A Strategic Analysis of Offline PDF Libraries for Custom Toolchain Development

Executive Summary

This report provides an exhaustive analysis of software development kits (SDKs) for building fully offline, on-premise PDF parsing and manipulation toolchains. The evaluation focuses on libraries offering low-level control, customizability, performance, and compatibility with the Adobe PDF specification. The findings are intended to guide architectural decisions for projects where data privacy, security, and operational independence are paramount.

The PDF tooling market is bifurcating, with major vendors increasingly promoting cloud-based, API-driven services. This trend positions truly offline, embeddable SDKs as specialized, premium solutions. For organizations requiring absolute control over their document workflows, selecting the right offline library is a critical strategic decision involving trade-offs between licensing costs, performance, feature sets, and long-term maintenance risks.

Top-Ranked Commercial Solutions:
For enterprises requiring robust, vendor-supported solutions, three main contenders emerge, each with a distinct strategic value:

1. **Adobe PDF Library (PDFL):** The definitive choice for applications demanding absolute, reference-standard compatibility with the Adobe ecosystem. Its API provides unparalleled low-level access to PDF internals. However, it comes with the highest cost and a complex, partner-driven licensing model.
2. **Apryse SDK (formerly PDFTron):** A highly feature-rich and versatile contender, offering a comprehensive, self-contained toolkit built entirely in-house. It provides extensive cross-platform support and clear documentation for offline deployment, making it ideal for projects needing a wide array of functionalities from a single vendor.
3. **Foxit PDF SDK:** A strong alternative that competes on performance and offers

the most pragmatic and well-documented solutions for deployment in secure, air-gapped environments. Its explicit support for offline activation makes it a compelling choice for enterprise and government clients with stringent security protocols.

Top-Ranked Open-Source Solutions:
The open-source landscape presents a stark choice governed by the interplay of licensing, performance, and API stability:

1. **Apache PDFBox (Java):** The premier choice for projects requiring a permissively licensed library. Its Apache 2.0 license allows for royalty-free use in any commercial application. It offers a stable, well-documented, low-level API for PDF manipulation. The primary trade-off is lower performance compared to native C/C++ libraries.
2. **MuPDF (C):** The undisputed performance champion for both rendering and parsing. Its lightweight and efficient design makes it ideal for high-throughput applications. However, its default AGPL license is highly restrictive for proprietary software, effectively requiring the purchase of a commercial license from its steward, Artifex Software, for most business use cases.
3. **Poppler (C++):** A mature and widely used library, standard on many Linux systems. It offers good performance and a rich set of command-line utilities. Its restrictive GPL license and the high-risk nature of its "unstable" internal API for low-level manipulation make it a challenging choice for proprietary commercial development.

Strategic Recommendations:
The selection of a PDF library should be driven by a clear understanding of project priorities:

- **For Proprietary Commercial Products:** The decision hinges on budget and performance needs. Apache PDFBox is the only truly free option. For performance-critical applications, purchasing a commercial license for MuPDF or investing in Foxit, Apryse, or Adobe PDFL is necessary.
- **For Open-Source Projects:** The choice depends on the project's own license. PDFBox (Apache 2.0) offers the most flexibility. Poppler (GPL) and MuPDF (AGPL) impose reciprocal licensing obligations.
- **For Low-Level Control:** Adobe PDFL and Apache PDFBox provide the most stable and well-documented low-level APIs. MuPDF offers powerful low-level access with high performance. Poppler provides this access only through an explicitly unstable and unsupported internal API, posing a significant maintenance burden.

Ultimately, building a robust offline PDF toolchain requires a strategic alignment of

technical requirements with licensing and budget realities. This report provides the foundational analysis to make that decision with confidence.

Introduction: Architecting an Offline PDF Engine

Defining the Landscape: Offline SDKs vs. Cloud APIs

The modern landscape of PDF manipulation tools presents a fundamental architectural choice: integrating a self-contained, on-premise Software Development Kit (SDK) versus leveraging a cloud-based Application Programming Interface (API). This decision has profound implications for application architecture, data security, cost structure, and operational autonomy.

Adobe's product suite serves as a perfect illustration of this dichotomy. The **Adobe PDF Services API** is a collection of cloud-based REST APIs designed to perform specific, high-level document tasks such as creation, combination, Optical Character Recognition (OCR), and content extraction.[1] Developers interact with this service via SDKs for languages like Node.js, Java, and.Net, which are essentially wrappers that handle the HTTP requests to Adobe's servers.[1] As demonstrated by

curl examples requiring API keys and bearer tokens, this model is fundamentally dependent on an active internet connection.[1] The licensing model, based on "Document Transactions" with defined rate and page limits, further cements its identity as a metered service, not a self-contained library.[2] This architecture is explicitly incompatible with the requirements of a fully offline system.

