BRANCHFIRE™

iANNOTATE PDF SDK

GETTING STARTED GUIDE

WWW.BRANCHFIRE.COM

Table of Contents

Welcome to the iAnnotate PDF SDK!

The SDK provides powerful PDF rendering and annotation technology for your iPad app. We provide Cocoa-style APIs, so you can learn to use our tools fast and start concentrating on your project. The SDK also includes search and navigation functionality, a built-in UI you can use, and plenty of features appropriate for security-conscious environments.

The best way to learn our SDK is to download an evaluation, open up a sample project, and dig in. All of our code is documented, and the samples demonstrate the basic method behind displaying, processing, and annotating PDFs. Beyond that, this guide provides general information about the usage and features of the SDK.

This guide is divided into six sections. The copy you're reviewing is current through v2.4. This guide is not intended to be an exhaustive instructional manual, but rather a way to speed up learning the basics of our SDK and answer a few common questions. If at any time you need additional technical support, contact us at library-support@branchfire.com.

Evaluation vs. Full Versions

The PDF SDK comes in two different versions: Evaluation for testing out the SDK prior to purchasing, and Full for current licensees. Our Evaluation version has the same features as the Full version, except annotations cannot be written or synced with a PDF, and documents will appear with a watermark.

Both versions include the following:

- license.xml file for your project's resources folder
- license key to initialize the library in the app delegate
- library binary, classes, and resources
- complete sample application

Incorporate the first three bullet points above into your existing project, or use the sample app as the basis for a new one.

Release vs. OpenSSL-Free

When you download the Full version of our SDK you'll see two options: Release and OpenSSL-Free. For the majority of our licensees the Release version of our application is the best choice, but if you have OpenSSL installed in your project (often from other libraries) you might need the OpenSSL-Free version to avoid conflicts. Please contact our SDK Support Team at library-support@branchfire.com if you have further questions.

Incorporating the SDK into a Project

You're more than welcome to start from one of our sample applications, but if you need to integrate our SDK with a current project here are the steps. This process will work for both Evaluation and full versions.

1. In the Other Linker Flags section of the build settings the following must be included: `-lz -lsqlite3 -lstdc++ -ObjC -all_load`
2. All of the resources from the Resources folder of the library distribution must be added to the project, including image files, string resources, and font data files. When adding the contents of the resources folder, make sure the Create Folder References for Any Added Folders option is selected, so that the FontCMaps folder is added to your project as a folder, and not a group in the project tree (i.e., should show as a blue icon).
3. Add the header files (from the Include directory) to the project.

4. Add the static library (lib/libAjiPDFLib.a) to the project.
5. Add your license file, which must be called "license.xml", to the project. Generally this should be included in the Resources folder for your application.
6. Import AjiPDFLib.h in any file that requires use of the API.
7. Change the bundle ID of your project to the bundle ID in your license file (for the full version only).

Switching the Evaluation Version Out for the Full

If you've already built quite a bit of your application with our Evaluation library, you may want to just swap it with the full library once you sign the license agreement and receive your file and key. Follow these four steps to switch them out.

8. Remove the current license.xml file from your project, and import the new one we provided you. Make sure to place it in the resources folder.
9. Replace the Evaluation library's license key with the new one we provided you. Do this where you initialize the APLibrary object (typically in your App Delegate).
10. Remove the binary provided with the Evaluation version and import the library binary for the full version.
11. Change your application's bundle ID (in the info.plist file). Make sure the new bundle ID is exactly the same as the one you provided to us.

Starting Your Project from Our Sample Apps

Both Evaluation and full libraries include a sample application with basic functionality. We encourage our clients who have not yet started a project to develop directly from the sample app. You will likely want to add a file management system, additional navigation features, more annotations, and your own interface elements / graphics.

Take a Tour of the SDK

Once you've downloaded the SDK you should open a sample app in Xcode and check it out. Here's a suggested "tour itinerary" to help you learn the basics.

