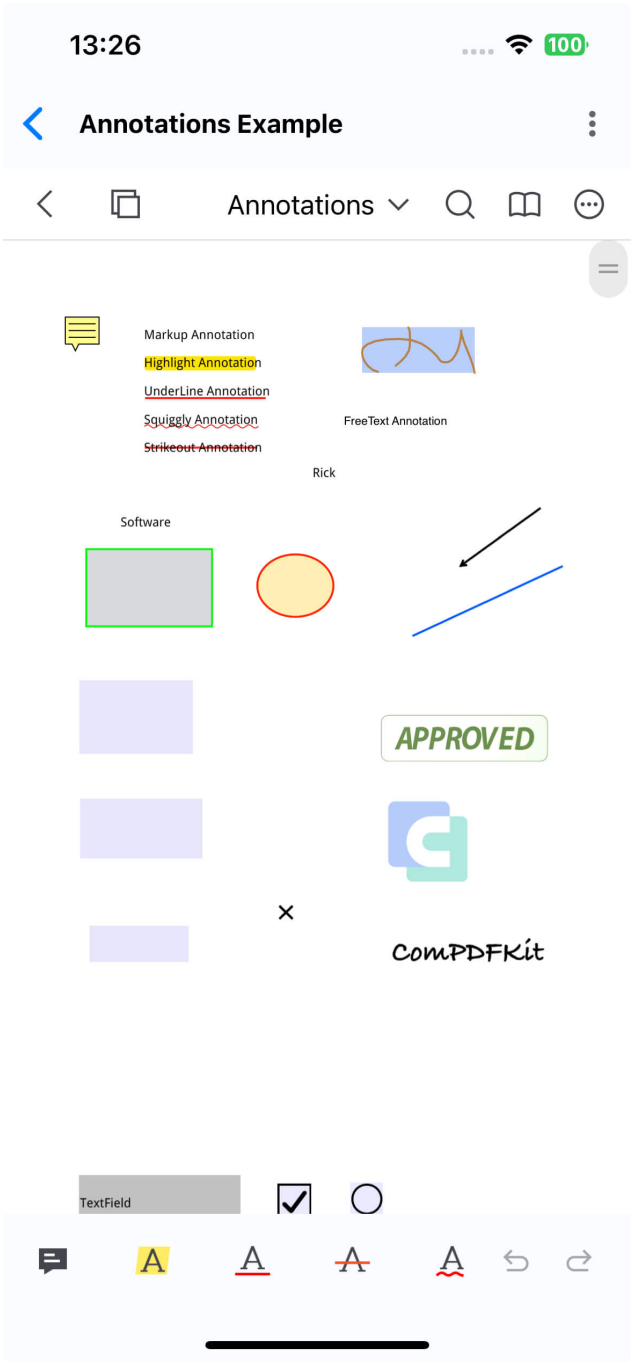
```
await pdfReaderRef.current?.setDisplayPageIndex(1);
```

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:

```
const pageIndex = 0;
const page = pdfReaderRef?.current?._pdfDocument.pageAtIndex(pageIndex);
const annotations = await page?.getAnnotations();
const annotation = annotations![0];

await pdfReaderRef.current?.setDisplayPageIndex(pageIndex: annotation.page, { rectList:
[annotation.rect!]});

// clear highlight area.
await pdfReaderRef.current?.clearDisplayRect();
```

Example Result:

Android	iOS
	

Get the current page number

```
const pageIndex = await pdfReaderRef.current?.getCurrentPageIndex();
```

You can also set a page listener when using `CPDFReaderView` to get the current page number in real-time as you scroll.

```
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  onPageChanged={onPageChanged}
  configuration={ComPDFKit.getDefaultConfig({
  })} />

const onPageChanged = (pageIndex: number) => {
  console.log('ComPDFKitRN --- onPageChanged:', pageIndex);
}
```

3.2.7 Highlight Form Fields and Hyperlinks

You can use the API provided by `CPDFConfiguration` or `CPDFReaderView` to highlight form fields and hyperlink annotations in the current document.

Highlighting PDF form fields helps users quickly locate and fill out forms, significantly improving efficiency in scenarios involving a lot of form filling.

The hyperlink highlighting feature allows users to add hyperlinks to key information in the PDF document, helping others quickly locate and understand the information. This not only improves the readability of the PDF document but also enhances its interactivity.

Here's an example of how to highlight form fields and hyperlinks:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);
<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  onPageChanged={onPageChanged}
  saveDocument={saveDocument}
  configuration={ComPDFKit.getDefaultConfig({
    readerViewConfig: {
      formFieldHighlight: true,
      linkHighlight: true
    }
  })} />
```

Use the provided API to highlight hyperlink annotations and form fields:

```
/// Enable hyperlink highlighting.
await pdfReaderRef.current?.setLinkHighlight(true);
/// Enable form field highlighting.
await pdfReaderRef.current?.setFormFieldHighlight(true);
```

3.2.8 Text Search and Selection

In the React Native 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`:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
/>

const document = pdfReaderRef.current?._pdfDocument;
const textSearcher = document!.textSearcher;
```

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
const results = await textSearcher!.searchText('keywords',
CPDFSearchOptions.CaseInsensitive);
```

> 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.length > 0) {
  await textSearcher.selection(results[0]);
}
```

4. Clear Search Results

To reset the search state and clear results:

```
await textSearcher.clearSearch();
```

5. Retrieve Contextual Text

With `textSearcher.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
const range = results[0];

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

// Retrieve contextual text from the corresponding page
const contextText = await textSearcher.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

```
ComPDFKit.getDefaultConfig({
  global: {
    search: {
      normalKeyword: {
        borderColor: '#00000000',
        fillColor: '#FFFF00AA',
      },
      focusKeyword: {
        borderColor: '#00000000',
        fillColor: '#FFA500AA',
      }
    }
  }
})
```

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

```
// Show search view
await pdfReaderRef.current?.showSearchTextView();

// Hide search view
await pdfReaderRef.current?.hideSearchTextView();
```

3.2.9 PDF Page to 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 base64-encoded, which can be used directly for display or saving as an image file.

Method Declaration

```
renderPage({
        pageIndex,
        width,
        height,
        backgroundColor = '#FFFFFF',
        drawAnnot = true,
        drawForm = true,
        pageCompression = CPDFPageCompression.PNG
    } : {
        pageIndex: number,
        width: number,
        height: number,
        backgroundColor?: HexColor,
        drawAnnot?: boolean,
        drawForm?: boolean,
        pageCompression?: CPDFPageCompression
    })
```

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
pageCompression	CPDFPageCompression	No	Rendering format, png, jpeg

Example Usage

```
// Get the size of the specified page
const size = await pdfReaderRef.current?._pdfDocument.getPageSize(pageIndex);
// Render the page as an image and return a base64 string
const image = await pdfReaderRef.current?._pdfDocument.renderPage({
  pageIndex,
  width: size.width,
```

```
  height: size.height,
  backgroundColor: '#FFFFFF',
  drawAnnot: true,
  drawForm: true,
});

// Displaying rendered images in React Native
<Image  source={{ uri: CPDFImageUtil.base64ToUri(image) }}
  style={{ width: '90%', height: '90%', resizeMode: 'contain', alignSelf: 'center' }}/>
```

3.3 Annotations

3.3.1 Overview

The ComPDFKit ReactNative 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

How to get all annotations in the current PDF document.

Create Annotation

Create various types of annotations through `CPDFReaderView`.

Import and Export

Managing comments in XFDF files.

Delete Annotations

How to delete all comments in the currently displayed document.

Flatten Annotations

Permanently embed existing annotations within 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

The steps to access a list of annotations and annotation objects are as follows:

1. Obtain the page object.
2. Access the list of annotations from the page object.
3. Iterate through each annotation object in the list.

This example shows how to access annotations:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

// Get the total number of pages in the PDF document
const pageCount = await pdfReaderRef!.current!._pdfDocument.getPageCount();
let allAnnotations: CPDFAnnotation[] = [];
for (let i = 0; i < pageCount; i++) {
  // Retrieve the page object for the corresponding page number
  const page = pdfReaderRef?.current?._pdfDocument.pageAtIndex(i);
  // Get all annotations on the page
  const annotations = await page?.getAnnotations();
  if (annotations) {
    allAnnotations = allAnnotations.concat(annotations);
  }
}
```

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.

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

await pdfReaderRef.current?.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 pdfReaderRef.current?.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:

```
const annotationMode = await pdfReaderRef.current?.getAnnotationMode();
```

3.3.4 Import and Export

XFDF is an XML-like standard from Adobe XFDF for encoding annotations and form field values. It's compatible with Adobe Acrobat and several other third-party frameworks.

ComPDFKit ReactNative SDK supports both reading and writing XFDF to import and export annotations. This guide shows how to import and export annotations using the XFDF format.

Import Annotations

You can import an XFDF file into the document by calling the `importAnnotations(xfdfFile)` method.

- The parameter `xfdfFile` specifies the path of the XFDF file to be imported.
- On the Android platform, this path can be:
    - A file Uri
    - A file path
    - A file located in the `assets` directory

Example code:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

const importResult = await pdfReaderRef.current?.importAnnotations('xxx.xfdf');
```

Export Annotations

You can export the current annotations in the document to an XFDF file by calling the `exportAnnotations()` method:

- This method does not require a path parameter. The system will automatically generate and return the exported XFDF file path.

Example code:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

const exportXfdfFilePath = await pdfReaderRef.current?.exportAnnotations();
```

3.3.5 Delete Annotations

The ComPDFKit React Native SDK provides APIs to delete selected annotations. The steps to delete annotations are as follows:

1. Obtain the document object.
2. Retrieve the page object where the annotation is located.
3. Get the list of annotations on that page.
4. Find the annotation you want to delete from the list.
5. Delete the specified annotation.

Example code:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

// Get the page object at a specific index
const page = pdfReaderRef?.current?._pdfDocument.pageAtIndex(0);
// Get all annotations on the page
const annotations = await page?.getAnnotations();
if (annotations[0]) {
  await page.removeAnnotation(annotations[0]);
}
// Or use this shortcut
pdfReaderRef?.current?._pdfDocument.removeAnnotation(annotations[0]);
```

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

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

Example code:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

const removeResult = await pdfReaderRef.current?.removeAllAnnotations();
```

> **Note**: This method does not delete hyperlink annotations.

3.3.6 Flatten Annotations

Annotation flattening refers to converting editable annotations into non-editable, non-modifiable static images or plain text forms. When the annotations are flattened, all editable elements of the entire document (including comments and forms) will be flattened, so the annotation flattening is also known as document flattening.

This example shows how to flatten annotations:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

const savePath = 'file:///storage/emulated/0/Download/flatten.pdf';
// or use Uri on the Android Platform.
const savePath = await ComPDFKit.createUri('flatten_test.pdf', 'compdfkit',
'application/pdf');
const fontSubset = true;
const result = await pdfReaderRef.current?._pdfDocument.flattenAllPages(savePath,
fontSubset);
```

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 `CPDFReaderView`:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
  })}/>

const historyManager = pdfReaderRef._annotationsHistoryManager;
```

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):

```
pdfReader._annotationsHistoryManager.setOnHistoryStateChangedListener((canUndo, canRedo)
=> {

});
```

Checking Whether Undo or Redo Is Available

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

```
const canUndo = await _annotationsHistoryManager.canUndo();
const canRedo = await _annotationsHistoryManager.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 _annotationsHistoryManager.undo(); // Undo the last annotation action
await _annotationsHistoryManager.redo(); // Redo the last undone annotation action
```

3.4 Forms

3.4.1 Overview

ComPDFKit ReactNative SDK provides functions for importing and exporting form data, allowing you to manage form data in documents. We are continuously enhancing these features to meet a wider range of use cases.

Guides

Access Forms

How to access all form data of the current document.

Import and Export

How to import and export all form data of the current document.

Create Form Fields

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

Fill Form Fields

Add content to form fields.

Delete Form Fields

Delete specific form fields.

3.4.2 Access Forms

Retrieves all form widgets on the current page.

This method fetches all form widgets present on the current page of the PDF document and returns a list of corresponding CPDFWidget instances.

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

// Get the total number of pages in the PDF document
const pageCount = await pdfReaderRef!.current!._pdfDocument.getPageCount();
let allWidgets: CPDFWidget[] = [];
for (let i = 0; i < pageCount1; i++) {
  // Retrieve the page object for the corresponding page number
  const page = pdfReaderRef?.current?._pdfDocument.pageAtIndex(i);
  // Get all forms on the page.
  const widgets = await page?.getWidgets();
  if (widgets) {
    allWidgets = allWidgets.concat(widgets);
  }
}
```

Related Widgets

Class	Description
`CPDFWidget`	Base class for all form widgets
`CPDFTextWidget`	Text input field widget
`CPDFSignatureWidget`	Signature widget
`CPDFRadiobuttonWidget`	Radio button widget
`CPDFPushbuttonWidget`	Button widget
`CPDFListboxWidget`	List box widget
`CPDFCheckboxWidget`	Checkbox widget
`CPDFComboboxWidget`	Combo box widget

3.4.3 Import and Export

ComPDFKit SDK supports importing and exporting form data in xfdf format to facilitate the management of data in PDF documents.

Import Forms

Imports the form data from the specified XFDF file into the current PDF document.

The API only imports form data and modifies the form content through the corresponding form name.