In stark contrast, the target technology for this report is the offline, embeddable SDK. This type of library is integrated directly into the host application's codebase and deployed on the client's own infrastructure, whether on a server or a desktop. All processing occurs locally, with no reliance on external network calls for core functionality. Adobe's offering in this category is the **Adobe PDF Library (PDFL) SDK**. It is marketed and licensed separately from the cloud APIs, distributed through specialized partners like Datalogics, and designed for deep integration into OEM and

enterprise applications.[3] The strategic separation of these products by Adobe indicates a broader market trend: cloud APIs are being positioned as the mainstream solution for general automation, while offline SDKs are evolving into a premium category for use cases where performance, security, and absolute control over the document lifecycle are non-negotiable. This report focuses exclusively on this latter category of powerful, self-hosted libraries.

Core Attributes of a Low-Level PDF Library

To provide a rigorous and actionable comparison, all evaluated libraries will be assessed against a consistent set of core attributes derived from the primary requirements of building a custom, offline PDF toolchain.

- **Low-Level API Access:** This is the most critical technical attribute. It refers to the library's ability to provide programmatic access beyond high-level convenience functions (e.g., merge_files) and allow for direct interaction with the fundamental structures of a PDF file as defined in the ISO 32000 specification. This includes the ability to traverse, inspect, and manipulate the object graph, often referred to as the COS (Carousel Object Structure) or SDF (Structured Document Format) layer. A strong low-level API allows a developer to directly create, modify, or delete dictionaries, arrays, streams, names, and other primitive objects, and to interact with the cross-reference table (xref).
- **Customizability and Extensibility:** This attribute measures the flexibility of the library to support custom workflows and logic. A highly customizable library allows developers to, for example, implement new annotation types, create custom rendering pipelines, modify the behavior of standard functions through callbacks, or integrate their own data processing logic seamlessly with the library's data structures.
- **Performance:** Performance is a multi-faceted attribute that will be evaluated based on available data for:
    - **Parsing Speed:** The time required to open and parse a PDF document to make its internal structure accessible.
    - **Rendering Speed and Fidelity:** The time required to convert a PDF page into a raster image (e.g., a bitmap) and the accuracy of the resulting image compared to a reference renderer like Adobe Acrobat.
    - **Manipulation Efficiency:** The performance of common operations such as merging multiple documents, splitting a large document, or applying changes

like redaction or watermarking.

- **Adobe Compatibility & Specification Adherence:** The ultimate measure of a PDF library's quality is its interoperability with the broader PDF ecosystem, which is dominated by Adobe's products. This attribute assesses how well a library adheres to the official PDF specification (ISO 32000) and whether documents created or modified by the library can be reliably opened, viewed, and processed by Adobe Acrobat and other standard-compliant viewers without errors or rendering artifacts.[3]
- **Licensing and Deployment Model:** This attribute covers the legal and financial framework governing the use of the library. Key considerations include:
  - **License Type:** Whether the license is commercial (requiring payment) or open-source, and if open-source, its specific type (e.g., permissive like Apache 2.0, or copyleft like GPL/AGPL) and the obligations it imposes.
  - **Distribution Terms:** Whether the license includes royalties based on the number of distributed application copies or servers.
  - **Deployment Restrictions:** Any limitations on how the library can be deployed, such as prohibitions on server-side use for libraries intended only for interactive desktop applications.[7]

The Commercial Titans: A Deep Dive into Proprietary SDKs

For organizations requiring guaranteed support, extensive features, and legal indemnity, commercial SDKs present a compelling option. These libraries are developed by specialized companies that stake their reputation on the quality, performance, and reliability of their PDF technology. The leading contenders in the offline, on-premise space each offer a unique value proposition, making the choice between them a strategic one based on priorities such as compatibility, feature breadth, and deployment logistics.

Adobe PDF Library (PDFL): The Reference Implementation

The Adobe PDF Library (PDFL) stands as the benchmark against which all other PDF manipulation tools are measured. Its primary value proposition is authenticity: it is

built from the very same core technology that powers Adobe's flagship Acrobat products, ensuring the highest possible degree of quality, reliability, and compatibility with the official PDF specification and the vast ecosystem of Adobe-centric workflows.[3]

Core Technology and Offline Nature:
PDFL is engineered specifically for developers who need to embed Adobe's PDF functionality into their own applications, particularly in high-volume, scalable server environments.1 It is provided as a set of standalone, thread-safe C/C++ libraries that operate entirely on-premise, with no requirement for an internet connection or communication with Adobe servers for its core functions.3 It supports a broad range of platforms, including 64-bit versions of Windows, macOS, and Linux.5
Low-Level Control and API Structure:
PDFL offers unparalleled access to the internal structure of a PDF document through its comprehensive C/C++ API.9 The API is organized into distinct layers, providing different levels of abstraction. For developers requiring the deepest level of control, the **COS_Layer** is paramount. This layer provides direct, granular access to the low-level object types and file structure that constitute a PDF document.[10] Through the