1. Check out the license.xml file in the sample code. The Evaluation Version includes a license file that can be integrated with another application alongside our library, as explained in Section 1.
2. Look over `APPDFDocument`. This class encapsulates a PDF file and its information. To display a new PDF you'll start with this class, initializing with either a path or an NSData object.
3. Next review `APPDFInformation`. The object generated by this class is either cached on disk (for file-based PDFs), or kept in memory (for in-memory PDFs). Annotations, indexed text, hyperlinks, and outlines are stored here. `APPDFInformation` has a number of useful properties and methods.
4. `APPDFProcessor` is a driver class that both processes PDF files and writes back their annotations. Make sure to check out `APPDFProcessorDelegate` too.
5. `APPDFViewController` and `APAnnotatingPDFViewController` contain most of the methods you hook interface elements up to. Also investigate the related delegates to find out how you can determine when your users take certain actions.
6. Look through the properties in `APPDFViewOptions` to see various options available for displaying and interacting with PDFs.
7. Finally look at the interface and implementation files in our sample application. Here you'll see how to use our APIs to initialize the library, render a PDF, and provide tools to work with it. When you're done, go ahead and build the sample app to see how it all works live.

Rendering and Annotating PDFs

To display a PDF, first decide if you're using disk-based or in-memory PDFs. We recommend using on-dsik documents unless you need to keep them off-disk for security reasons. Then choose the appropriate method to initialize an APPDFDocument.

```
APPDFDocument initWithPath:

APPDFDocument initWithPath:information:

APPDFDocument initWithData:
```

After you have the APPDFDocument object, use it to initialize a view controller.

```
    APAnnotatingPDFViewController initWithPDF:
```

Add the view controller's view to the host view and you're off to the races! Of course, there's a good bit more to it than all this, so we recommend reading through our sample applications to get more details.

Working with PDFs On-Disk

The PDF SDK allows you to work with documents either on-disk or in-memory. Working on disk is the preferred way to manage PDF documents. With a disk-based document our SDK only loads part of the PDF in memory at a given time, and also caches information about text and annotations. That means you only need to process a PDF once.

Having a PDF on disk is also a good choice if your users work with very large PDFs, given that data-backed PDFs are necessarily limited by system memory. Both of our sample apps demonstrate the process of using documents saved to disk. For projects where security is not the biggest concern, we recommend using on-disk PDFs.

Working with PDFs In-Memory

Many of our licensees choose to keep PDFs in-memory for security reasons. In this case we provide APIs to work with PDFs as `NSData` objects that never touch the disk. There is no need to initialize an information file either, as all indexed text, PDF outlines, and previous annotations are stored in memory. At the end of a session you can sync your user's annotations to the PDF, then encrypt the data using whatever technology you use for that purpose. You can also write the annotated (or flattened) PDF to a new NSData object and work with it from there.

One drawback to this method is that PDF size will be limited by available system memory. Another drawback is that it's not possible to save an information file securely, so if you want to offer search and enable viewing of previous annotations, you'll need to reprocess the PDF every time it's used.

Processing PDFs

If you intend on supporting search and outlines, or enabling hyperlinks and previously made annotations, you must process the document first. Follow these steps to process a PDF:

8. Initialize an `APPDFProcessor` object.
9. Assign its delegate
10. Call `processPDF:` and pass the `APPDFDocument`

Please note we have several methods in `APPDFProcessorDelegate` to help you fine tune how processing works. Processed information is stored in our APPDFInformation object, which can be reused to avoid processing in the future (on-disk PDFs only).

We strongly recommend processing PDFs on a background thread. Trying to process on the main thread is likely to slow down your app. Our sample applications provide examples of this.

Live Processing

Our live processing feature allows your users to mark up a document as soon as it's rendered, without having to wait for processing to finish. It's also possible to forego processing altogether, and just stick with live processing. Users will still be able to select text, and you'll still have access to the text content of your users' markups. You can also sync the annotations back to the document or write the file out as normal.

If you choose to use live processing only selectable text won't be indexed, so you won't have the ability to search unless the PDF is processed. Likewise you will not be able to show previously made annotations or PDF outlines, and hyperlinks will be inactive.

Processing Annotations-Only to Improve Initial Performance

When you initialize an `APPDFProcessor` object, you have two processor options available to you. `kAPPDFProcessingOptionsAnnotationsOnly` does everything our default processor does except for indexing text. This can be useful as a way to improve performance by avoiding a full process until your user attempts to search.