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

const xfdfFile = '/data/user/0/com.compdfkit.reactnative.example/xxx/xxx.xfdf';
// or use Uri on the Android Platform.
const xfdfFile = 'content://xxxx';
const result = await pdfReaderRef.current.importWidgets(xfdfFile);
```

Export Forms

exports the form data from the current PDF document to an XFDF file.

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

const exportXfdfFilePath = await pdfReaderRef.current?.exportWidgets();
```

3.4.4 Create Form Fields

Create Form Fields

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

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

await pdfReaderRef.current?.setFormCreationMode(CPDFWidgetType.TEXT_FIELD);
```

Exit Creation Mode

```
await pdfReaderRef.current?.exitFormCreationMode();
```

Supported Form Field Types

Type
CPDFWidgetType.TEXT_FIELD
CPDFWidgetType.CHECKBOX
CPDFWidgetType.RADIO_BUTTON
CPDFWidgetType.LISTBOX
CPDFWidgetType.COMBOBOX
CPDFWidgetType.SIGNATURES_FIELDS
CPDFWidgetType.PUSH_BUTTON

3.4.5 Fill Form Fields

`ComPDFKit` supports programmatically filling form fields in a PDF document.

The steps to fill in form fields using code are as follows:

1. Get the page object of the form to be filled in from CPDFDocument.
2. Retrieve all forms from the page object.
3. Traverse all forms to find the one to be filled in.
4. Modify the form field contents as needed.

This example shows how to fill in form fields:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

const pageIndex = 0;
// Retrieve the page object of the first page
const cpdfPage: CPDFPage = pdfReaderRef?.current?._pdfDocument.pageAtIndex(pageIndex);

// Retrieve all form widgets on the current page
const widgets = await page?.getWidgets();

// Fill in the text field content
// Assume that there is a text field form on the current page and retrieve the
CPDFTextWidget object
const textWidget = widgets[0] as CPDFTextWidget;
// Set the text field content to "Hello World"
```

```
await textWidget.setText('Hello World');
// Refresh the appearance of the form to apply changes, this step is necessary
await textWidget.updateAp();

// Modify the radio button's checked state
const radioButtonWidget = widgets[0] as CPDFRadiobuttonWidget;
// Set the radio button to checked
await radioButtonWidget.setChecked(true);
// Refresh the appearance of the radio button
await radioButtonWidget.updateAp();

// Modify the checkbox's checked state
const checkboxWidget = widgets[0] as CPDFCheckboxWidget;
// Set the checkbox to checked
await checkboxWidget.setChecked(true);
// Refresh the appearance of the checkbox
await checkboxWidget.updateAp();

// Add an electronic signature to the signature form
const signatureWidget = widgets[0] as CPDFSignatureWidget;
// Android-supported URI format:
await signatureWidget.addImageSignature('content://media/external/images/media/123');
// Or file path:
await signatureWidget.addImageSignature('/path/to/image');
// Refresh the appearance of the signature form
await signatureWidget.updateAp();
```

3.4.6 Delete Form Fields

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

1. Obtain the document object.
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.

Example Code:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

// Get the page object at a specific index
const page = pdfReaderRef?.current?._pdfDocument.pageAtIndex(0);
```

```
// Get all annotations on the page
const widgets = await page?.getWidgets();
if (widgets[0]) {
  await page.removeWidget(widgets[0]);
}
// Or use this shortcut
pdfReaderRef?.current?._pdfDocument.removeWidget(widgets[0]);
```

3.5 Document Editor

3.5.1 Overview

The document editing functionality offers a range of capabilities to manipulate pages, allowing users to control the document structure and adjust the layout and formatting, ensuring that the document content is presented accurately and in a well-organized manner.

Guides

Insert Pages

Inserts pages from another document into the target document.

Split Pages

Divide a portion of a multi-page document into independent documents.

3.5.2 Insert Pages

Insert a Blank Pages

Here is an example of inserting a blank page:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

const insertResult = await pdfReaderRef.current?._pdfDocument.insertBlankPage(1,
CPDFPageSize.a4);
```

Insert Pages from other PDFs

This example shows how to insert pages from other PDFs:

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
```

```
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

// Define the file path of the document to import
// For local files (e.g., from app cache):
const filePath = '/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';
// For Android content URIs (e.g., from media storage):
const filePath = 'content://media/external/file/1000045118';

// Specify the pages to import. An empty array [] imports all pages.
// In this example, only the first page (index 0) is imported.
const pages = [0];

// Define the position to insert the imported pages.
// 0 means inserting at the beginning of the document.
const insertPosition = 0;

// Provide the document password if encrypted. Leave empty if not required.
const password = '';

// Import the document into the PDF reader.
const importResult = await pdfReaderRef.current?._pdfDocument.importDocument(
  filePath,
  pages,
  insertPosition,
  password
);
```

3.5.3 Split Pages

Splits the specified pages from the current document and saves them as a new document.

This function extracts the given pages from the current PDF document and saves them as a new document at the provided save path.

```
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({})}
/>

const savePath = '/data/user/0/com.compdfkit.flutter.example/cache/temp/PDF_Document.pdf';
// Pages to extract from the current document
const pages = [0, 1, 2];
const result = await pdfReaderRef.current?.splitDocumentPages(savePath, pages);
```

3.6 Security

3.6.1 Overview

The security module provides password protection, permission settings, Bates encoding, background, header, and footer functions. By managing document passwords and permissions, and adding logos and copyright information, it ensures the security of documents.

Key Features

- **Password Management:** Set, modify, or remove document passwords and permissions.
- **Encryption Standards:** Supports 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, you can prevent unauthorized access and control user operations on the document.

Remove Password

Use the API to remove document passwords and permissions.

3.6.2 Set Password

The PDF permission module ensures document security by providing encryption, document permissions, decryption, and password removal functions. It allows users to securely control and manage the document.

Encryption

Encryption consists of two parts: user password and owner password.

- **User Password** is used to open the document, ensuring that only authorized users can access the content. When a user password is set, certain document permissions such as editing, copying, or printing are usually restricted.
- **Owner Password** allows not only to open the document but also to unlock all restricted permissions, enabling the user to edit, copy, or print the document.

This dual-password system is designed to offer more flexible and secure document access and management.

ComPDFKit provides various encryption algorithms and permission settings. Based on your needs, you can choose the appropriate algorithm and configure custom permissions to protect the document.

The encryption steps are as follows:

1. Set different user passwords and owner passwords.
2. Create permission information.
3. Specify the encryption algorithm.

4. Encrypt the document using the user password, owner password, permission information, and specified algorithm.
5. Save the document.

The following example demonstrates how to encrypt a document:

```
const CPDFReaderViewExampleScreen = () => {

  const pdfReaderRef = useRef<CPDFReaderView>(null);

  const [samplePDF] = useState(
    (Platform.OS === 'android'
     ? 'file:///android_asset/PDF_Document.pdf'
     : 'PDF_Document.pdf')
  );

  return (
    <View style={{ flex: 1 }}>
      <Button title='Set Password' onPress={async () => {

          var document = pdfReaderRef.current?._pdfDocument;
          const allowsPrinting = false;
          const allowsCopy = false;
          const result = await document?.setPassword(
            '1234', // Document opening password
            '4321', // Owner password
            allowsPrinting,
            allowsCopy,
            CPDFDocumentEncryptAlgo.AES128);
          console.log('ComPDFKit-RN setPassword:', result);

        }}></Button>
      <CPDFReaderView
        ref={pdfReaderRef}
        document={samplePDF}
        configuration={ComPDFKit.getDefaultConfig({
          toolbarConfig: {
            mainToolbarVisible: true,
            iosLeftBarAvailableActions: [
              CPDFToolbarAction.THUMBNAIL
            ]
          }
        })}
        />
    </View>
  );
};
```

Different Encryption Algorithms

Encryption Algorithm	Description	Enum Value
No Encrypt Algo	No encryption	`CPDFDocumentEncryptAlgo.noEncryptAlgo`
RC4	Key-based XOR encryption of plaintext	`CPDFDocumentEncryptAlgo.rc4`
AES-128	AES encryption with a 128-bit key	`CPDFDocumentEncryptAlgo.aes128`
AES-256	AES encryption with a 256-bit key	`CPDFDocumentEncryptAlgo.aes256`

3.6.3 Remove Password

Removing the password means the user with owner privileges removes the user password and owner password, allowing the document to be opened without a password and granting full access by default.

The steps to remove a password:

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

The following example demonstrates how to remove a password:

```
const CPDFReaderViewExampleScreen = () => {

  const pdfReaderRef = useRef<CPDFReaderView>(null);

  const [samplePDF] = useState(
    (Platform.OS === 'android'
     ? 'file:///android_asset/PDF_Document.pdf'
     : 'PDF_Document.pdf')
  );

  return (
    <View style={{ flex: 1 }}>
      <Button title='Remove Password' onPress={async () => {

          var document = pdfReaderRef.current?._pdfDocument;
          const removeResult = await document?.removePassword();

        }}></Button>
      <CPDFReaderView
        ref={pdfReaderRef}
        document={samplePDF}
        configuration={ComPDFKit.getDefaultConfig({
          toolbarConfig: {
            mainToolbarVisible: true,
            iosLeftBarAvailableActions: [
              CPDFToolbarAction.THUMBNAIL
            ]
          }
        })}
```

```
        />
      </View>
    );
};
```

Check if the Document is Encrypted:

```
var document = pdfReaderRef.current?._pdfDocument;
const isEncrypted = await document?.isEncrypted()
```

Get Document Permission Status:

```
var document = pdfReaderRef.current?._pdfDocument;
const permissions = await document?.getPermissions();
```

Permission Status Explanation:

Permission Status	Description	Enum Value
No Permissions	No permissions applied to the document	`CPDFDocumentPermissions.NONE`
User Permissions	Document opened with user password, some operations may be restricted	`CPDFDocumentPermissions.USER`
Owner Permissions	Document opened with owner password, no restrictions	`CPDFDocumentPermissions.OWNER`

Check Owner Permissions:

```
var document = pdfReaderRef.current?._pdfDocument;
// Check if the document is unlocked with owner privileges
const unlocked = await document?.checkOwnerUnlocked()

// Check if the owner password is correct
const result = await document?.checkOwnerPassword('4321')
```

3.7 Watermark

3.7.1 Overview

A watermark is a mark added to a PDF document, typically appearing as semi-transparent text or an image. Adding a watermark helps highlight document ownership or other relevant information without affecting readability.

Advantages of ComPDFKit Watermark

- **Copyright Protection**: Display document source and copyright information through watermarks to prevent screenshots or unauthorized use.

- **Brand Promotion**: Customize watermark styles to showcase brand logos or other preset content.
- **Quick UI Integration**: Easily integrate and customize watermark functionality using extensible UI components.

ComPDFKit Watermark Features

- Add Text Watermark

  Configure and add text watermarks through an interactive dialog.

- Add Image Watermark

  Configure and add image watermarks through an interactive dialog.

3.7.2 Add Text Watermark

ComPDFKit React Native SDK supports adding text watermarks through a UI dialog. You can configure default watermark styles (such as text, color, and font size) in CPDFConfiguration.

```
// Set default text watermark style in configuration
ComPDFKit.getDefaultConfig({
  global: {
    watermark: {
      types: ['text', 'image'],
      text: 'ComPDFKit-RN',
      textColor: '#000000',
      textSize: 36,
      rotation: -45,
      opacity: 255,
    }
  }
})

// Open the add watermark dialog via API
await pdfReaderRef.current?.showAddWatermarkView({
  types: ['text', 'image'],
  text: 'ComPDFKit RN',
  textColor: '#000000',
  textSize: 36,
  rotation: -45,
  opacity: 255,
});
```

3.7.3 Add Image Watermark

ComPDFKit React Native SDK supports adding image watermarks through a UI dialog. You can configure default watermark styles such as image source, opacity, and rotation angle in `CPDFConfiguration`.

```
// Set default image watermark style in configuration
ComPDFKit.getDefaultConfig({
  global: {
    watermark: {
```

```
      types: ['text', 'image'],
      image: 'ic_logo',
      opacity: 120,
      rotation: -45
    }
  }
});

// Open the add watermark dialog via API
await pdfReaderRef.current?.showAddWatermarkView({
  types: ['text', 'image'],
  image: 'ic_logo',
  opacity: 120,
  rotation: -45,
  opacity: 255,
});
```

To customize the default image, you need to handle it differently on Android and iOS.

Android

1. Add the image file to your resource directory:

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

2. Reference the image by its filename **without extension** in your code:

```
image: 'ic_logo'
```

iOS

1. Import the image file into your project through Xcode.

```
<img src="./screenshots/guides_rn_3.7.4_1.png" alt="guides_rn_3.7.4_1" style="zoom:50%;"
/>
```

2. Reference the image by its filename **without extension** in your code:

```
image: 'ic_logo'
```

Alternative Approach

You can also copy the image to the device's local storage and pass its file path directly:

```
// Assume imagePath is the local file path of the image
const imagePath = 'data/user/0/com.compdfkit.reactnative.example/cache/temp/ic_logo.png';

await pdfReaderRef.current?.showAddWatermarkView({
  types: ['text', 'image'],
  image: imagePath,
  opacity: 120,
  rotation: -45,
  opacity: 255,
});
```

3.8 UI Customization

3.8.1 Overview

Custom PDF Viewer in React Native

ComPDFKit PDF SDK for React Native 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.

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 the ComPDFKit PDF SDK for React Native SDK.

Tools

The toolset includes page editing dialogs, watermark addition dialogs, security settings, and other related APIs. Users can directly call and open the corresponding features via `CPDFReaderView` to implement these functionalities in React Native.

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.

3.7.2 Toolbars

3.7.2.1 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:

```
let config = ComPDFKit.getDefaultConfig({
  toolbarConfig: {
    // only ios platform
    iosLeftBarAvailableActions: [
      CPDFToolbarAction.BACK,
      CPDFToolbarAction.THUMBNAIL
    ],
    // only ios platform
    iosRightBarAvailableActions: [
      CPDFToolbarAction.SEARCH,
      CPDFToolbarAction.BOTA
```

```
    ],
    // only android platform
    androidAvailableActions: [
      // The back button will only appear on the far left side of the toolbar
      CPDFToolbarAction.BACK,
      CPDFToolbarAction.SEARCH,
      CPDFToolbarAction.THUMBNAIL,
      CPDFToolbarAction.BOTA,
      CPDFToolbarAction.MENU
    ]
  }
});
ComPDFKit.openDocument(_document, '', config);
```

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 `CPDFToolbarAction` for the relevant options.

Available menu Customization options

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

Menu Button Item	Description
VIEW_SETTINGS	Open the settings view and set the scrolling direction, display mode, theme color and other related settings for reading PDF.
DOCUMENT_EDITOR	Open the document thumbnail list, and you can delete, rotate, and add document pages in the view.
DOCUMENT_INFO	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.
OPEN_DOCUMENT	Open the system file selector and open a new pdf document.

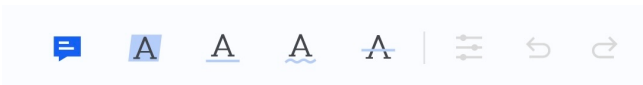
> Note: Please refer to `CPDFToolbarMenuAction` for the relevant options.

3.7.2.2 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
	

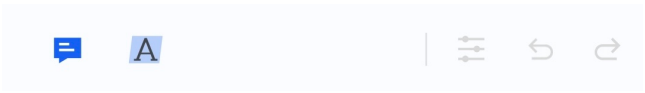
Customizing the Annotation Toolbar Buttons

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.