COS_Layer, developers can programmatically create, read, and manipulate fundamental PDF objects, including CosDoc (the entire file), CosDict (dictionaries), CosArray (arrays), CosStream (streams for content, images, etc.), CosName, and other primitive types.[11] This allows for the construction or deconstruction of a PDF's object tree from the ground up. A detailed code example in the documentation demonstrates creating a PDF from scratch by manually constructing the page dictionary, resource dictionary, font objects, and content stream using

Cos layer functions, showcasing the depth of control available.[12] For tasks that do not require such granular control, the

PDFEdit_Layer and **PD_Layer** (PDModel) offer higher-level abstractions for manipulating page content and document-level structures like bookmarks and annotations.[10]

Features and Capabilities:
As the reference implementation, PDFL boasts a complete feature set that covers nearly every aspect of the PDF specification. Key functionalities include 3:

- **Creation and Assembly:** Programmatically generate complex PDF files and assemble new documents by inserting, deleting, or merging pages.
- **Prepress and Print:** Advanced support for professional print workflows, including color management, transparency flattening, color separations, and high-volume

variable data printing.
- **Security:** Apply password-based encryption and sophisticated digital signatures.
- **Forms:** Create and manipulate both AcroForm and XFA forms.
- **Content Extraction:** Extract text, images, and structured content from tagged PDFs.
- **Font Handling:** Full support for embedding and subsetting a wide range of fonts, including complex scripts like Arabic, Hebrew, and CJK languages.

Licensing and Distribution:
Acquiring a license for PDFL is a formal process geared towards established software vendors (ISVs), OEMs, and large enterprises. Adobe does not typically license it directly to end-users or for small projects. Instead, it is distributed through a network of channel partners, with Datalogics handling most of the world, Tech Soft 3D for CAD-related customers, and EAST for Japan.3 The pricing model reflects its premium positioning, typically involving a significant annual fee plus royalties based on product distribution or a per-server fee for SaaS or internal deployments.4 This makes it one of the most expensive options but provides the legal and technical assurance of using Adobe's own engine.

Apryse SDK (formerly PDFTron): The Feature-Rich Contender

The Apryse SDK has established itself as a leading commercial alternative to Adobe, positioning itself as a comprehensive, high-performance, and fully self-contained solution for PDF and other document format manipulation. Its core marketing message revolves around providing developers with a powerful toolkit that is built entirely in-house, ensuring quality and security without reliance on external or open-source components.[14]

Core Technology and Offline Capabilities:
Apryse emphasizes that its technology is developed from the ground up, giving it full control over the rendering pipeline and feature implementation.14 The SDK is designed for deployment on private infrastructure, allowing organizations to maintain complete control over their document workflows and data, a critical factor for industries with stringent security and compliance requirements.14
The SDK is architected for full offline functionality. The core server-side libraries for C++, Java,.NET, and Python are inherently offline, designed to be embedded within an application and run on the customer's own hardware.[16] For web-based applications using their

WebViewer component, Apryse provides detailed documentation and sample projects

demonstrating how to achieve offline operation. This is accomplished by using modern web technologies like service workers to cache all necessary application resources (JavaScript, CSS, Web Workers) and using IndexedDB or localforage to store the document data itself as a blob in the browser's local storage.[18] The API includes specific options like

startOffline to instruct the viewer to load a document directly from this local cache, enabling functionality even when the user has no internet connection.[19]

Features and Low-Level Control:
Apryse offers an exceptionally broad feature set that often goes beyond standard PDF manipulation to include processing for MS Office documents, CAD files, and various image formats.14 Key PDF features include high-fidelity viewing, programmatic creation and editing, annotation and real-time collaboration, advanced form filling and creation, true programmatic redaction, and document generation from templates.14
For low-level control, the core SDK provides a rich API that allows for deep manipulation of PDF structures. The documentation for the core C++, Java, and.NET APIs reveals a class structure that enables access to the underlying document object model, known as the SDF (Structured Document Format) layer.[16] This allows developers to traverse the document's object tree, access individual objects, and modify their properties, similar in principle to Adobe's

COS_Layer.

Licensing and Support:
The Apryse SDK is a commercial product that requires a license key for production use.20 While specific pricing is not publicly listed and is tailored to the use case, the model is built around providing a complete solution with dedicated expert support.14 They offer an unlimited trial period to allow developers to fully evaluate the SDK and build a proof-of-concept before committing to a purchase.14 This developer-centric approach, combined with the promise of owning the entire data lifecycle, makes Apryse a strong contender for businesses that want a powerful, all-in-one document processing engine without the complexities of integrating multiple open-source libraries or relying on cloud services.