Processing for annotations only is faster than fully processing a document. Therefore, if your user opens a document but never uses search, you can avoid the more expensive text indexing while still providing access to earlier markups, outlines, and active hyperlinks. Once the user hits a Search Button, you can show a spinner or otherwise indicate that text is indexing.

All that said, our SDK processes most documents very quickly. We recommend testing your project with a number of documents your users are likely to use. If the time it takes to fully process most documents is acceptable, we recommend doing that. Only when you're dealing with very large or complex PDFs does the technique outlined in this section make sense.

Processing Annotations-Only to Reduce Disk Space

The typical workflow with on-disk PDFs involves reusing the information file from a previous session. Because processing is "expensive", we recommend reusing the information database for sessions beyond the first, which eliminates the need to process. Information files are often about the same size as their associated PDFs, however, so persisting these databases can double the size of your application's library.

With the annotations-only processing option, you can choose to delete information files when your users close out a PDF. You will need to reprocess every time a user opens a PDF, but by skipping the text-indexing step it goes faster. PDF outlines, hyperlinks, and previously made annotations will still become usable, so you will only need to fully process a file when a user chooses to search. At the end of a session you simply sync annotations to the PDF and delete the information file.

Syncing and Writing

`APPDFProcessor` contains several methods for persisting annotations. For starters, call `syncAnnotationsToPDF:` anytime you want to save annotations with the PDF file. This will store annotations directly with the file, to ensure they're available for future sessions (or for use in other compliant applications).

Choose `writePDFWithAnnotations:toPath:options:` to write a copy of the PDF to file. The related `writePDFWithAnnotations:toData:options:` will write the PDF to an NSData object, for secure applications. Use the options parameter to create a copy complete with all annotations, a copy with annotated pages only, or to flatten the PDF.

Searching

We provide a simple built-in UI for document search. `Call showSearch:animated:` to show a search bar in the view. For more fine-tuned control over the search experience (or to build your own search UI), just create an `APSearchRequest` from user input, run the search (preferably on a background thread), and get the results in our `APPDFSearchDelegate` methods.

Making Annotations

With our SDK adding annotations to a PDF is simple. Just call the following method.

```
APAnnotatingPDFViewController addAnnotationOfType:
```

Our SDK handles all touch events and offers a built-in annotation UI, providing control over color and other properties. You can also hide our interface and serve up your own instead.

If necessary you can create annotations programmatically too. Make sure to read the documentation for `APPDFObject`, `APAnnotation`, and other classes related to the annotation you're creating, as certain properties are required. You should also call the following methods whenever you programmatically add an annotation.

```
APPDFInformation addUserAnnotation:

APAnnotatingPDFViewController reloadAnnotationViews
```

Finally, we offer a variety of delegate methods for more fine grained control of the interface. You can discover when annotation mode begins, when it ends, when an annotation is modified, and much more.

Annotation Types

There are ten major types of annotations available in the SDK, which fall into particular categories.

1. Ink annotations are encapsulated in the `APInk` class.
2. Three types of annotations (Highlights, Underlines, and Strikeouts) are all types of `APTextMarkup`.
3. `APFileAttachment` annotations usually contain image files (Photo annotations), but can be used to attach any type of file.
4. `APSound` is a special type of file attachment that can record/playback an audio file.
5. `APText` is a note popup annotation.
6. `APFreeText` is an editable text box.
7. Stamps are placed as `APStamp` objects.

8. `APBookmark` provides a bookmark tool that's unique to our SDK (it is not part of the PDF Standard) that we provide as a convenience. Please note that, because they are not a standard type of annotation, bookmarks will not appear in other compliant PDF applications. They can still be synced to a PDF to view in your application later.

Annotation Modes

When you call **`addUserAnnotation`**: the view controller waits for touch input to place the annotation. At that point your project enters annotation mode. There are several delegate methods to help you manage how your app responds to this event. With our standard UI, your user can either cancel the annotation, which calls **`pdfController:didEndAnnotationMode:`**, or place the annotation and select Done, which calls **`pdfController:didCreateAnnotationMode:`** as well.