```
let config = ComPDFKit.getDefaultConfig({
  annotationsConfig: {
```

```
    availableTypes: [
      CPDFAnnotationType.NOTE,
      CPDFAnnotationType.HIGHLIGHT
    ],
    availableTools: [
      CPDFConfigTool.SETTING,
      CPDFConfigTool.UNDO,
      CPDFConfigTool.REDO,
    ]
  }
});
ComPDFKit.openDocument(_document, '', config);
```

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

Show or Hide Toolbar

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

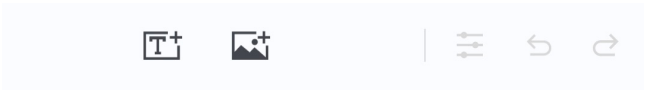
```
let config = ComPDFKit.getDefaultConfig({
  toolbarConfig: {
    annotationToolbarVisible: true // or false to hide the toolbar
  }
});
```

3.7.2.3 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.

```
let config = ComPDFKit.getDefaultConfig({
  contentEditorConfig: {
    availableTypes: [
      CPDFContentEditorType.EDITOR_IMAGE,
      CPDFContentEditorType.EDITOR_TEXT
    ],
    availableTools: [
      CPDFConfigTool.UNDO,
      CPDFConfigTool.REDO,
    ]
  }
});
ComPDFKit.openDocument(_document, '', config);
```

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

Android	iOS
	

Available Content Editor Toolbar Customization Options

Type
EDITOR_TEXT
EDITOR_IMAGE

> 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

Show or Hide Toolbar

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

```
let config = ComPDFKit.getDefaultConfig({
  toolbarConfig: {
    contentEditorToolbarVisible: false // or true to show the toolbar
  }
});
```

3.7.2.4 Form 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.

```
let config = ComPDFKit.getDefaultConfig({
  formsConfig: {
    availableTypes: [
      CPDFFormType.TEXT_FIELD,
      CPDFFormType.LISTBOX
    ],
    availableTools: [
      CPDFConfigTool.UNDO,
      CPDFConfigTool.REDO,
    ]
  }
});
ComPDFKit.openDocument(_document, '', config);
```

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

Android	iOS
	

Available Form Toolbar Customization Options

Type
TEXT_FIELD
CHECKBOX
RADIO_BUTTON
LISTBOX
COMBOBOX
SIGNATURES_FIELDS
PUSH_BUTTON

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

Available Form Toolbar Tool Customization Options

Tool	Description
UNDO	Undo annotation, content editing, form operations.
REDO	Redo an undone action.

Show or Hide Toolbar

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

```
let config = ComPDFKit.getDefaultConfig({
  toolbarConfig: {
    formsToolbarVisible: false // or true to show the toolbar
  }
});
```

3.8.6 Signature Toolbar

Show or Hide Toolbar

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