Foxit PDF SDK: The Performance-Oriented Alternative

Foxit has carved out a significant market share by offering a powerful, fast, and ISO 32000-compliant PDF engine that serves as a direct competitor to Adobe and other commercial offerings. The Foxit PDF SDK is known for its high-performance rendering

and processing, and it provides a robust set of libraries for all major platforms and development languages, including C++, C#, Java, and Python.[22]

Core Technology and Offline Capabilities:
The core of the Foxit PDF SDK is its proprietary, battle-tested PDF engine, which is engineered for speed and a lightweight footprint.24 The SDK is designed to be embedded in both desktop and server applications and to operate entirely on-premise.22
Foxit's most notable strength in the context of this report is its clear and practical support for fully offline and air-gapped deployments. The documentation explicitly outlines licensing and activation strategies for environments with no internet access. **Shared Device Licensing** is specifically designed for restricted environments.[28] Furthermore, Foxit provides command-line utilities like

Activation.exe and MSI installer properties (keycode=...) that allow for offline activation using a pre-shared key code.[29] This demonstrates a deep understanding of the operational realities of enterprise IT, government, and other secure sectors where machines may never connect to the public internet. This contrasts with other solutions that may assume at least intermittent connectivity for license validation.

Features and Low-Level Control:
Foxit provides a comprehensive suite of features that rivals other top-tier commercial SDKs. This includes standard capabilities like document creation, editing, and annotation, as well as more advanced features such as:

- **Smart Forms:** Support for both standard AcroForms and dynamic XFA forms.[22]
- **Security:** Programmatic redaction, digital signatures, and encryption.[22]
- **Advanced Operations:** Enterprise-grade OCR, PDF/A conversion and validation, and full-text search.[22]

The Core API provides developers with the tools to manipulate PDF documents at a granular level. Code examples in the developer guides show the use of classes like PDFDoc and PDFPage, and methods for adding and modifying objects like annotations, demonstrating a robust object model for programmatic control.[22] While the deepest internal structures are abstracted, the API is designed to give developers the necessary control for building complex, custom workflows.

Licensing and Distribution:
Foxit offers a variety of commercial licensing models, including per-developer, per-server, and distribution-based licenses, which are tailored through their sales process.31 A free 30-day trial is available for evaluation.22 It is important for potential customers to note that, like many software companies, Foxit appears to be encouraging a shift from perpetual licenses to subscription-based models, which could have long-term cost implications.33 The EULA prohibits the redistribution of source code and sample documents, which is standard for a

commercial product.35
The choice between these commercial SDKs ultimately reflects a company's strategic priorities. Adobe PDFL is the choice for guaranteed, reference-level compatibility, especially in print and prepress workflows, but it comes at a premium price and with a more complex procurement process. Apryse offers a vast, self-contained universe of features for multiple document types, appealing to those who want a single, powerful toolkit to handle all their document processing needs. Foxit presents a compelling case based on performance and, most significantly, its pragmatic and well-defined support for true offline and air-gapped deployment, making it a standout choice for security-conscious enterprises.

The Open-Source Powerhouses: Flexibility and Control

Open-source libraries offer an alternative path for building a PDF toolchain, providing unparalleled transparency, flexibility, and freedom from vendor lock-in. However, this path requires a careful evaluation of licensing obligations, community support, and API stability. The leading open-source contenders each occupy a distinct niche in the "license-performance-stability" triangle, making the choice between them a critical architectural decision.

MuPDF: The Speed and Efficiency Champion

MuPDF is a free and open-source software framework renowned for its primary focus on speed, small code size, and high-quality, anti-aliased rendering.[36] Developed and maintained by Artifex Software, it is written in portable C and includes its own lightweight graphics library, Fitz, which contributes to its exceptional performance by avoiding dependencies on larger external libraries like Cairo.[36]

Core Technology and Performance:
MuPDF's architecture is engineered for maximum efficiency. It consistently emerges as the performance leader in various comparisons against other libraries. Anecdotal and formal benchmarks, though some are several years old, repeatedly show MuPDF to be significantly faster than Poppler, with performance gains ranging from 10% to as high as 95% in rendering tasks.37 A more recent investigation by Alfresco Engineering confirmed that MuPDF is "clearly

the fastest engine" when compared to other native and Java-based renderers.39 The developers of its popular Python binding, PyMuPDF, claim that its text extraction is 30 to 45 times faster than some pure Python alternatives, further cementing its reputation for speed.40

Low-Level Control and API:

MuPDF is designed to give developers full control over the document. The official website explicitly lists "Parse PDF Documents. Access low-level PDF objects and layout" as a core feature.41 While the core C API documentation is somewhat sparse on examples, the capabilities of the underlying library are clearly exposed through its various language bindings. The documentation for PyMuPDF, for instance, provides extensive examples of low-level interfaces, including methods to iterate through a document's cross-reference (xref) table, read the raw definition of any PDF object (xref_object), and directly read or update individual keys within a PDF dictionary (xref_get_key, xref_set_key).[42] This powerful, granular access allows developers to perform surgical modifications to a PDF's internal structure. The

mutool command-line utility, which ships with the library, also offers functions like clean and show that enable deep inspection and rewriting of PDF files, further attesting to the library's low-level prowess.[36]

