pub.dev

pdf_render 1.3.6

Published 24 days ago • espresso3389.jp Null safety

SDK FLUTTER PLATFORM ANDROID IOS MACOS WEB

121

Metadata →

A plugin that provides you with intermediate PDF rendering APIs and easy-to-use Flutter Widgets.

More...

Readme Changelog Example Installing Versions Scores

Introduction

pdf_render is a PDF renderer implementation that supports iOS (>= 8.0), Android (>= API Level 21), and Web. It provides you with intermediate PDF rendering APIs and also easy-to-use Flutter Widgets.

Easiest sample

The following fragment illustrates the easiest way to show a PDF file in assets:

```
@override
Widget build(BuildContext context) {
  return new MaterialApp(
    home: new Scaffold(
      appBar: new AppBar(
        title: const Text('Easiest PDF sample'),
      ),
      backgroundColor: Colors.grey,
      body: PdfViewer.openAsset('assets/hello.pdf')
    )
  );
}
```


localhost:60275/#/
localhost:60275/#/
Page #1
DEBUG
Hello, Flutter!
こんにちは、Flutter!

Install

Add this to your package's `pubspec.yaml` file and execute `flutter pub get`:

```
dependencies:
  pdf_render: ^1.3.6
```

Web

For Web, you should add `<script>` tags on your `index.html`:

The plugin utilizes PDF.js to support Flutter Web.

To use the Flutter Web support, you should add the following code just before `<script src="main.dart.js" type="application/javascript"></script>` inside `index.html`:

```
<!-- IMPORTANT: load pdfjs files -->
<script
  src="https://cdn.jsdelivr.net/npm/pdfjs-dist@2.12.313/build/pdf.js"
  type="text/javascript"
></script>
<script type="text/javascript">
  pdfjsLib.GlobalWorkerOptions.workerSrc =
    "https://cdn.jsdelivr.net/npm/pdfjs-dist@2.12.313/build/pdf.worker.min.js";
  pdfRenderOptions = {
    // where cmaps are downloaded from
    cMapUrl: "https://cdn.jsdelivr.net/npm/pdfjs-dist@2.12.313/cmaps/",
    // The cmaps are compressed in the case
    cMapPacked: true,
    // any other options for pdfjsLib.getDocument.
    // params: {}
  };
</script>
```

You can use any URL that specify `PDF.js` distribution URL. `cMapUrl` indicates cmap files base URL and `cMapPacked` determines whether the cmap files are compressed or not.

iOS/Android

For iOS and Android, no additional task needed.

macOS

For macOS, there are two notable issues:

- Asset access is not working yet; see Flutter issue #47681: [macOS] add lookupKeyForAsset to FlutterPluginRegistrar
- Flutter app restrict its capability by enabling App Sandbox by default. You can change the behavior by editing your app's entitlements files depending on your configuration. See the discussion below.
  - `macos/Runner/Release.entitlements`
  - `macos/Runner/DebugProfile.entitlements`

Deal with App Sandbox

The easiest option to access files on your disk, set `com.apple.security.app-sandbox` to `false` on your entitlements file though it is not recommended for releasing apps because it completely disables App Sandbox.

Another option is to use `com.apple.security.files.user-selected.read-only` along with file_selector_macos. The option is better in security than the previous option.

Anyway, the example code for the plugin illustrates how to download and preview internet hosted PDF file. It uses `com.apple.security.network.client` along with flutter_cache_manager:

```
<dict>
  <key>com.apple.security.app-sandbox</key>
  <true/>
  <key>com.apple.security.network.client</key>
  <true/>
</dict>
```

For the actual implementation, see Missing network support? and the example code.

Widgets

Import Widgets Library

Firstly, you must add the following import:

```
import 'package:pdf_render/pdf_render_widgets.dart';
```

PdfViewer

PdfViewer is an extensible PDF document viewer widget which supports pinch-zoom.