```
let config = ComPDFKit.getDefaultConfig({
  toolbarConfig: {
    signatureToolbarVisible: false // or true to show the toolbar
  }
});
```

3.8.3 Configuration

When displaying a PDF, customization can be achieved through `CPDFConfiguration`, encompassing UI views and various PDF property values. For detailed information, please refer to CONFIGURATION.md. This document is applicable across iOS, Android, Flutter, and React Native platforms.

3.8.4 Localization

The ComPDFKit React Native SDK supports the following languages by default:

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

Adding Additional Localization to ComPDFKit

- **Android Platform**

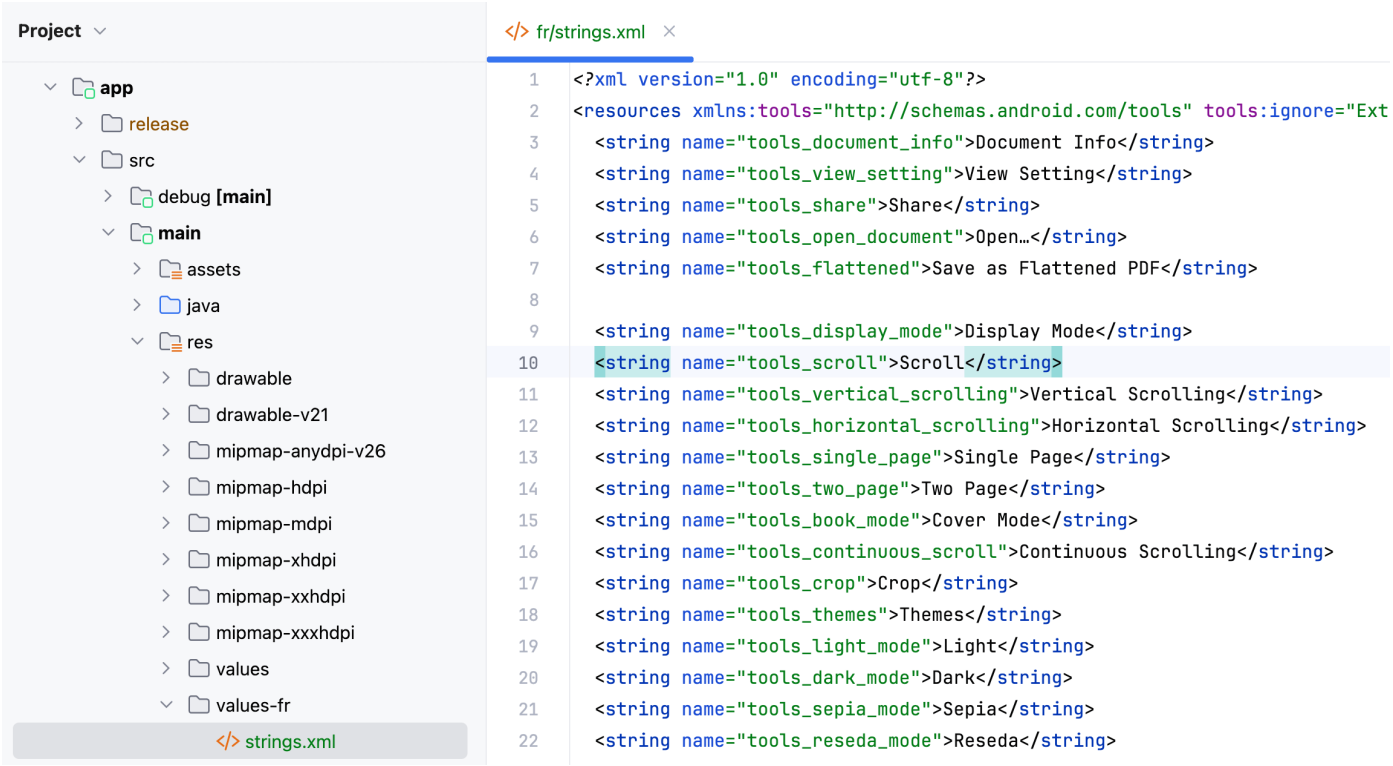
You can add additional translations by putting them into the `res/strings-XX` directory of your app. Android will automatically merge all string resources at build time. You can also override ComPDFKit strings of existing languages by putting them into the respective string folders.

> 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 support:

1. Create a `app/src/main/res/values-fr` directory in your project and create a `strings.xml` file in that directory.

2. Copy the content from the link above into the `strings.xml` file.



3. Translate the strings in the file to French. For example:

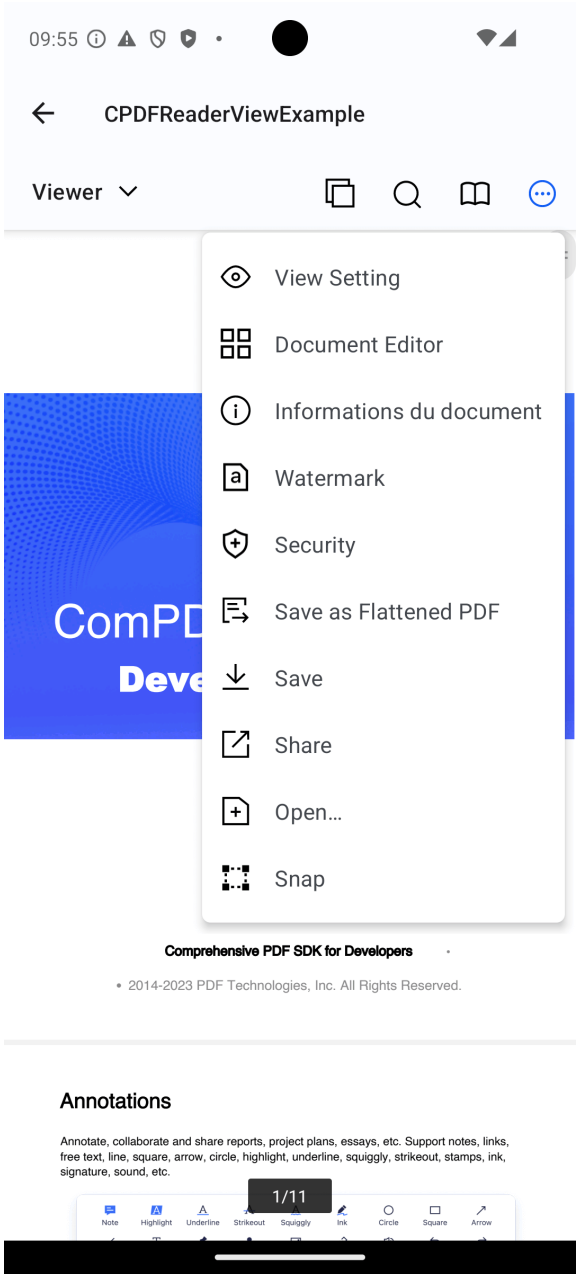
Original content:

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

Translated to French:

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

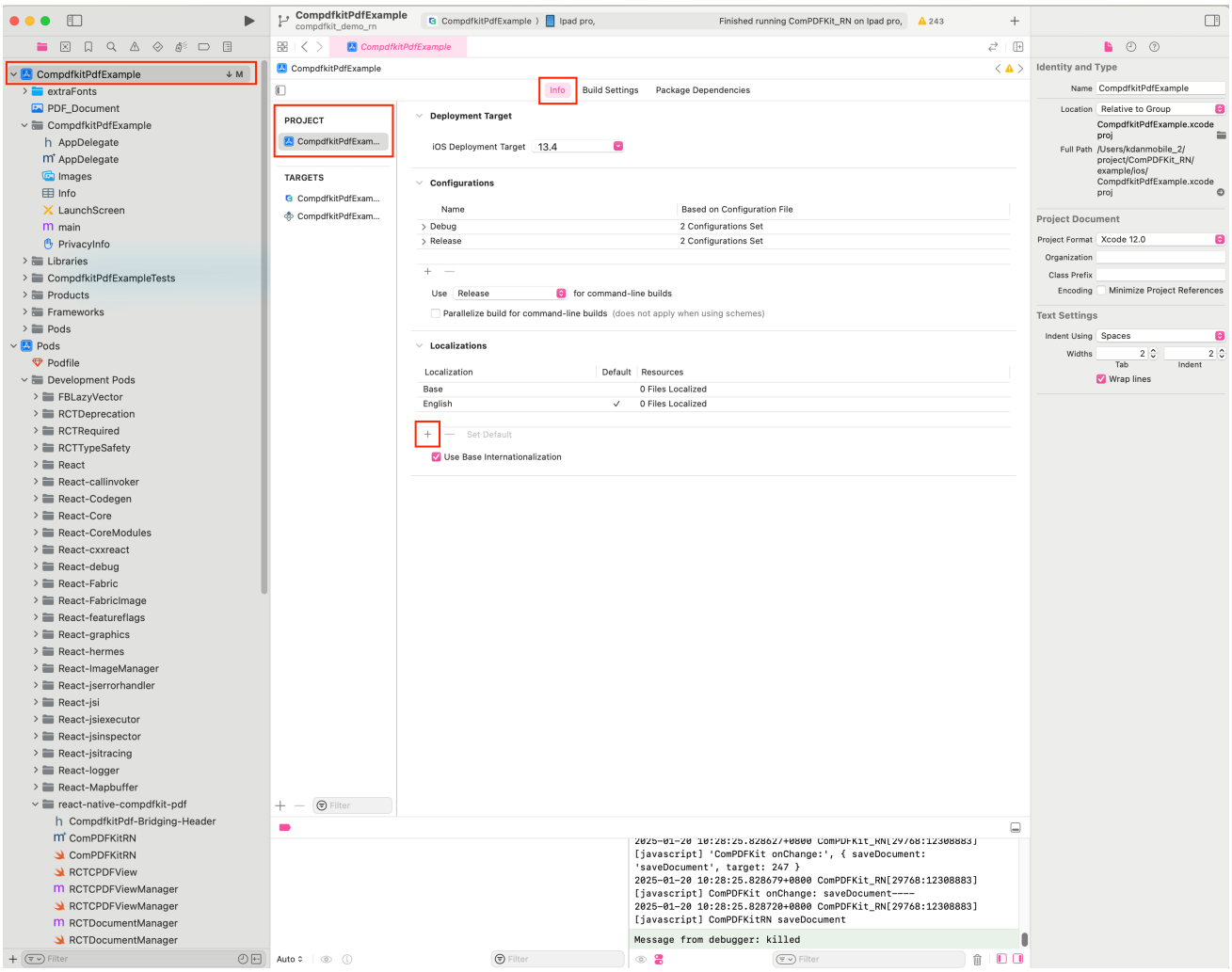
4. Switch your device to French, run the project, and you will see the translated content:



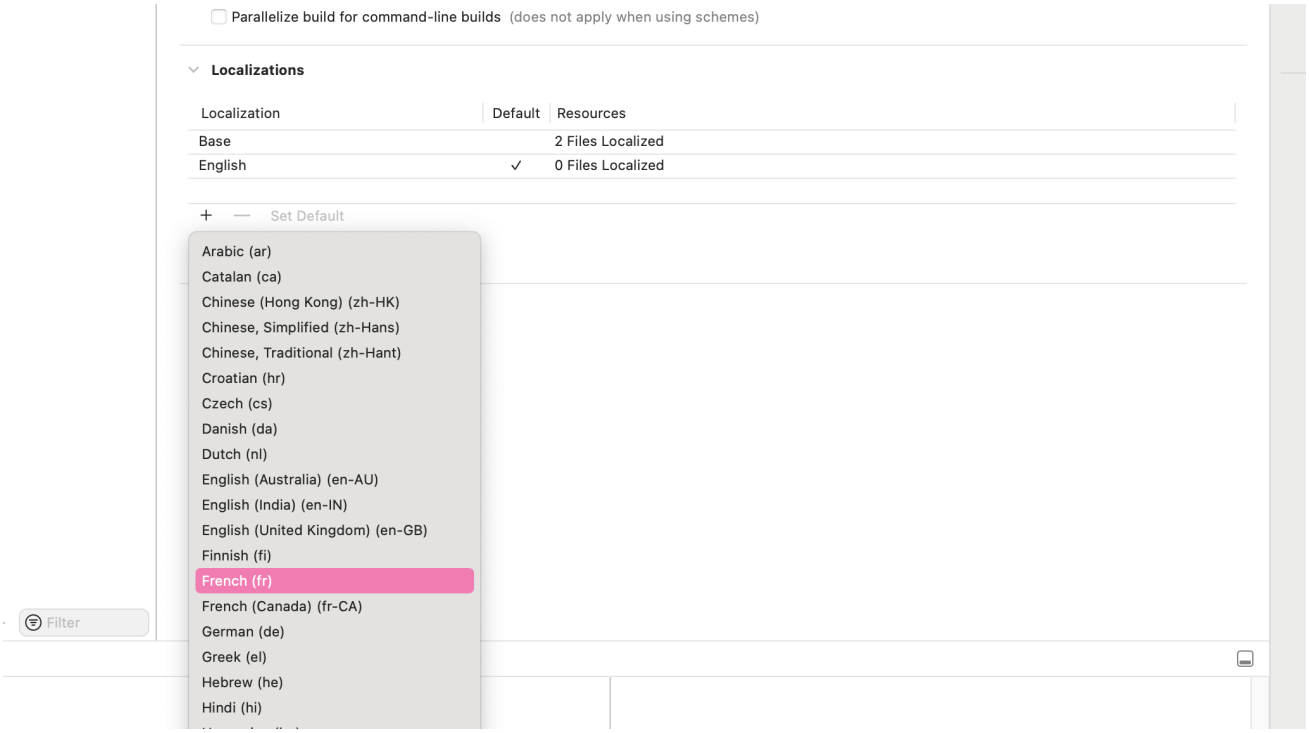
- **iOS Platform**

Configure the languages that require internationalization

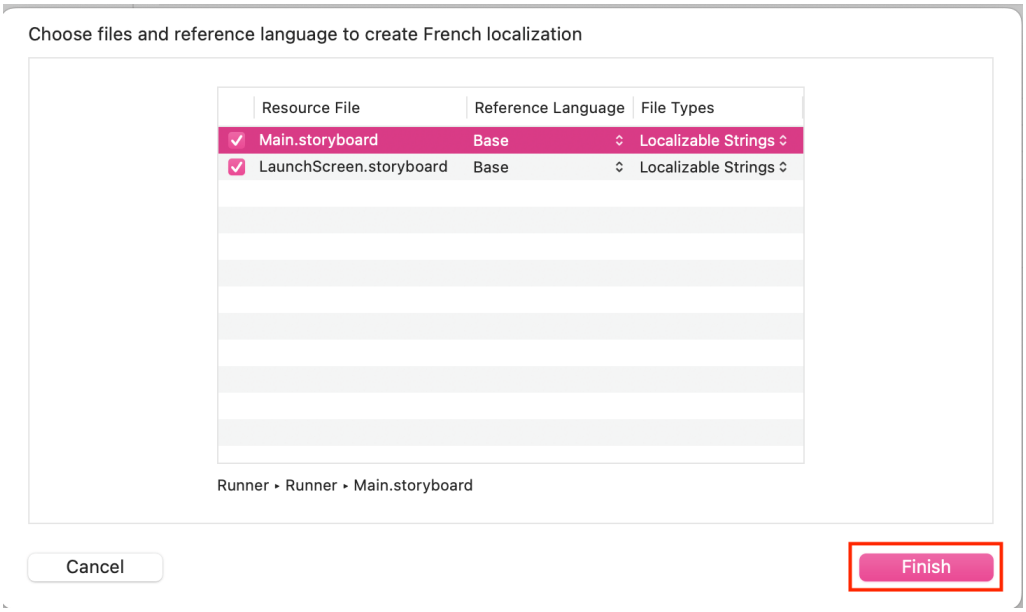
1. Open your Flutter project's iOS project using Xcode.Then, Select project -> Info -> Localizations, then click '+', add the desired language for internationalization/localization, as shown below (Make sure to check 'Use Base Internationalization' by default):



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



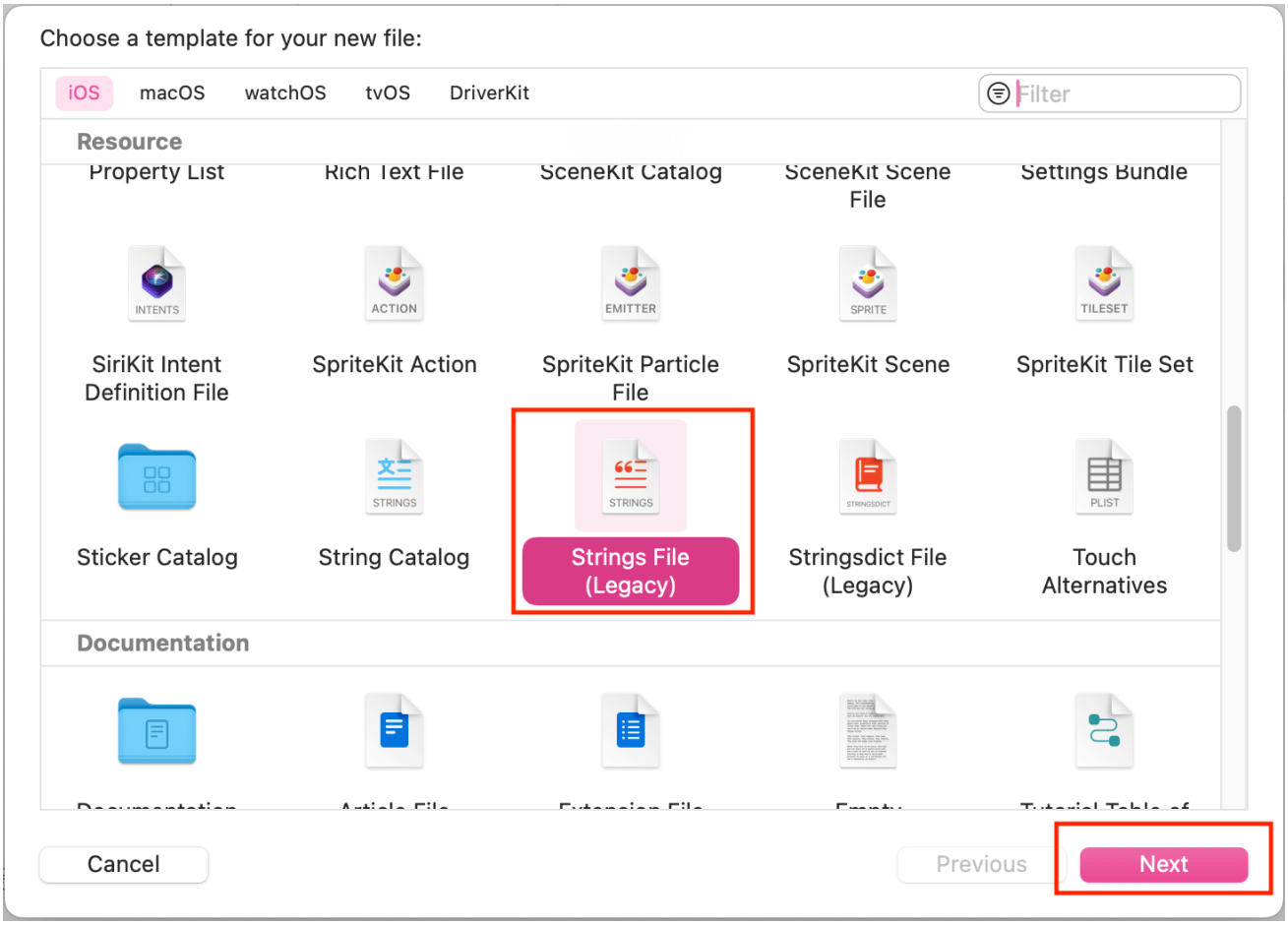
3. Pop up the following dialog, and click 'Finish' directly, as shown below:



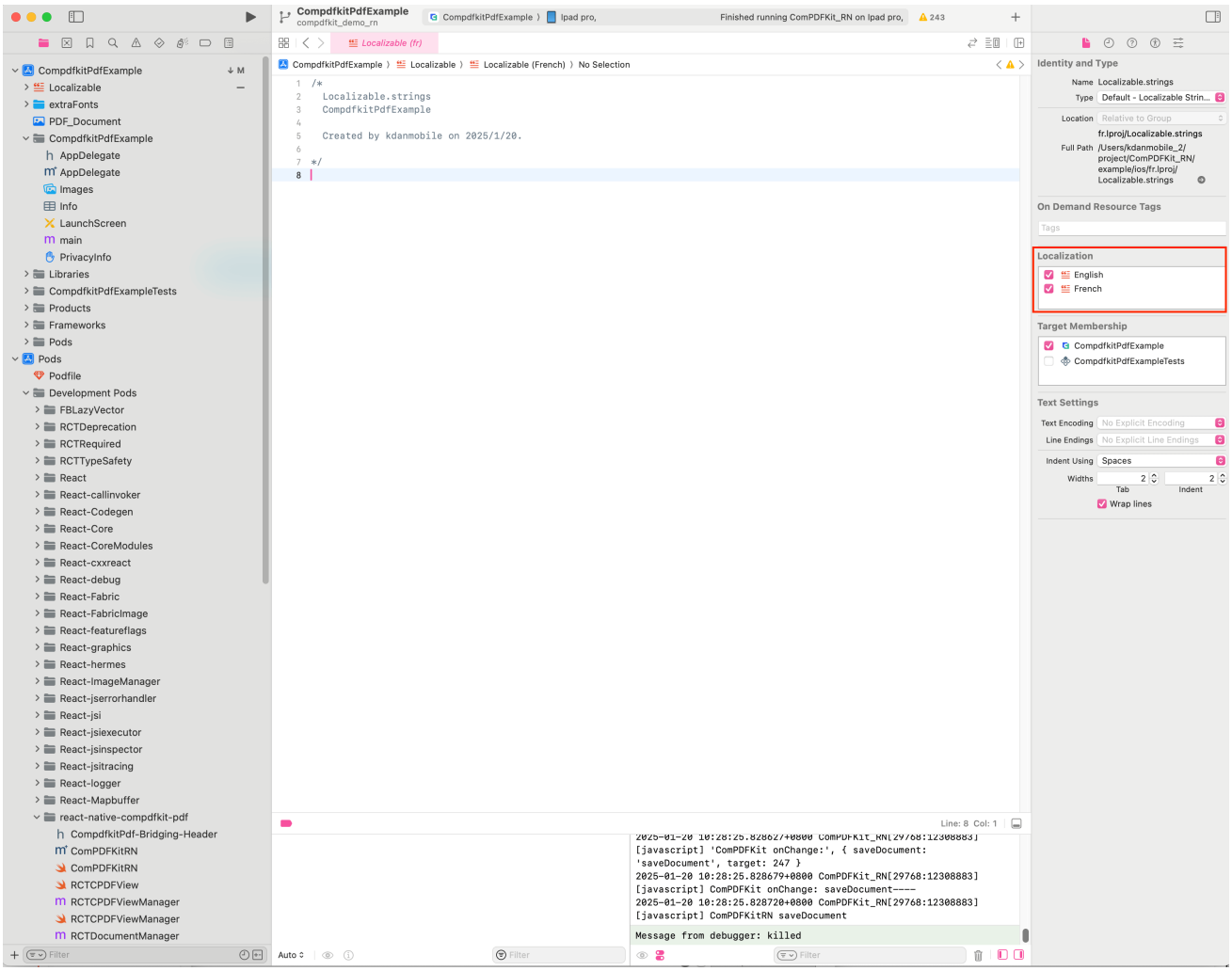
Internationalizing the application name

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).

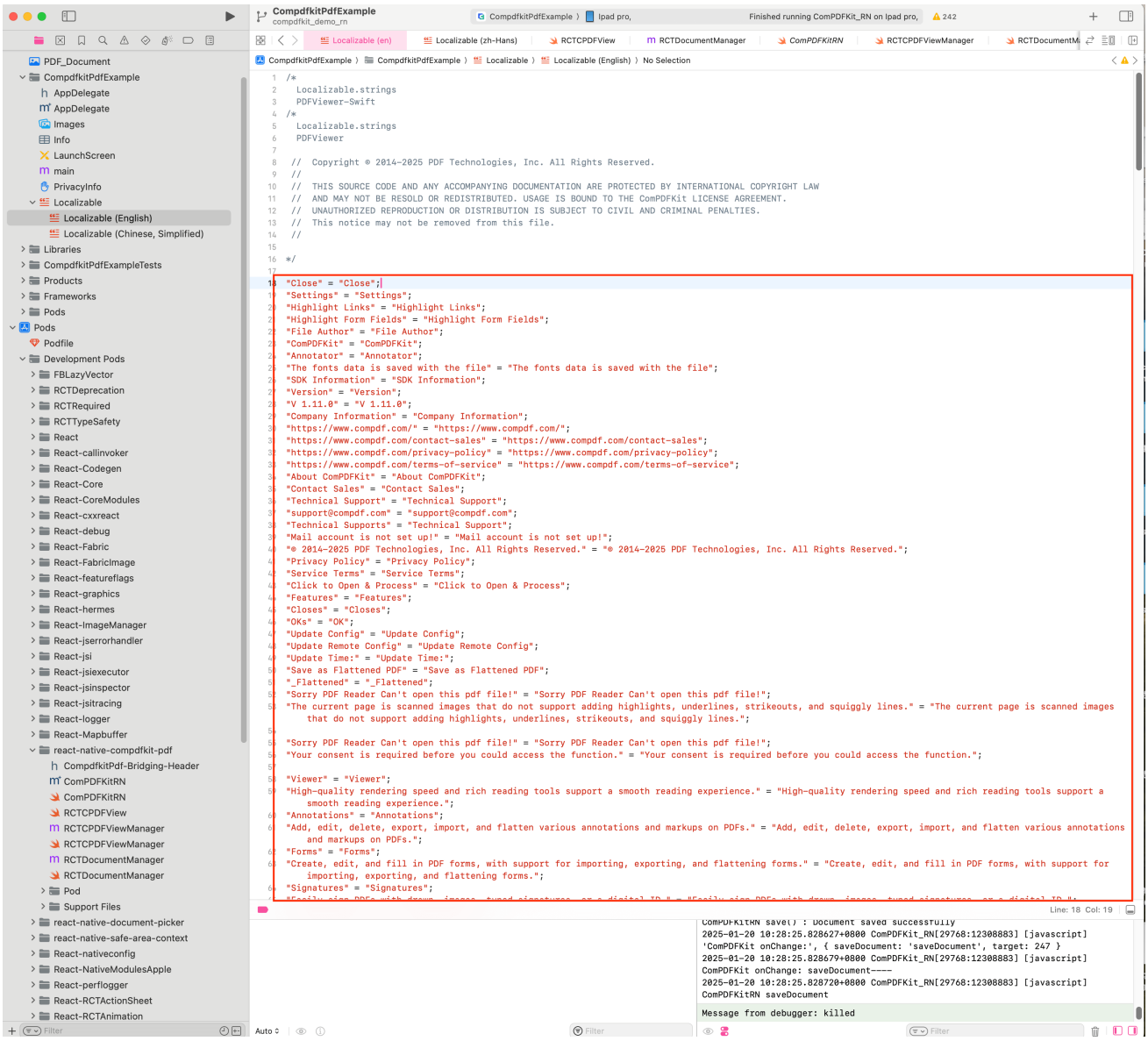
1. Creating `Localizable.strings` file



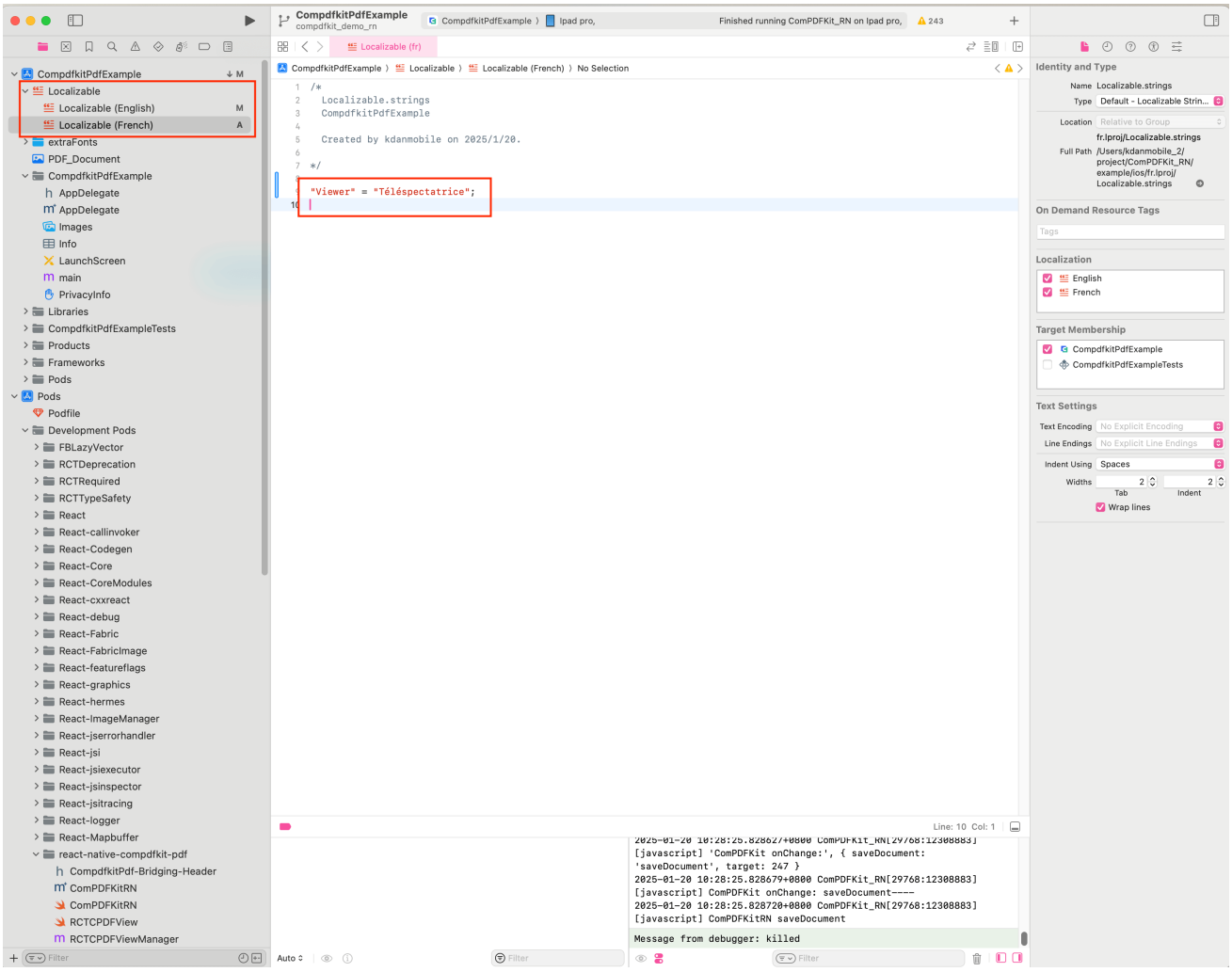
2. Select `Localizable.strings` , click on Localize in the File inspection (right-side file inspector) in Xcode. Next, check French and English, as shown below:



3. 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";`.



3.8.5 Tools

This section explains how to use the API provided by `CPDFReaderView` to directly access features such as the page editing view, watermark addition view, security settings view, and more. By displaying PDF documents through `CPDFReaderView` in React Native, it offers users a more flexible experience.

Open Page Editing View