Licensing—The Critical Consideration:

The licensing model is the most important factor when considering MuPDF for a project. It is dual-licensed under the GNU Affero General Public License (AGPL) v3 and a commercial license available for purchase from Artifex.36 The AGPL is a strong copyleft license that requires any software that interacts with the library over a network to be released under a compatible open-source license.36 This provision makes the AGPL version unsuitable for most proprietary commercial software, especially SaaS products. Therefore, for any closed-source application, purchasing a commercial license is the only viable path. This effectively transforms MuPDF from a "free" open-source option into a commercial one for a significant portion of its potential user base.

Community and Support:

MuPDF is under active development by Artifex Software, with a public GitHub repository showing frequent commits and maintenance activity.45 Community support is available through a Discord server and a public bug tracker.46 For those who purchase a commercial license, Artifex provides dedicated technical support, offering a direct line to the developers who build the library.41

Poppler: The Linux Standard

Poppler is a mature and robust C++ library for rendering and manipulating PDF

documents. It originated as a fork of the xpdf-3.0 codebase and has since become the de facto standard PDF engine for the Linux desktop, powering the viewers in major environments like GNOME (Evince) and KDE (Okular).[6] Its development is supported by freedesktop.org.

Core Technology and Features:
Poppler is designed to be a shared library that centralizes PDF rendering functionality.6 It uses pluggable backends for drawing, with Cairo being the most common, providing high-quality anti-aliased vector graphics rendering.6 A key strength of Poppler is its suite of command-line utilities, including
pdftotext, pdfimages, pdftohtml, pdfunite, and pdftocairo.[47] These tools are extremely powerful for scripting and building automated document processing workflows. The library itself provides a complete implementation of the ISO 32000-1 (PDF 1.7) standard and supports features like annotations and Acroforms, though it lacks support for JavaScript and full XFA forms.[6]

Low-Level Control—A High-Risk Proposition:
Poppler's approach to low-level access is a critical point of distinction and a significant source of risk. The project officially maintains stable, high-level APIs for its C++, GLib, and Qt frontends, which are intended for common tasks like rendering pages or extracting text.50 However, for the kind of deep, structural manipulation required by the user, one must venture into the library's internal C++ API, located in the
poppler/ source directory. The developers issue a stark warning about this: it is an **"UNSTABLE, INTERNAL C++ API"**.[52] The documentation explicitly states, "If you use this API, you are on your own. This API may change at any time, even among minor versions of Poppler!".[52] This means that an application built using this internal API could break with any routine update of the library, creating a significant and unpredictable long-term maintenance burden.

Licensing:
Poppler is licensed under the GNU General Public License (GPL), version 2 or later.6 The GPL is a strong copyleft license which mandates that any application that links to the Poppler library must also be distributed under the GPL.52 This makes Poppler fundamentally incompatible with closed-source, proprietary software development. Unlike MuPDF, there is no commercial licensing option available to bypass the GPL's requirements.
Community and Support:
As a cornerstone of the open-source desktop ecosystem, Poppler benefits from a large, active, and long-standing community. Development is managed through the freedesktop.org GitLab instance, which hosts the source code, issue tracker, and merge requests.6 The library is exceptionally well-maintained, with a constant stream of updates and security fixes.53 This strong community backing is a major advantage, but it does not mitigate the risks associated with its licensing or the instability of its internal API for commercial use.

Apache PDFBox: The Java Workhorse

Apache PDFBox is a pure Java open-source library for working with PDF documents, developed and maintained under the umbrella of the Apache Software Foundation.[57] Its status as a pure Java implementation makes it highly portable to any platform with a Java Virtual Machine (JVM), but this comes with a performance trade-off compared to native C/C++ libraries.

Core Technology and Performance:
PDFBox implements the PDF specification entirely in Java, with no native dependencies.57 This simplifies deployment in Java-based environments significantly. However, performance benchmarks have shown that Java-based libraries are generally slower for CPU-intensive tasks like PDF parsing and rendering compared to their native counterparts. A benchmark by Snowtide, for example, found PDFBox to be more than twice as slow as the native pdftotext utility for text extraction.[59] While PDFBox has seen performance optimizations in recent versions, developers building high-throughput, latency-sensitive applications should be mindful of this inherent performance characteristic.[60]

Low-Level Control and API:
A primary strength of PDFBox is its well-defined and stable API for low-level PDF manipulation. The library provides the org.apache.pdfbox.cos package, which is specifically designed for this purpose.61 This package contains a class for each of the fundamental object types in the PDF specification, such as
COSDocument, COSDictionary, COSArray, COSStream, COSName, COSInteger, and COSBoolean.[64] This allows a developer to programmatically construct a PDF document from scratch or to parse an existing document and traverse its internal object graph with full control. For example, one can retrieve a field's dictionary (

getCOSObject), extract the array defining its rectangular bounds (getDictionaryObject(COSName.RECT)), and then create a PDRectangle object from that low-level COSArray.[65] This stable, object-oriented approach to low-level access is a significant advantage over the high-risk, unstable internal API offered by Poppler.