The following fragment is a simplest use of the widget:

```
@override
Widget build(BuildContext context) {
  return new MaterialApp(
    home: new Scaffold(
      appBar: new AppBar(
        title: const Text('Pdf_render example app'),
      ),
      backgroundColor: Colors.grey,
      // You can use either PdfViewer.openFile, PdfViewer.openAsset, or PdfVie
      body: PdfViewer.openAsset(
        'assets/hello.pdf',
        params: PdfViewerParams(pageNumber: 2), // show the page-2
      )
    )
  );
}
```

In the code above, the code uses PdfViewer.openAsset to load a asset PDF file. There are also PdfViewer.openFile for local file and PdfViewer.openData for `Uint8List` of PDF binary data.

Missing network support?

A frequent feature request is something like `PdfViewer.openUri`. The plugin does not have it but it's easy to implement it with flutter_cache_manager:

```
FutureBuilder<File>(
  future: DefaultCacheManager().getSingleFile(
    'https://github.com/espresso3389/flutter_pdf_render/raw/master/example/asset
  builder: (context, snapshot) => snapshot.hasData
    ? PdfViewer.openFile(snapshot.data!.path)
    : Container( /* placeholder */),
)
```

PdfViewerParams

PdfViewerParams contains parameters to customize PdfViewer.

It also equips the parameters that are inherited from InteractiveViewer. You can use almost all parameters of InteractiveViewer.

PdfViewerController

PdfViewerController can be used to obtain number of pages in the PDF document.

goTo/goToPage/goToPointInPage

It also provide goTo and goToPage methods that you can scroll the viewer to make certain page/area of the document visible:

```
@override
Widget build(BuildContext context) {
  PdfViewerController? controller;
  return new MaterialApp(
    home: new Scaffold(
      appBar: new AppBar(
        title: const Text('Pdf_render example app'),
      ),
      backgroundColor: Colors.grey,
      body: PdfViewer.openAsset(
        'assets/hello.pdf',
        params: PdfViewerParams(
          // called when the controller is fully initialized
          onViewerControllerInitialized: (PdfViewerController c) {
            controller = c;
            controller.goToPage(pageNumber: 3); // scrolling animation to page
          }
        )
      ),
    ),
      floatingActionButton: Column(
        mainAxisAlignment: MainAxisAlignment.end,
        children: <Widget>[
          FloatingActionButton(
            child: Icon(Icons.first_page),
            onPressed: () => controller.ready?.goToPage(pageNumber: 1),
          ),
          FloatingActionButton(
            child: Icon(Icons.last_page),
            onPressed: () => controller.ready?.goToPage(pageNumber: controller
          ),
        ],
      ),
    ),
  );
}
```

goToPointInPage is just another version of goToPage, which also accepts inner-page point and where the point is anchored to.

The following fragment shows page 1's center on the widget's center with the zoom ratio 300%:

```
controller.goToPointInPage(
  pageNumber: 1,
  x: 0.5,
  y: 0.5,
  anchor: PdfViewerAnchor.center,
  zoomRatio: 3.0,
);
```

And, if you set `x: 0, y: 0, anchor: PdfViewerAnchor.topLeft`, the behavior is identical to goToPage.

setZoomRatio

setZoomRatio is a method to change zoom ratio without scrolling the view (***it's not exactly the true but almost).

The following fragment changes zoom ratio to 2.0:

```
controller.setZoomRatio(2.0);
```

During the zoom changing operation, it keeps the center point in the widget being centered.

The following fragment illustrates another use case, zoom-on-double-tap:

```
final controller = PdfViewerController();
TapDownDetails? doubleTapDetails;

...

GestureDetector(
  // Supporting double-tap gesture on the viewer.
  onDoubleTapDown: (details) => doubleTapDetails = details,
  onDoubleTap: () => controller.ready?.setZoomRatio(
    zoomRatio: controller.zoomRatio * 1.5,
    center: doubleTapDetails!.localPosition,
  ),
  child: PdfViewer.openAsset(
    'assets/hello.pdf',
    viewerController: controller,
    ...
```

Using GestureDetector, it firstly captures the double-tap location on onDoubleTapDown. And then, onDoubleTap uses the location as the zoom center.