```
// Control whether to enter page editing mode; when false, display the thumbnail list
const editMode = true;
await pdfReaderRef.current?.showThumbnailView(true);
```

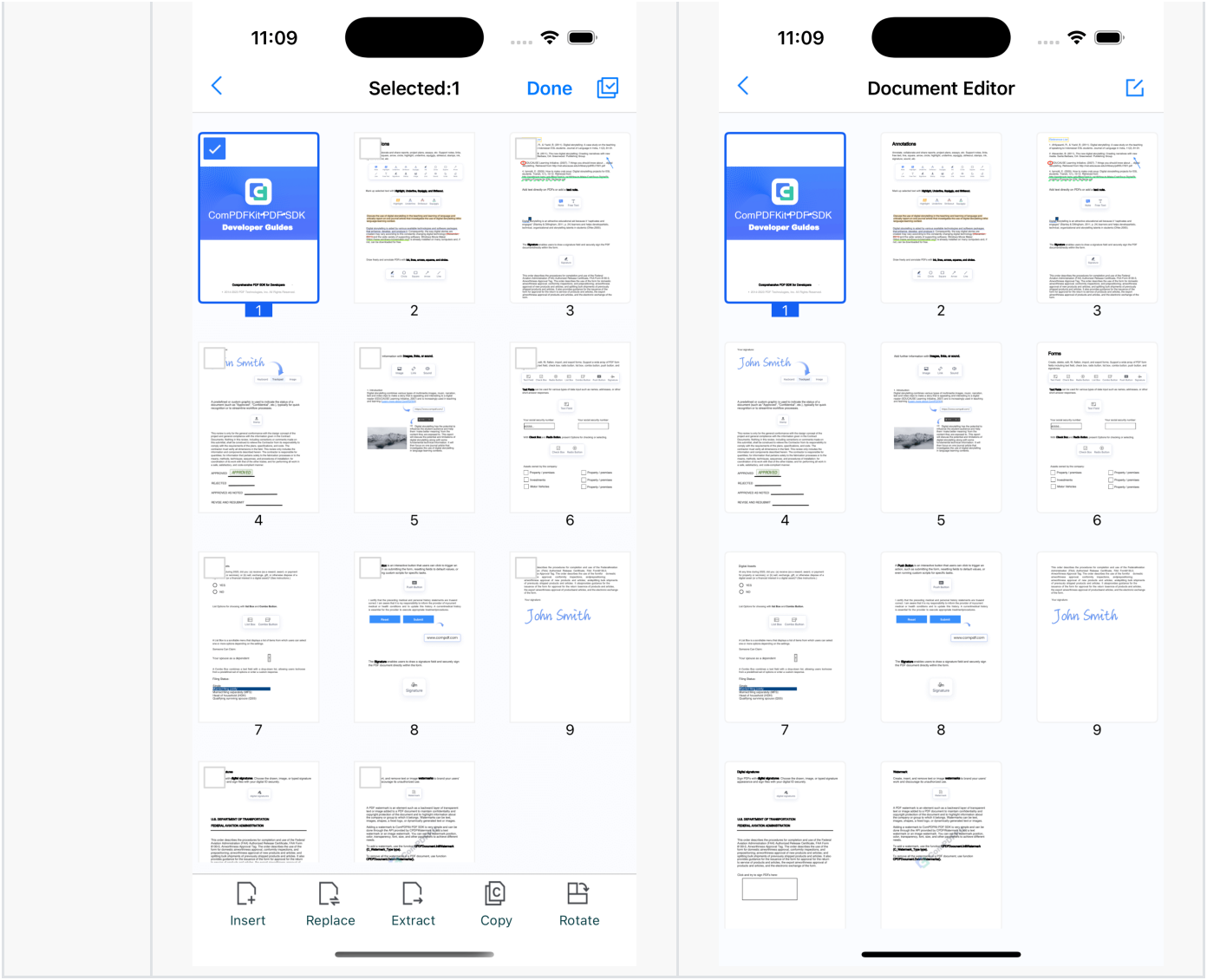
	editMode:true	editMode:false

Android


10:57
Document Editor
Done
Insert
Replace
Extract
Copy
Rotate


10:57
Document Editor

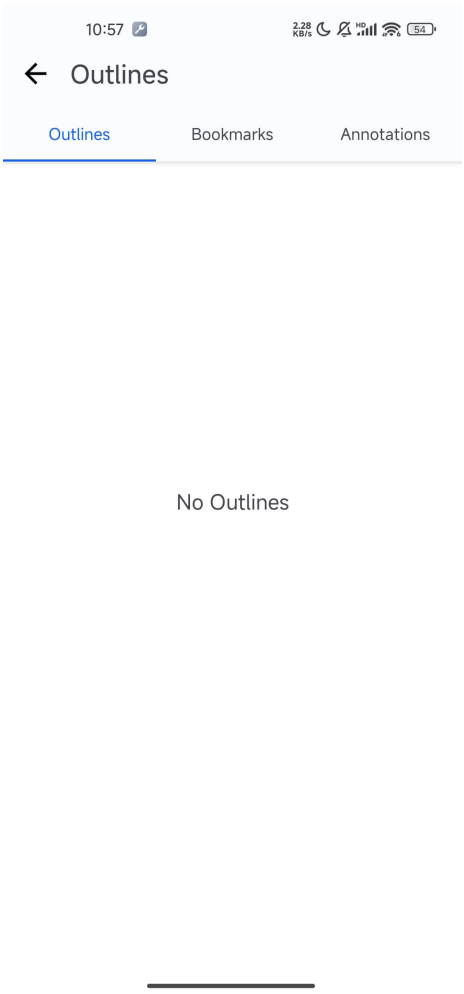
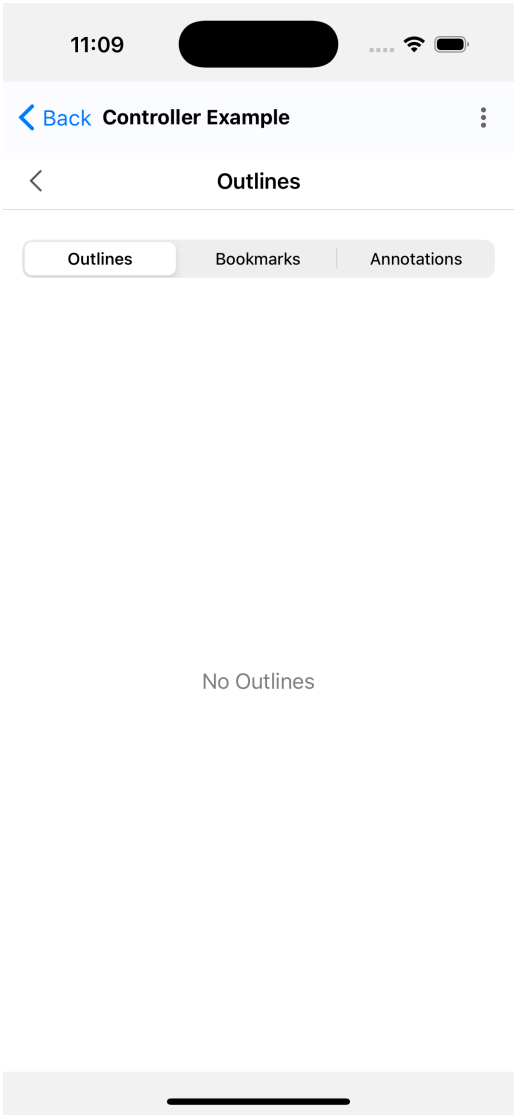
iOS



Open BOTA View

This view displays the document outline, bookmarks, and annotation list.