Features:
PDFBox provides a solid set of core functionalities, including creating new documents, manipulating existing ones, extracting text and images, filling interactive forms (AcroForms), splitting and merging documents, and adding digital signatures.58 It is important to note that PDFBox is intentionally a low-level library; it does not provide a high-level layout engine for

automatically flowing text or creating complex tables. Developers are responsible for calculating and specifying the exact coordinates for all content placed on a page, which reinforces its suitability for building custom toolchains that require precise control.57

Licensing—The Decisive Advantage:

The single greatest advantage of Apache PDFBox for commercial development is its license. It is distributed under the Apache License 2.0, a permissive, business-friendly license.57 This license allows developers to freely use, modify, and distribute the library as part of both open-source and proprietary, closed-source commercial applications without any requirement to disclose their own source code or pay licensing fees. This removes all legal and financial friction associated with licensing, making it the default choice for many corporate development projects.

Community and Support:

As a top-level Apache project, PDFBox benefits from a well-established governance model and a strong community. Support is available through official mailing lists, and development is managed via a public issue tracker.58 The project is actively maintained, with regular releases and security updates visible in its source repository.71

The choice within the open-source realm is therefore a strategic one. Apache PDFBox offers unparalleled legal safety and a stable low-level API, making it the lowest-risk option, especially for Java shops, with the main trade-off being performance. MuPDF offers the highest performance but its AGPL license pushes most commercial users towards a paid license, blurring the line between open-source and commercial. Poppler offers good performance and a vibrant community but is hamstrung by its restrictive GPL license and the explicit instability of the very low-level API that a power user would need.

Head-to-Head: Comparative Analysis and Benchmarks

A direct, side-by-side comparison of features, performance, and API characteristics is essential for making an informed decision. This section synthesizes the available data into comparative tables and analyses to highlight the key differentiators between the leading commercial and open-source PDF libraries.

Master Feature Comparison Matrix

The following matrix provides a consolidated overview of the core capabilities of each

evaluated library. It is designed to serve as a quick reference for assessing which tool best aligns with specific project requirements.

Feature	Adobe PDF Library	Apryse SDK	Foxit PDF SDK	MuPDF	Poppler	Apache PDFBox
Primary Language	C/C++	C++ (bindings available)	C++ (bindings available)	C (bindings available)	C++	Java
License	Commercial (Royalty-based)	Commercial	Commercial	AGPLv3 / Commercial	GPLv2+	Apache 2.0
Low-Level API Stability	Stable	Stable	Stable	Stable	**Unstable** (Internal API)	Stable
--- Creation & Manipulation ---						
Create from Scratch	Yes [3]	Yes [14]	Yes [22]	Yes [41]	Limited (via utils)	Yes [58]
Page Manipulation	Yes [3]	Yes [14]	Yes [22]	Yes [41]	Yes (via pdfunite/pdfseparate) [47]	Yes [58]
Text/Image Editing	Yes [3]	Yes [14]	Yes [22]	Yes [41]	No	Yes (low-level) [57]
--- Content Extractio						

n ---						
Text Extraction (Layout)	Yes [3]	Yes [41]	Yes [22]	Yes [41]	Yes (via pdftotext) [47]	Yes [58]
Image Extraction	Yes [3]	Yes [41]	Yes [22]	Yes (via mutool) [43]	Yes (via pdfimages) [47]	Yes [57]
--- Forms ---						
AcroForm (Fill/Create)	Yes [4]	Yes [14]	Yes [22]	Yes (optional support) [36]	Partial (Fill) [6]	Yes [58]
XFA Support	Yes [4]	Limited	Yes (Add-on) [22]	No [6]	No [6]	No
--- Security ---						
Encryption/Decryption	Yes [3]	Yes [14]	Yes [22]	Yes [74]	Yes (via utils)	Yes [58]
Programmatic Redaction	Yes [4]	Yes [14]	Yes [22]	Yes [41]	No	Limited (overwrite)
Digital Signatures	Yes [5]	Yes [41]	Yes [22]	Yes [41]	Yes (Validate) [47]	Yes [58]
--- Advanced Features ---						