Managing gestures

PdfViewer does not support any gestures except panning and pinch-zooming. To support other gestures, you can wrap the widget with GestureDetector as explained above.

Page decoration

Each page shown in PdfViewer is by default has drop-shadow using BoxDecoration. You can override the appearance by PdfViewerParams.pageDecoration property.

Further page appearance customization

PdfViewerParams.buildPagePlaceholder is used to customize the white blank page that is shown before loading the page contents.

PdfViewerParams.buildPageOverlay is used to overlay something on every page.

Both functions are defined as BuildPageContentFunc:

```
typedef BuildPageContentFunc = Widget Function(
  BuildContext context,
  int pageNumber,
  Rect pageRect);
```

The third parameter, `pageRect` is location of page in viewer's world coordinates.

Single page view

The following fragment illustrates the easiest way to render only one page of a PDF document using PdfDocumentLoader widget. It is suitable for showing PDF thumbnail.

```
@override
Widget build(BuildContext context) {
  return new MaterialApp(
    home: new Scaffold(
      appBar: new AppBar(
        title: const Text('Pdf_render example app'),
      ),
      backgroundColor: Colors.grey,
      body: Center(
        child: PdfDocumentLoader.openAsset(
          'assets/hello.pdf',
          pageNumber: 1,
          pageBuilder: (context, textureBuilder, pageSize) => textureBuilder()
        )
      )
    ),
  );
}
```

Of course, PdfDocumentLoader has the following factory functions:

- PdfDocumentLoader.openAsset
- PdfDocumentLoader.openFile
- PdfDocumentLoader.openData

Multi-page view using ListView.builder

Using PdfDocumentLoader in combination with PdfPageView, you can show multiple pages of a PDF document. In the following fragment, `ListView.builder` is utilized to realize scrollable PDF document viewer.

The most important role of PdfDocumentLoader is to manage life time of PdfDocument and it disposes the document when the widget tree is going to be disposed.

```
@override
Widget build(BuildContext context) {
  return new MaterialApp(
    home: new Scaffold(
      appBar: new AppBar(
        title: const Text('Pdf_render example app'),
      ),
      backgroundColor: Colors.grey,
      body: Center(
        child: PdfDocumentLoader.openAsset(
          'assets/hello.pdf',
          documentBuilder: (context, pdfDocument, pageCount) => LayoutBuilder(
            builder: (context, constraints) => ListView.builder(
              itemCount: pageCount,
              itemBuilder: (context, index) => Container(
                margin: EdgeInsets.all(margin),
                padding: EdgeInsets.all(padding),
                color: Colors.black12,
                child: PdfPageView(
                  pdfDocument: pdfDocument,
                  pageNumber: index + 1,
                )
              )
            )
          ),
        )
      )
    ),
  );
}
```

Customizing page widget

Both PdfDocumentLoader and PdfPageView accepts `pageBuilder` parameter if you want to customize the visual of each page.

The following fragment illustrates that:

```
PdfPageView(
  pageNumber: index + 1,
  // pageSize is the PDF page size in pt.
  pageBuilder: (context, textureBuilder, pageSize) {
    //
    // This illustrates how to decorate the page image with other widgets
    //
    return Stack(
      alignment: Alignment.bottomCenter,
      children: <Widget>[
        // the container adds shadow on each page
        Container(
            margin: EdgeInsets.all(margin),
            padding: EdgeInsets.all(padding),
            decoration: BoxDecoration(boxShadow: [
              BoxShadow(
                  color: Colors.black45,
                  blurRadius: 4,
                  offset: Offset(2, 2))
            ]),
            // textureBuilder builds the actual page image
            child: textureBuilder()),
        // adding page number on the bottom of rendered page
        Text('${index + 1}', style: TextStyle(fontSize: 50))
      ],
    );
  },
)
```

textureBuilder

`textureBuilder` (PdfPageTextureBuilder) generates the actual widget that directly corresponding to the page image. The actual widget generated may vary upon the situation. But you can of course customize the behavior of the function with its parameter.