```
await pdfReaderRef.current?.showBotaView();
```

Android	iOS
	

Open Add Watermark Dialog

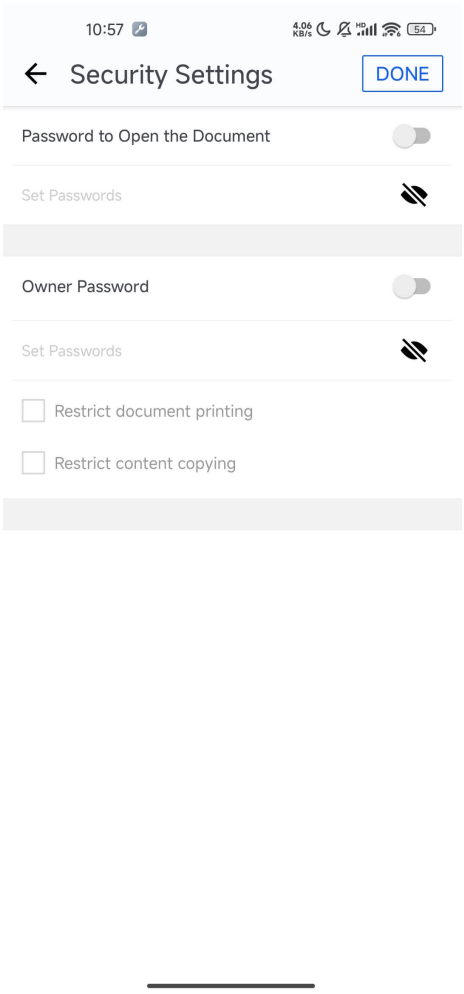
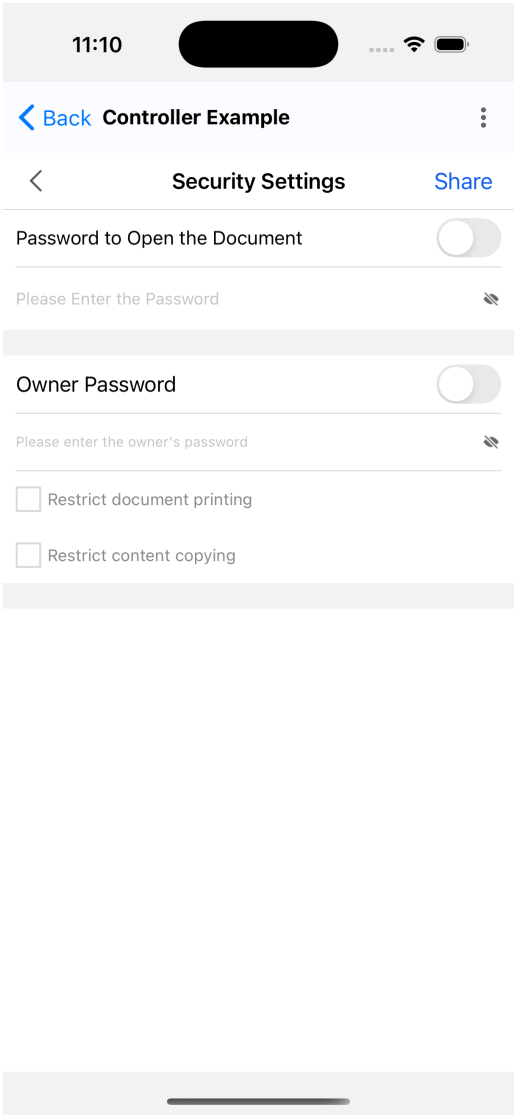
```
await pdfReaderRef.current.showAddWatermarkView();
```

Android	iOS
	

Open Security Settings View

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

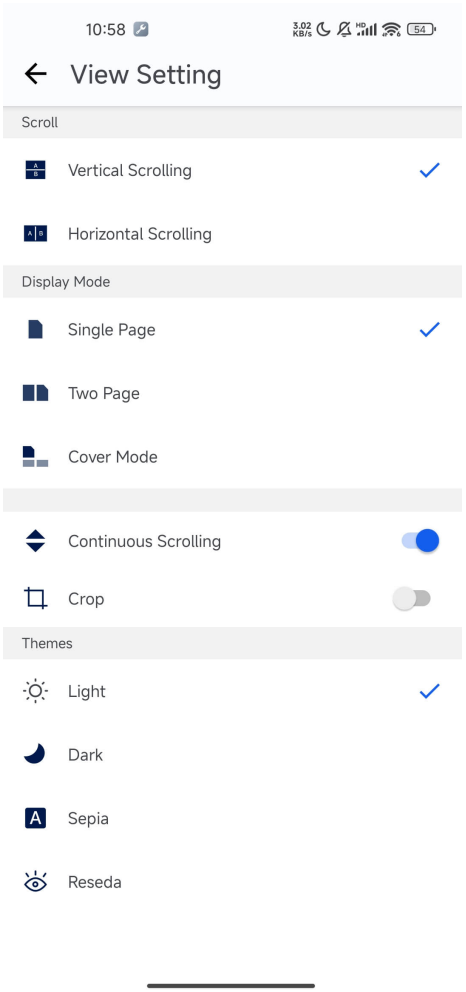
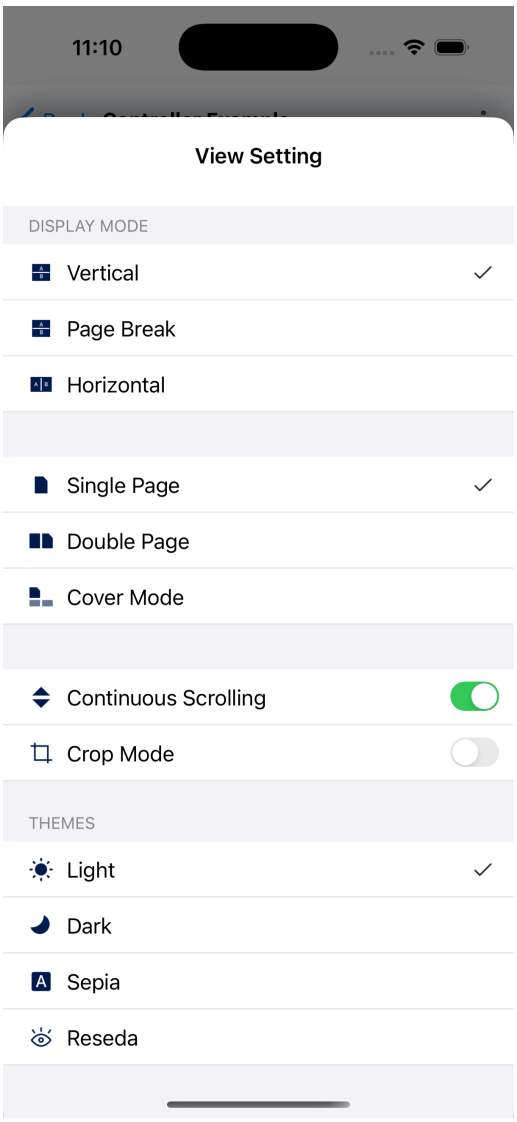
```
await pdfReaderRef.current.showSecurityView();
```

Android	iOS
	

Open Display Settings View

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

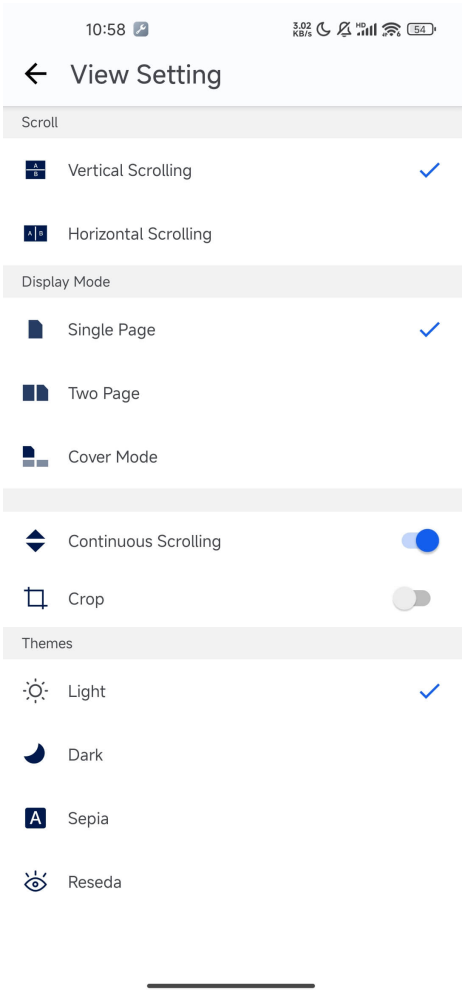
```
await pdfReaderRef.current.showDisplaySettingView();
```

Android	iOS
	

Screenshot Feature

You can enter screenshot mode while displaying PDF documents using the API to capture specific areas.

```
// Enter screenshot mode
await pdfReaderRef.current.enterSnipMode();
// Exit screenshot mode
await pdfReaderRef.current.exitSnipMode();
```

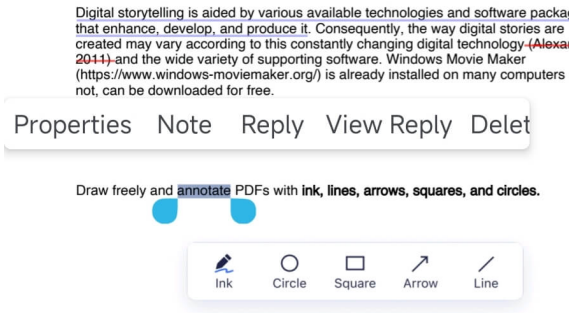
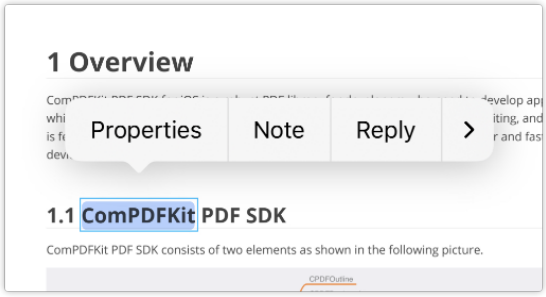
Android	iOS
	

3.8.6 Context Menu

When creating a `CPDFReaderView`, 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

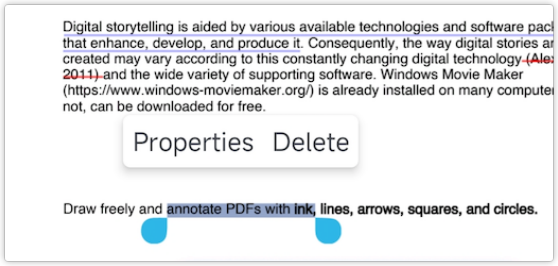
By default, when a user selects a **highlight annotation**, ComPDFKit displays a context menu with common options such as “Note”, “Delete”, and “Properties”:

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:

```
<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
    contextMenuConfig:{
      annotationMode: {
        markupContent: menus('properties', 'delete')
      }
    }
  })} />
```

Android	iOS
	

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

3.8.7 UI Visibility

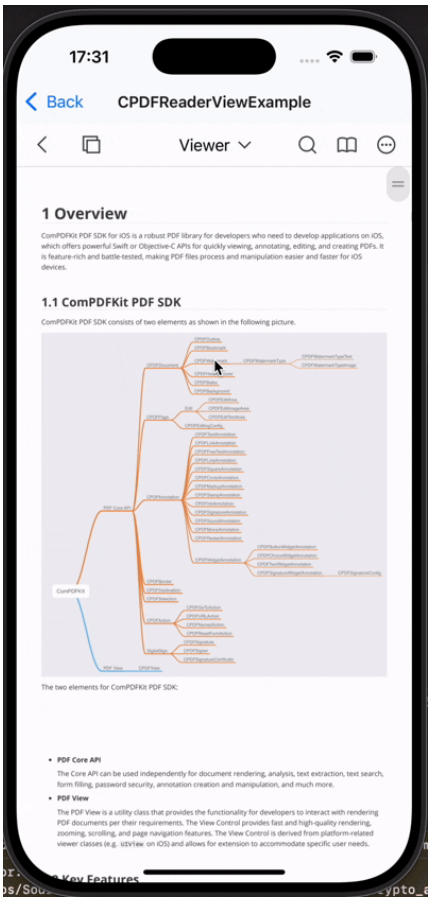
The ComPDFKit React Native SDK provides several ways to control the visibility of user interface (UI) elements. The following table summarizes the supported options:

Option	Description
automatic	The toolbar and other UI elements automatically show or hide when the user taps the page.
always	The user interface is always visible.
never	The user interface remains hidden at all times.

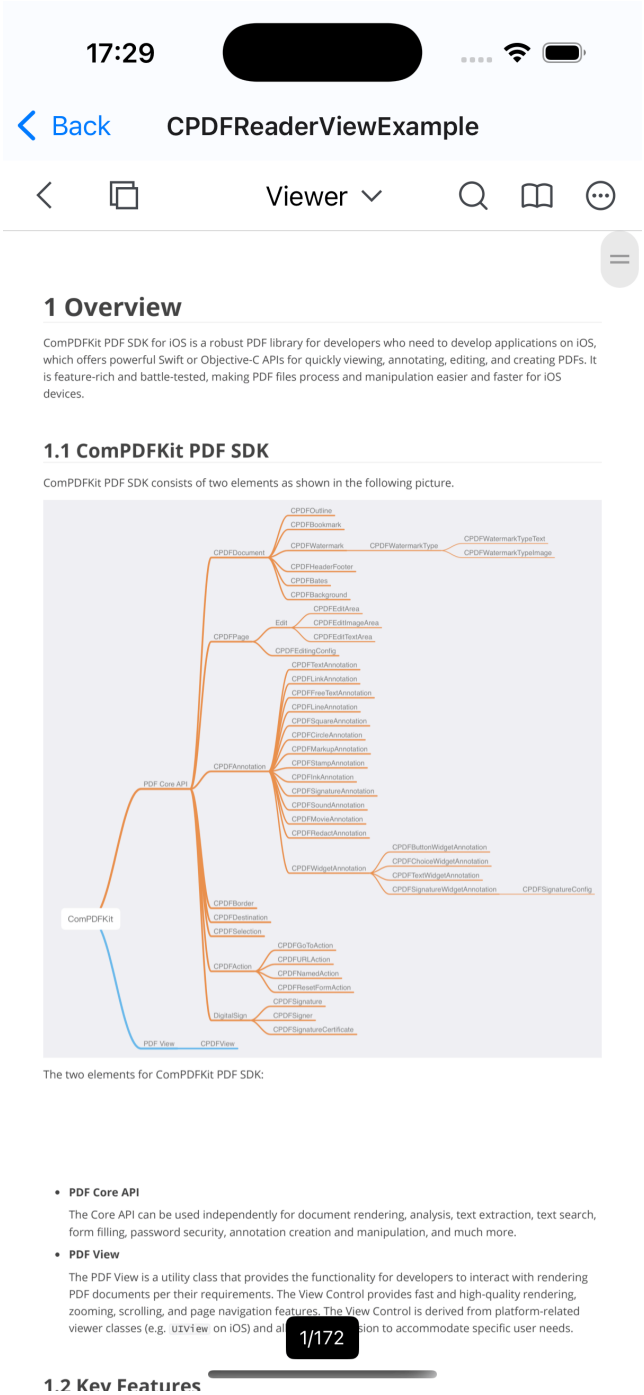
You can use the `uiVisibilityMode` configuration option to change the UI visibility mode. The example below shows how to use the **automatic** mode:

```
ComPDFKit.getDefaultConfig({
  modeConfig: {
    uiVisibilityMode: 'automatic'
  }
});
```

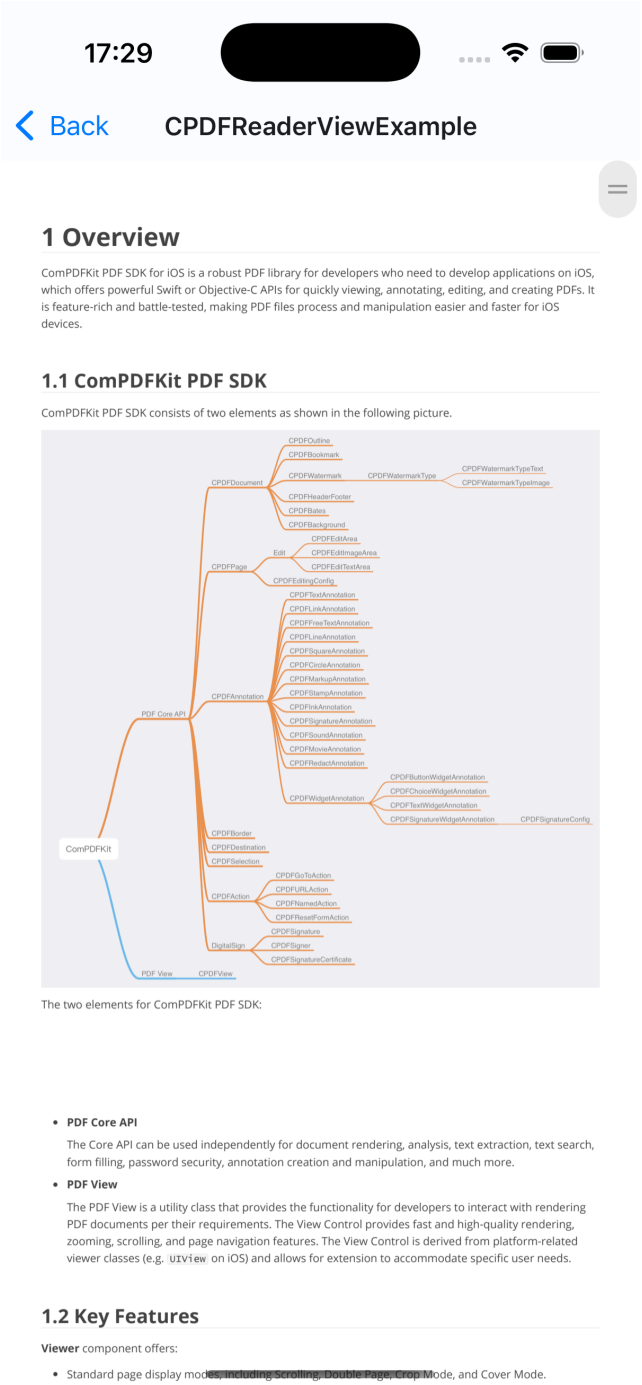
Behavior of automatic mode:

Android	iOS
	

Behavior of always mode:

Android	iOS
	

Behavior of never mode:

Android	iOS
	

3.8.8 BOTA Configuration

Feature Description

The BOTA interface is used to display the **Bookmarks**, **Outline**, and **Annotations** lists of a PDF document.

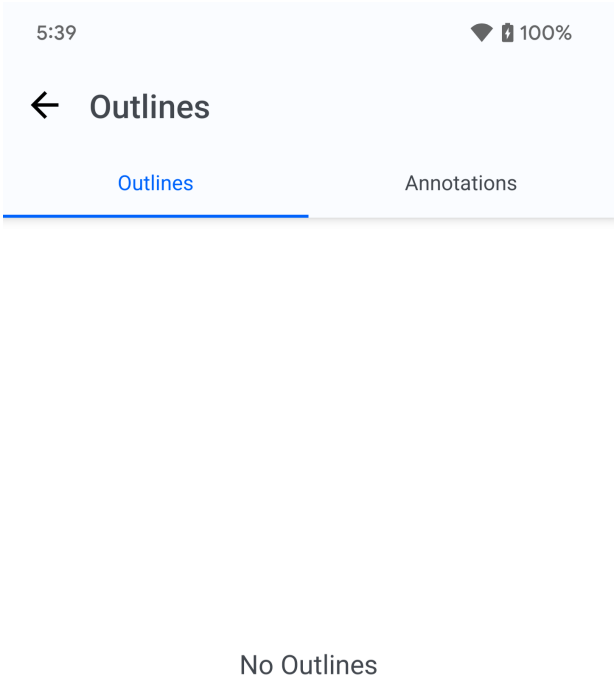
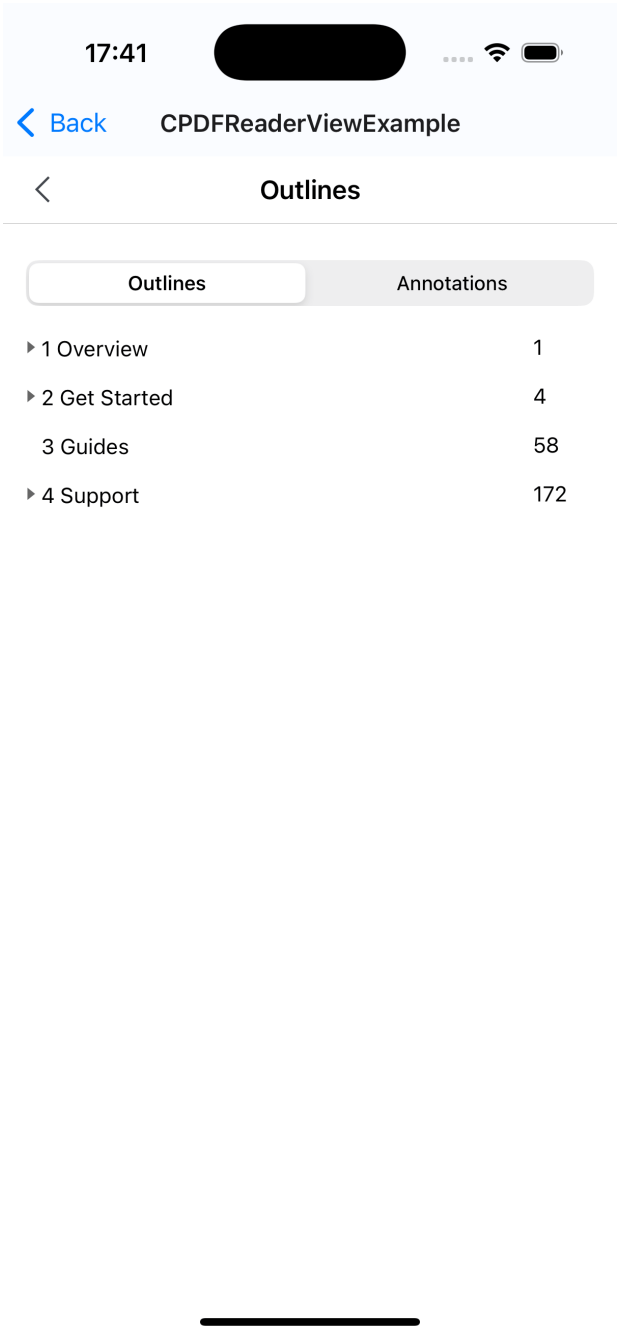
Through BOTA configuration, you can control which types of tabs are displayed and what menu options are available within each interface.

Configure Enabled Tabs

Use `CPDFBotaConfig.tabs` to specify which tabs should be enabled:

```
ComPDFKit.getDefaultConfig({
  global: {
    bota: {
      tabs: ['outline', 'annotations']
    }
  }
});
```

Example Result:

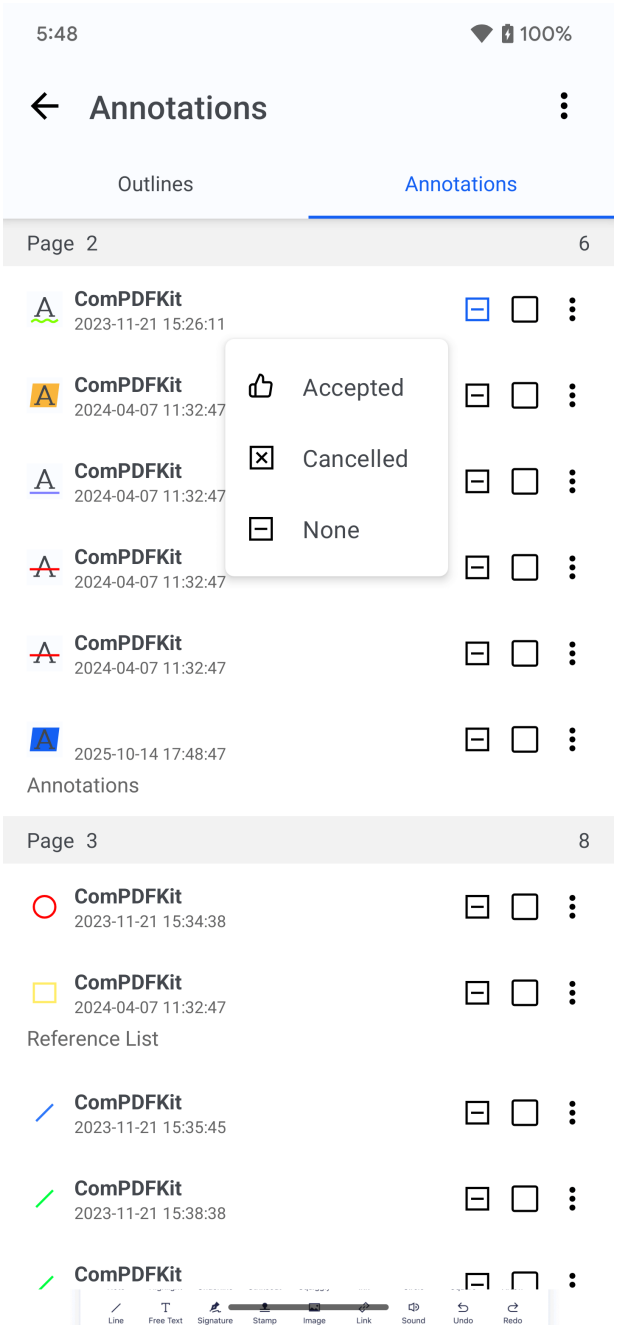
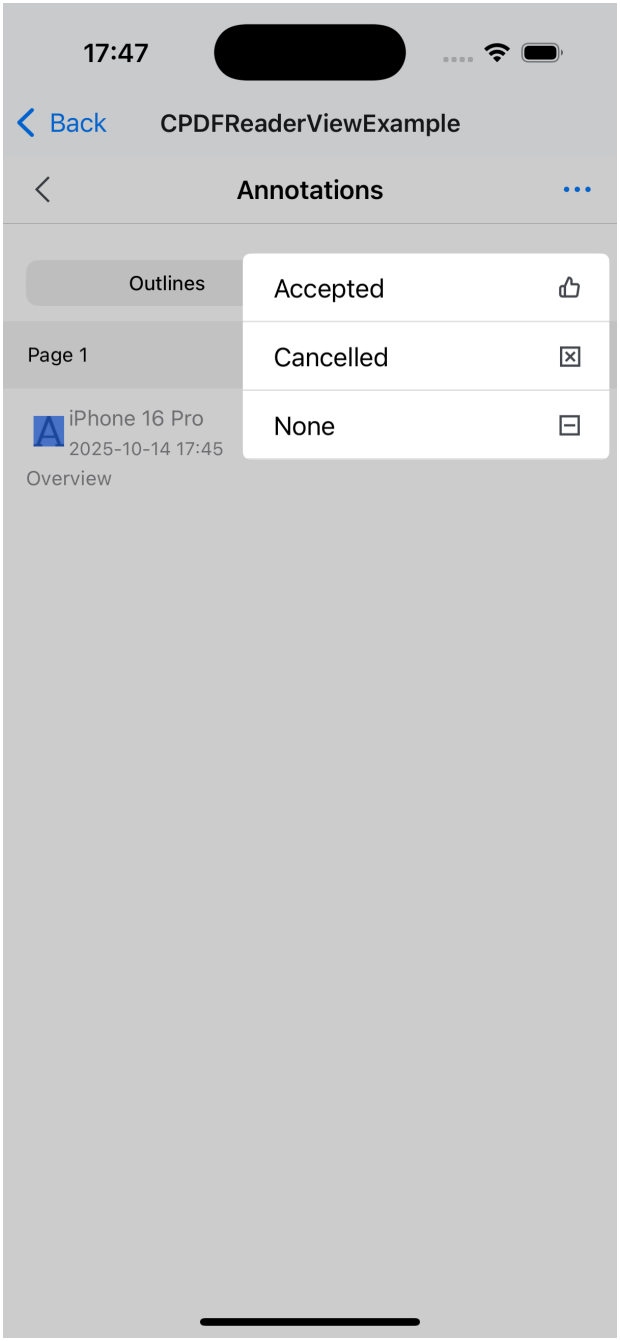
Android	iOS
	

Configure Annotation Menu Options

BOTA supports setting **global menus** and **individual annotation item menus** for the Annotations list interface:

```
ComPDFKit.getDefaultConfig({
  global: {
    bota: {
      tabs: ['outline', 'annotations'],
      menus: {
        annotations: {
          global: botaMenus('importAnnotation', 'exportAnnotation',
'removeAllAnnotation'),
          item: [
            { id: 'reviewStatus', subMenus: ['accepted', 'cancelled', 'none'] },
            { id: 'markedStatus'},
            { id: 'more', subMenus: ['delete']}
          ]
        }
      }
    }
  }
})
```

Example Result:

Android	iOS
	

Annotation Global Menu Options

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

Annotation Review Status Submenu Options

Option	Description
accepted	Accepted
rejected	Rejected
cancelled	Cancelled
completed	Completed
none	No status

More Menu Options

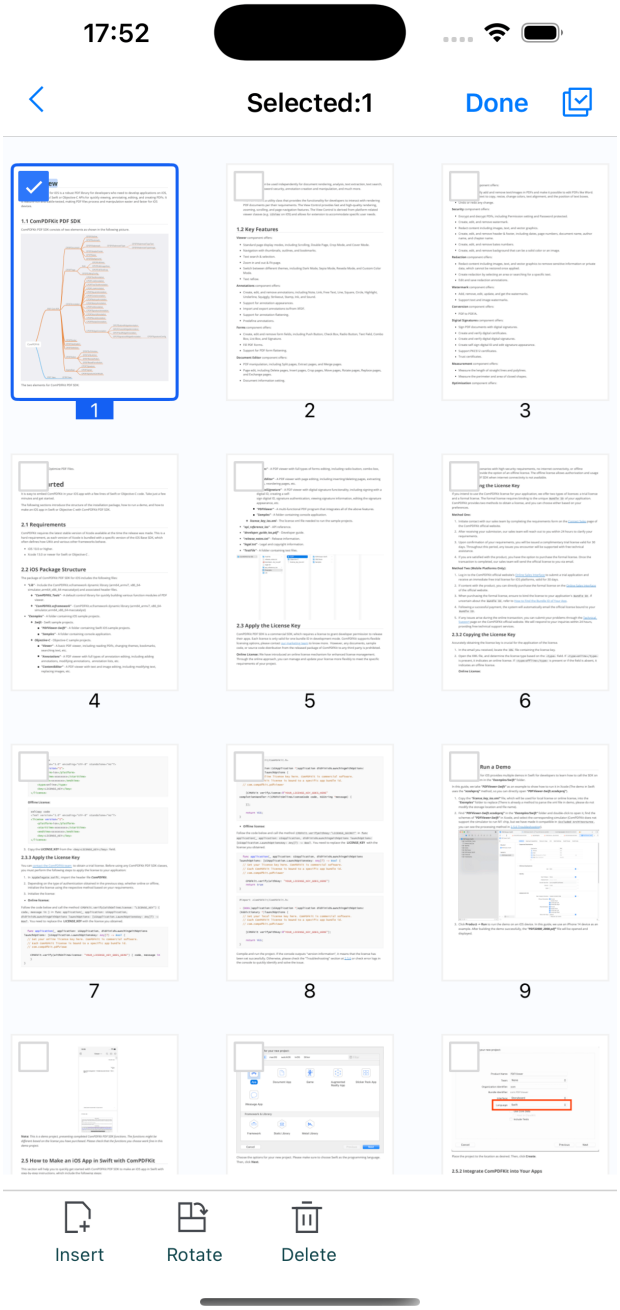
Option	Description
addReply	Add annotation reply
viewReply	View annotation replies
delete	Delete annotation

3.8.9 Page Editor Menu

The ComPDFKit React Native SDK allows configuring the available menu options in the **Page Editor** interface via `CPDFConfiguration`.

```
ComPDFKit.getDefaultConfig({
  global: {
    pageEditor: {
      menus: [
        'insertPage',
        'rotatePage',
        'deletePage'
      ]
    }
  }
})
```

Example Result:

Android	iOS
	

3.9 Content Editing

3.9.1 Overview

Content editing provides text, image, and path editing capabilities, allowing users to easily insert, adjust, or delete text, images, and paths — making the PDF editor function like a rich text editor that supports direct document creation and modification.

Guides

Content Editor Mode

Learn how to switch to content editing mode.

Setting Edit Type

Switch between different editable element types via API or toolbar.

Undo and Redo

Undo or redo modifications using toolbar actions or API calls.

3.9.2 Content Editor Mode

Before performing content editing in `CPDFReaderView`, you need to switch to **CPDFViewMode.contentEditor** mode.

You can switch modes in the following ways:

1. **UI Switching**: Select the mode from the mode switch menu in the upper-right corner of `CPDFReaderView`.
2. **Initialization Setting**: Define the initial mode in `CPDFConfiguration`.
3. **Runtime Switching**: Switch modes programmatically using the `CPDFReaderView API`.

```
// Set content editing mode on initialization
const pdfReaderRef = useRef<CPDFReaderView>(null);

<CPDFReaderView
  ref={pdfReaderRef}
  document={samplePDF}
  configuration={ComPDFKit.getDefaultConfig({
    modeConfig: {
      initialViewMode: 'contentEditor'
    }
  })}
/>

// Switch to content editing mode programmatically
await pdfReaderRef.current?.setPreviewMode(CPDFViewMode.CONTENT_EDITOR);
```

3.9.3 Setting Edit Type

After entering content editing mode, developers can control which types of elements users can edit:

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

Toolbar Switching

Users can switch between editing types using the **bottom toolbar**.

API Switching

If the bottom toolbar is hidden, you can switch edit types using the API:

```
const manager = pdfReaderRef.current._editManager;

// Enable text editing only
await manager.changeEditType([CPDFEditType.Text]);

// Enable both text and image editing
await manager.changeEditType([CPDFEditType.Text, CPDFEditType.Image]);

// Disable all editing (read-only)
await manager.changeEditType([CPDFEditType.NONE]);
```

3.9.4 Undo and Redo

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

Toolbar Actions

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

API Calls

Developers can use the `historyManager` to implement custom undo/redo logic.

Set Callbacks

```
const manager = pdfReaderRef.current._editManager;
const historyManager = manager.historyManager;

// Set a callback to monitor history state changes
historyManager.setOnHistoryStateChangedListener((pageIndex, canUndo, canRedo) => {
});

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

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

3.10 API

`ComPDFKit` contains static methods for global library initialization, configuration, and utility methods.

3.10.1 Initialize SDK

Method Name: `initWithPath(licensePath: string)`

Use your ComPDFKit commercial license key XML file to initialize the ComPDFKit SDK. Please contact our sales team to obtain a trial license.

Parameters:

Name	Type	Description
licensePath	string	Path to the license XML file

Returns a Promise.

Name	Type	Description
result	boolean	Returns true if initialization is successful, otherwise false.

```
// Android: Place in the assets directory
await ComPDFKit.initWithPath('assets://license_key_android.xml')

// iOS: Copy to the iOS project root directory
await ComPDFKit.initWithPath('license_key_ios.xml')

// Alternatively, use an absolute file path
await ComPDFKit.initWithPath('/data/data/your.package.name/files/license_key.xml')
```

3.10.2 Offline initialization

Method: `init_(license : string)`

Initialize the ComPDFKit SDK offline using your ComPDFKit commercial license key. Please contact our sales to obtain a trial license.

Parameters:

Name	Type	Description
license	String	Your ComPDFKit license key

Returns a Promise.

Name	Type	Description
result	boolean	Returns `true` if initialization is successful, otherwise returns `false`.

```
ComPDFKit.init_('your compdfkit license')
```

3.10.3 Online initialization

Method: `initialize(androidOnlineLicense : string, iosOnlineLicense : string)`

Use your ComPDFKit commercial license key to initialize the ComPDFKit SDK using online authentication. Please contact our sales to obtain a trial license.

Parameters:

Name	Type	Description
androidOnlineLicense	string	Your ComPDFKit for React Native Android online license key.
iosOnlineLicense	string	Your ComPDFKit for React Native iOS online license key.

Returns a Promise.

Name	Type	Description
result	boolean	Returns `true` if initialization is successful, otherwise returns `false`.

```
ComPDFKit.initialize('android online license', 'ios online license')
```

3.10.4 Open Document

Method: `openDocument(document : string, password : string, configuration : string)`

Used to present a PDF document.

Parameters:

Name	Type	Description
document	string	The path to the PDF document to be presented.
password	string	PDF document password.
configuration	string	Configuration objects to customize the appearance and behavior of ComPDFKit.

- (Android) For local storage file path:

```
document = '/storage/emulated/0/Download/PDF_document.pdf'
ComPDFKit.openDocument(document, '', ComPDFKit.getDefaultConfig({}))
```

- (Android) For content Uri:

```
document = 'content://...'
ComPDFKit.openDocument(document, '', ComPDFKit.getDefaultConfig({}))
```

- (Android) For assets path:

```
document = "file:///android_asset/..."
ComPDFKit.openDocument(document, '', ComPDFKit.getDefaultConfig({}))
```

- (iOS) For app bundle file path:

```
document = 'pdf_document.pdf'
ComPDFKit.openDocument(document, '', ComPDFKit.getDefaultConfig({}))
```

3.10.5 Get Default Config

Method: `getDefaultConfig()`

When you use the `ComPDFKit.openDocument` method to present a PDF file, you need to pass configuration parameters to customize the UI features and PDF view properties. `ComPDFKit` provides default configuration parameters through `ComPDFKit.getDefaultConfig`. You can retrieve them using the following example:

```
ComPDFKit.getDefaultConfig({})
```

You can modify certain parameters to meet your requirements. Here are some usage examples:

1. Setting the initial display mode and available mode list. The following code is an example that enables only the viewer mode and annotation mode:

```
ComPDFKit.getDefaultConfig({
  modeConfig: {
    initialViewMode: CPDFViewMode.VIEWER,
    availableViewModes: [
      CPDFViewMode.VIEWER,
      CPDFViewMode.ANNOTATIONS
    ]
  }
})
```

2. Setting the enabled annotation types and the default annotation attribute values list. For example, enabling only note annotations and setting the color and transparency of note annotations:

```
ComPDFKit.getDefaultConfig({
  annotationsConfig:{
    availableType:[
      CPDFAnnotationType.NOTE
    ],
    availableTools:[
      CPDFConfigTool.SETTING,
      CPDFConfigTool.UNDO,
      CPDFConfigTool.REDO
    ],
```

```
    initAttribute:{
      note:{
        color: '#1460F3',
        alpha: 255
      }
    }
  }
})
```

3.Setting the display mode and page turning direction:

```
ComPDFKit.getDefaultConfig({
  readerViewConfig:{
    displayMode: CPDFDisplayMode.DOUBLE_PAGE,
    verticalMode: false
  }
})
```

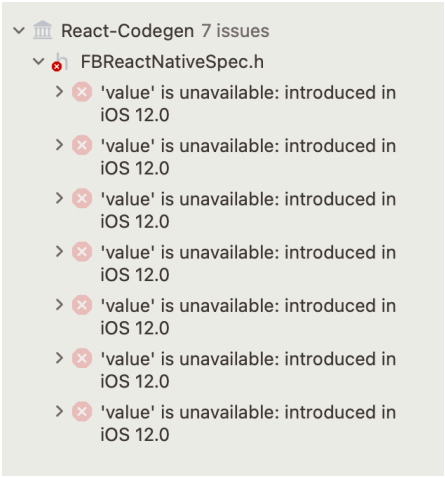
For more configuration parameter descriptions, please see CPDFCONFIGURATION.md.

4 Troubleshooting

Differences in React Native development environments and project versions may lead to error prompts during integration. This chapter compiles some potential issues and provides solutions for your reference. If you encounter a problem not listed in this chapter, please feel free to contact our technical team, and we will promptly provide technical support.

Issue 1: 'value' is unavailable: Introduced in iOS 12.0

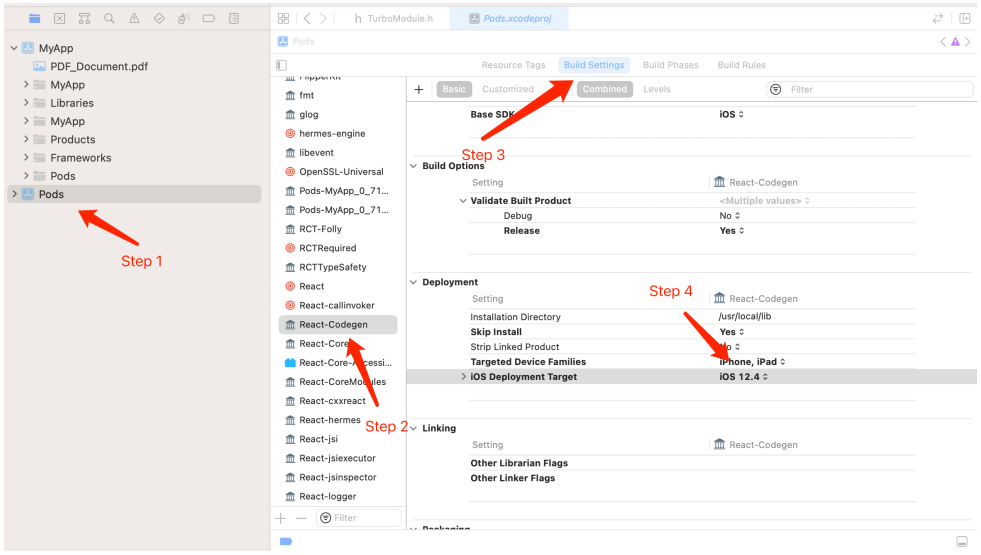
This issue may arise when upgrading the React Native version to 0.71.0 and compiling an iOS platform project, as shown in the screenshot below:



Solution:

1. Open your iOS project using Xcode.
2. n the Xcode project navigator, select Pods.

3. Under Targets, choose React-Codegen.
4. Set the window to Build Settings.
5. Under Deployment, set iOS Deployment Target to 12.4.
6. Clean the project and rebuild: Product > Clean Build Folder, Product > Build.



5 Support

5.1 Reporting Problems

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