Rendering (to bitmap)	Yes [5]	Yes [41]	Yes [22]	Yes [36]	Yes (via pdftoppm/ pdftocairo) [47]	Yes [58]
OCR	Yes [4]	Yes (Add-on) [15]	Yes (Add-on) [22]	Yes (via Tesseract) [75]	No	No
PDF/A Conversion	Yes [3]	Yes [14]	Yes [22]	Limited	Limited	Yes (Validate) [58]
Font Embedding	Yes [3]	Yes [16]	Yes [22]	Yes	Yes	Yes [58]
--- Deployment ---						
Server-Side Use	Yes [3]	Yes [14]	Yes [22]	Yes (license dependent)	Yes (license dependent)	Yes
Explicit Offline Activation	Yes (via partner)	Yes (Core SDK)	Yes (Command-line) [29]	Yes (Core lib)	Yes	Yes

Performance Benchmark Analysis

No single, comprehensive, up-to-date benchmark exists that compares all six of these libraries across a wide range of manipulation tasks. However, by synthesizing data from various independent tests and vendor claims, a clear performance hierarchy can be established.

Native (C/C++) vs. Managed (Java):
The most definitive conclusion from the available data is that libraries written in native code (C/C++) consistently and significantly outperform those written in managed code (Java). An

independent benchmark focused on text extraction demonstrated this clearly: the native C/C++ pdftotext utility (from the Poppler/Xpdf family) was approximately 2.25 times faster than Apache PDFBox.59 This performance gap is fundamental to the technologies; native code compiles directly to machine instructions, avoiding the overhead of a virtual machine layer like the JVM. For applications where throughput and low latency are critical, native libraries hold a distinct advantage.

MuPDF vs. Poppler:

Within the native open-source world, MuPDF is consistently cited as the performance leader. A direct comparison rendering pages to images found MuPDF to be, on average, 78% faster than Poppler, and up to 95% faster in some cases.37 Other sources corroborate this, suggesting MuPDF is anywhere from 10% to 50% faster depending on the specific file and task.38 This performance edge is largely attributed to MuPDF's lightweight design and its use of its own highly optimized Fitz graphics library.37 The developers of PyMuPDF (the Python binding for MuPDF) also claim its text extraction is roughly three times faster than pdftotext, further solidifying MuPDF's position at the top of the performance hierarchy.[40] An extensive evaluation by Alfresco Engineering for a new rendering engine also concluded that "MuPDF is clearly the fastest engine" among all native and Java options they tested.[39]

Commercial Library Performance:

Direct, independent benchmarks for the commercial libraries are scarce. These vendors build their reputations on providing high-performance solutions, and their SDKs are expected to be highly optimized. Adobe PDFL, as the reference implementation, should have performance on par with Acrobat itself. Foxit and Apryse both market their SDKs as high-performance alternatives. Without independent data, these claims should be validated with proof-of-concept testing, but it is reasonable to assume their performance is competitive and will generally exceed that of a pure Java library like PDFBox.

Performance Scaling:

The concept of performance scaling is also crucial. Some libraries may have a higher initial startup cost but scale efficiently with document size or complexity, while others may have a lower startup cost but scale linearly. For example, recent benchmarks on data extraction frameworks showed that some tools process documents at a near-constant time regardless of page count (suggesting efficient streaming or parsing), while others show a linear increase in processing time with the number of pages.76 This is an important consideration for architects designing systems that must handle documents of varying sizes.

Low-Level API Access Comparison

The user's requirement for "low-level control" necessitates a specific comparison of how each library exposes the internal workings of a PDF file. The stability, language,

and design philosophy of these deep APIs are critical differentiators.

Library	API Name / Package	API Language	Stability	Key Strengths	Key Weaknesses / Risks
Adobe PDF Library	COS_Layer	C/C++	**Stable**	The reference implementation; complete and authoritative access to all PDF object types. Guarantees compatibility.[10]	Highest cost; licensing is complex and managed through partners.[4]
Apache PDFBox	org.apache.pdfbox.cos	Java	**Stable**	Well-documented, object-oriented, easy to use within a Java ecosystem. Permissive Apache 2.0 license.[61]	Performance overhead inherent to the JVM compared to native libraries.[59]
MuPDF	Core C API	C	**Stable**	Highest performance; provides direct object and stream manipulation capabilities. Commercially supported by Artifex.[41]	Default AGPL license is unsuitable for most proprietary use, necessitating a commercial license purchase.[36]
Poppler	Internal C++ API (poppler/	C++	**UNSTABLE**	Provides direct	Explicitly unsupported

	dir)			access to the library's internal object representation.[52]	; API can and does break between minor versions, creating high maintenance risk. Restrictive GPL license.[52]
Apryse SDK	Core SDF/Cos API	C++, Java,.NET	**Stable**	Vendor-supported and documented; part of a comprehensive, feature-rich toolkit.[14]	Closed-source, making the underlying implementation opaque. Commercial cost.
Foxit PDF SDK	Core API	C++, Java,.NET	**Stable**	Vendor-supported and documented; known for high performance and excellent offline deployment support.[22]	Closed-source, implementation is opaque. Commercial cost.