The function is defined as:

```
typedef PdfPageTextureBuilder = Widget Function({
  Size? size,
  PdfPagePlaceholderBuilder? placeholderBuilder,
  bool backgroundFill,
  double? renderingPixelRatio
});
```

So if you want to generate widget of an exact size, you can specify `size` explicitly.

Please note that the size is in density-independent pixels. The function is responsible for determining the actual pixel size based on device's pixel density.

`placeholderBuilder` is the final resort that controls the "placeholder" for loading or failure cases.

```
/// Creates page placeholder that is shown on page loading or even page load fai
typedef PdfPagePlaceholderBuilder = Widget Function(Size size, PdfPageStatus sta

/// Page loading status.
enum PdfPageStatus {
  /// The page is currently being loaded.
  loading,
  /// The page load failed.
  loadFailed,
}
```

PDF rendering APIs

The following fragment illustrates overall usage of PdfDocument:

```
import 'package:pdf_render/pdf_render.dart';

...

// Open the document using either openFile, openAsset, or openData.
// For Web, file name can be relative path from index.html or any arbitrary URL
// but affected by CORS.
PdfDocument doc = await PdfDocument.openAsset('assets/hello.pdf');

// Get the number of pages in the PDF file
int pageCount = doc!.pageCount;

// The first page is 1
PdfPage page = await doc!.getPage(1);


// For the render function's return, see explanation below
PdfPageImage pageImage = await page.render();

// Now, you can access pageImage!.pixels for raw RGBA data
// ...

// Generating dart:ui.Image cache for later use by imageIfAvailable
await pageImage.createImageIfNotAvailable();

// PDFDocument must be disposed as soon as possible.
doc!.dispose();
```

And then, you can use PdfPageImage to get the actual RGBA image in dart:ui.Image.

To embed the image in the widget tree, you can use RawImage:

```
@override
Widget build(BuildContext context) {
  return Center(
    child: Container(
      padding: EdgeInsets.all(10.0),
      color: Colors.grey,
      child: Center(
        // before using imageIfAvailable, you should call createImageIfNotAvaila
        child: RawImage(image: pageImage.imageIfAvailable, fit: BoxFit.contain))
    )
  );
}
```

If you just building widget tree, you had better use faster and efficient PdfPageImageTexture.

PdfDocument.openXXX

On PdfDocument class, there are three functions to open PDF from a real file, an asset file, or a memory data.

```
// from an asset file
PdfDocument docFromAsset = await PdfDocument.openAsset('assets/hello.pdf');

// from a file
// For Web, file name can be relative path from index.html or any arbitrary URL
// but affected by CORS.
PdfDocument docFromFile = await PdfDocument.openFile('/somewhere/in/real/file/sy

// from PDF memory image on Uint8List
PdfDocument docFromData = await PdfDocument.openData(data);
```

PdfDocument members

PdfDocument class overview:

```dart
class PdfDocument {
  /// File path, `asset:[ASSET_PATH]` or `memory:` depending on the content oper
  final String sourceName;
  /// Number of pages in the PDF document.
  final int pageCount;
  /// PDF major version.
  final int verMajor;
  /// PDF minor version.
  final int verMinor;
  /// Determine whether the PDF file is encrypted or not.
  final bool isEncrypted;
  /// Determine whether the PDF file allows copying of the contents.
  final bool allowsCopying;
  /// Determine whether the PDF file allows printing of the pages.
  final bool allowsPrinting;

  // Get a page by page number (page number starts at 1)
  Future<PdfPage> getPage(int pageNumber);

  // Dispose the instance.
  Future<void> dispose();
}
```

PdfPage members

PdfPage class overview:

```dart
class PdfPage {
  final PdfDocument document; // For internal purpose
  final int pageNumber; // Page number (page number starts at 1)
  final double width; // Page width in points; pixel size on 72-dpi
  final double height; // Page height in points; pixel size on 72-dpi

  // render sub-region of the PDF page.
  Future<PdfPageImage> render({
    int x = 0,
    int y = 0,
    int? width,
    int? height,
    double? fullWidth,
    double? fullHeight,
    bool backgroundFill = true,
    bool allowAntialiasingIOS = false
  });
}
```

render function extracts a sub-region `(x,y) - (x + width, y + height)` from scaled `fullWidth x fullHeight` PDF page image. All the coordinates are in pixels.