XML Import/Export

Our SDK offers an XML-based import/export system. These **`APPDFInformation`** methods offer you the ability to export annotations in an **`NSOutputStream`**, to later import to a duplicate PDF as an **`NSInputStream`**. The primary use case for this API is to store information about annotations on a remote server, for users who need to retain that information but don't want to transfer (potentially large) PDFs.

Another use might be transferring annotations from a document to another that has slight modifications, though importing annotations to a document that isn't an exact duplicate of the source PDF could result in crashes (contact our support team at library-support@branchfire.com if you need more details).

The XML nomenclature is based upon modeling our **`APAnnotation`** objects. This is a wholly proprietary system that will only work with our SDK (so it is not possible to import or export annotations from PDFs on a server).

Using Your Own Annotation Interface

There are two ways to go about customizing your app's interface. First of all, if you want to retain our menus but incorporate your own graphics, you can substitute in your resource files for ours. That will make small adjustments to personalize the look of your application.

If you want even greater control over your project's UI, you can choose to hide the ribbon and forego our built-in interfaces for search, bookmarks, and etc. Our SDK provides you with methods to reskin the application however you'd like.

Creating Custom Bookmark and Annotation Navigation Interfaces

Our SDK comes built-in with bookmark navigation menus, allowing your uses to navigate through PDF outlines and bookmark annotations. We also offer an annotation navigation menu, so your users can scroll through and navigate to their markups. If you don't want to use our built-in UI (e.g. you wish to mimic our end user application's Navigation Panel) we have several methods you can use to construct your own interface. Specifically, review the following:

```
APPDFInformation allUserAnnotations

APPDFInformation bookmarks

APPDFInformation outlineRoot
```

These methods provide you with all the information you need about your PDFs annotations, bookmarks, and outline elements to build your own navigation interface. You'll just need to create a UITableView or another type of view to display it.

Page Coordinates vs. View Coordinates

You are probably familiar with View Coordinates in iOS. The SDK uses a page space coordinate system for defining annotation rectangles and other purposes. Page Coordinates range from 0 to 1, representing percentages of location or dimension. So if you wanted to set an annotation's rect to be half the document's height, half its width, and centered on the page, you would do that as follows.

```
annotation.rect = CGRectMake(.25, .25, .5, .5);
```

The SDK includes methods for translating between view coordinates and page coordinates, to help you determine where a user tapped (picked up in iOS by view coordinates) relative to the page (determined in page coordinates).

Animating PDF Page Transitions

Here are a few steps to enable you to add animations (flips, curls, et al.) to page transitions.

1. Set `singlePageMode:` to `YES`. You must enable single page mode, as page transitions are visually incompatible with scrolling.
2. Set `enableSwipeToChangePages` and `slideToChangePages to YES`.
3. Call `nextPageAnimated:` wherever you allow users to change pages. Pass `NO` to prevent iAnnotate from using our built-in page slide animation.
4. Implement your animation in the `pdfControllerDidChangePage:` delegate method.

Creating a Global Document Search

Many of our developers wish to offer a global document search similar to the Full Library Search available in our end user app, so users can search through all documents simultaneously. The most straightforward way to do this with our SDK is to use our search APIs to search through all documents.

For instance, if your user searches for “spider monkeys” in your global search you can run the search in every document and present the results however you’d like. Please note that this might take some time, and would only be effective with PDFs (and their associated information files) saved to disk.

Localizing your Project

The SDK comes with a string file, AjiPDFLib.strings, located in its resources folder. If you wish to localize your project but still use our UI, you just need to create another version of AjiPDFLib.strings containing your translations. Then you can localize just as you normally would with any iOS project.

Universal Applications and PDFs on iPhone / iPod Touch

The PDF SDK compiles in universal applications. We also support rendering PDFs and annotations on the iPhone and iPod Touch. You need this functionality turned on in your license file, so if you are interested in this feature please contact our SDK Support Team (for current licensees) or your Product Consultant (for prospective licensees).

`APTextMarkup` and `APInk` annotations have a scroll/markup button in the Annotation ribbon. This allows users to scroll and zoom without accidentally placing an annotation, and return to annotating when they're ready. Also, our text markup annotations allow for two finger scrolling while in annotation mode.