The following fragment renders the page at 300 dpi:

```
const scale = 300.0 / 72.0;
const fullWidth = page.width * scale;
const fullHeight = page.height * scale;
var rendered = page.render(
  x: 0,
  y: 0,
  width: fullWidth.toInt(),
  height: fullHeight.toInt(),
  fullWidth: fullWidth,
  fullHeight: fullHeight);
```

PdfPageImage members

PdfPageImage class overview:

```
class PdfPageImage {
  /// Page number. The first page is 1.
  final int pageNumber;
  /// Left X coordinate of the rendered area in pixels.
  final int x;
  /// Top Y coordinate of the rendered area in pixels.
  final int y;
  /// Width of the rendered area in pixels.
  final int width;
  /// Height of the rendered area in pixels.
  final int height;
  /// Full width of the rendered page image in pixels.
  final int fullWidth;
  /// Full height of the rendered page image in pixels.
  final int fullHeight;
  /// PDF page width in points (width in pixels at 72 dpi).
  final double pageWidth;
  /// PDF page height in points (height in pixels at 72 dpi).
  final double pageHeight;
  /// RGBA pixels in byte array.
  final Uint8List pixels;

  /// Get [dart:ui.Image] for the object.
  Future<Image> createImageIfNotAvailable() async;

  /// Get [Image] for the object if available; otherwise null.
  /// If you want to ensure that the [Image] is available,
  /// call [createImageIfNotAvailable].
  Image? get imageIfAvailable;
}
```

createImageIfNotAvailable generates image cache in dart:ui.Image and imageIfAvailable returns the cached image if available.

If you just need RGBA byte array, you can use pixels for that purpose. The pixel at `(x,y)` is on `pixels[(x+y*width)*4]`. Anyway, it's highly discouraged to modify the contents directly though it would work correctly.

PdfPageImageTexture members

PdfPageImageTexture is to utilize Flutter's Texture class to realize faster and resource-saving rendering comparing to PdfPageImage/RawImage combination.

```
class PdfPageImageTexture {
  final PdfDocument pdfDocument;
  final int pageNumber;
  final int texId;

  bool get hasUpdatedTexture;

  PdfPageImageTexture({required this.pdfDocument, required this.pageNumber, requ

  /// Create a new Flutter [Texture]. The object should be released by calling [
  static Future<PdfPageImageTexture> create({required PdfDocument pdfDocument, r

  /// Release the object.
  Future<void> dispose();

  /// Extract sub-rectangle ([x],[y],[width],[height]) of the PDF page scaled to
  /// If [backgroundFill] is true, the sub-rectangle is filled with white before
  Future<bool> extractSubrect({
    int x = 0,
    int y = 0,
    required int width,
    required int height,
    double? fullWidth,
    double? fullHeight,
    bool backgroundFill = true,
  });
}
```

Custom Page Layout

PdfViewerParams has a property layoutPages to customize page layout.

Sometimes, when you're using **Landscape** mode on your Phone or Tablet and you need to show pdf fit to the center of the screen then you can use this code to customize the pdf layout.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    backgroundColor: Colors.white70,
    body: PdfViewer.openAsset(
      'assets/hello.pdf',
      params: PdfViewerParams(
        layoutPages: (viewSize, pages) {
          List<Rect> rect = [];
          final viewWidth = viewSize.width;
          final viewHeight = viewSize.height;
          final maxHeight = pages.fold<double>(0.0, (maxHeight, page) => max(m
          final ratio = viewHeight / maxHeight;
          var top = 0.0;
          for (var page in pages) {
            final width = page.width * ratio;
            final height = page.height * ratio;
            final left = viewWidth > viewHeight ? (viewWidth / 2) - (width / 2
            rect.add(Rect.fromLTWH(left, top, width, height));
            top += height + 8 /* padding */;
          }
          return rect;
        },
      ),
    ),
  );
}
```

Preview