Recognizing Gestures in Annotation Mode

At present it is not possible to pan or zoom in the middle of placing an ink annotation, and we do not support any attempts to implement this. Your users will have to finish the annotation via scroll/markup button if they want to pan or zoom. There are technical reasons why this is the case, and while we are considering possible workarounds for the future, at this time scrolling while still drawing is not supported.

Changing the License File and Bundle ID

You are required to use the license file and the bundle ID referenced in that file exactly as provided. Any changes to the file or ID will prevent the library from compiling. If you need to change anything about your bundle ID(s) please contact our Business Development Team to get approval.

Working on a Background Thread

We strongly recommend processing PDFs, calling `allUserAnnotations`, and performing other processor intensive tasks on a background thread. If you run these functions on a main thread you may experience a slowdown. You can see how we handle PDF processing in the `process` and `doProcessing` methods found in the `PDFAnnotationSample` sample application for further details.

Secure Information Files

If you're working with in-memory PDFs you may want to show previous annotations and offer search. In order to do that you'll need to process the PDF every time it's opened, because the information file isn't persisted to file. Instead it's kept in memory. It is not possible to persist information files stored in data, in order to avoid processing them again.

Your PDF only ever Renders with a Checkered Image

If you only see a checkered image where your PDF should be, the first thing to check is whether you're passing the appropriate view controller notifications. This is illustrated in the "passing view notifications" section of the sample project `PDFAnnotationSampleViewController.m`.

Sound Annotations and the iOS Simulator

There is a bug in the iOS audio frameworks that causes odd behavior when loading the audio system on the simulator, so this functionality only works on the device.

Documents with Multiple Page Sizes

Documents with multiple page sizes can cause issues when used with our SDK. Specifically, calling `fitToWidth` and `fitToHeight` can work in unexpected ways. These types of documents may also not play well with single page mode. If your users will be working with a lot of documents that have multiple page sizes we recommend contacting our support team at library-support@branchfire.com for workarounds and advice.

License Key Validation Errors

This error usually occurs because the license.xml file has been edited. This file must be used exactly as provided, with no changes, or else the license key validation will fail.

Your app must also use the bundle ID exactly as it's spelled out in the license.xml file. You can change the bundle ID in your project's info.plist file.

If this doesn't solve the problem, one possible issue is that when changing the bundle identifier, a known issue in Xcode may not properly reset the project. You may need to do a full clean of the project, quit Xcode, reset your iPad simulator, and then start up and do a fresh build. If that doesn't help, here are a few more things to verify:

1. Make sure the license.xml file is added to the project's resources.
2. Double-check that your license.xml file is not modified, as any modifications to that file will cause validation to fail.
3. Double-check that you are using the full and correct license key we provided to you.
4. Please verify that you are using the latest version of the library, and that it is the release version of the library.
5. Please verify that all of the project resources are added to the project (see the API documentation "project configuration" section for details).

Please let us know if you are still having issues running your application and validating the license. Make sure you paste the exact console error message that is reported by the library.

Getting Help with Your Evaluation

We're happy to support our Evaluation customers! If you want assistance please contact our team here:

library-support@branchfire.com

Please note that we handle requests for support in the order we receive them, and prioritize fully licensed customers who have payed for support. But we do everything we can to respond within two business days (often less).

Getting Help with the Full Version

We provide email support to our users through the following address:

library-support@branchfire.com

Our support staff and engineers are able to answer questions about the API, walk you through certain procedures, and take feature requests. Our turnaround time on support requests is usually fast, but we ask for up to two business days to answer, depending on the complexity of your request. We are located in Chicago, IL, and our support staff triages cases during normal business hours.

Documentation

Our Web Development Team works to keep documentation online here:

http://www.branchfire.com/lib-web/index.html

We do not have a printable copy of the documentation, but the API is searchable in Xcode. Just use Find in Workspace to search through our documentation there.

Updates

We update the SDK frequently, and notify our licensees via email. We try to keep both technical and business contacts for our customers, and notify both as soon as there's a new release. All you need to do is log into your account and download the new version and switch it out in your project. With each new version you'll get release notes describing any fixes and new APIs.