This comparison reveals a clear trade-off. For the safest, most stable low-level access, Adobe PDFL and Apache PDFBox are the top choices, with the decision between them driven by language preference (C++ vs. Java) and budget. For the highest performance, MuPDF's C API is the best technical option, but its licensing forces a commercial decision. Poppler offers the desired access but at an unacceptably high risk for any production system due to the explicit instability of its internal API. The commercial offerings from Apryse and Foxit provide a stable, supported, but opaque alternative for those who prefer a single-vendor solution.

Final Rankings and Strategic Recommendations

The selection of an offline PDF library is a foundational architectural decision that extends beyond a simple feature checklist. It requires a strategic assessment of the project's core priorities, including licensing constraints, performance requirements, budget, and tolerance for long-term maintenance risk. The following recommendations are ranked according to these distinct strategic priorities to provide an actionable framework for decision-making.

Ranked Recommendations by Strategic Priority

For Uncompromising Adobe Compatibility and Features (Cost is not a primary constraint)

When the primary goal is to ensure absolute fidelity with the Adobe ecosystem and access the most comprehensive feature set, a top-tier commercial SDK is the only viable option.

1. **Adobe PDF Library (PDFL):** As the reference implementation built from the same code as Acrobat, PDFL guarantees the highest possible level of compatibility and adherence to the PDF specification.[3] It offers unparalleled low-level control through its stable COS_Layer API and a complete set of features for creation, prepress, and security.[3] It is the definitive choice when the application's output must be flawless in any Adobe-centric workflow. The trade-off is its high cost and complex, partner-mediated licensing process.[4]
2. **Apryse SDK:** A very close second, Apryse offers a massive, self-contained feature set that often extends beyond PDF to other document formats.[14] Its commitment to in-house development ensures a consistent and well-supported toolkit.[14] With excellent cross-platform support and clear guidance for offline deployment, it is a powerful choice for projects that need a versatile, all-in-one solution from a single vendor.

3. **Foxit PDF SDK:** A strong and mature competitor, Foxit provides a comprehensive feature set, including robust support for XFA forms and advanced security features.[22] It is highly regarded for its performance and, crucially, offers the most practical and well-documented support for offline activation in secure, air-gapped environments.[28]

For Maximum Performance (Willing to consider commercial licenses)

When application throughput, rendering speed, or processing latency are the most critical factors, native C/C++ libraries are paramount.

1. **MuPDF (with Commercial License):** The available evidence consistently points to MuPDF as the performance champion among all evaluated libraries, both open-source and commercial.[37] Its lightweight, focused design makes it exceptionally fast for rendering and parsing. For proprietary projects, purchasing a commercial license from Artifex is necessary to bypass the restrictive AGPL license, but this provides the best of both worlds: top-tier performance and the legal certainty required for business applications.[36]
2. **Foxit PDF SDK:** Foxit has built its brand on performance, and its native C++ engine is engineered for speed.[24] It is a strong commercial alternative for performance-critical applications.
3. **Adobe PDF Library (PDFL):** As the engine behind Acrobat, PDFL is highly optimized and can be expected to deliver excellent performance, especially in complex rendering and print-production scenarios.[3]

For Permissive Open-Source Integration (Primarily for Java-based projects)

When the primary driver is the need for a truly free and open-source library with a permissive license that can be used in proprietary commercial products without fees or source code disclosure.

1. **Apache PDFBox:** In this category, PDFBox stands alone. Its Apache 2.0 license provides maximum legal freedom.[57] It offers a robust, stable, and well-documented low-level API (org.apache.pdfbox.cos) for granular control over PDF structures.[64] Its pure Java

nature ensures easy integration into any Java-based project. The only significant trade-off is its performance, which is inherently lower than that of the native C/C++ libraries.[59] For applications where performance is not the absolute top priority, PDFBox is the safest and most cost-effective open-source choice.

For Open-Source Integration (For C/C++ projects, with license considerations)

When the development environment is C/C++ and an open-source solution is preferred, the choice is constrained by licensing and API stability.

1. **MuPDF (with AGPL compliance):** If the project is itself open source and can comply with the terms of the AGPL (or is for internal use where distribution is not a concern), MuPDF is the superior technical choice due to its outstanding performance and powerful C API.[36]
2. **Poppler:** Poppler is a viable choice only under a specific set of circumstances: the host application must be licensed under the GPL, and the development team must be willing to accept the significant long-term maintenance risks associated with using its **unstable internal API** for low-level manipulation.[52] For most professional development, this combination of a restrictive license and an unstable API makes it a high-risk option compared to the alternatives.

A Decision Framework for Your Toolchain

To assist in navigating these strategic trade-offs, the following decision framework can guide the selection process:

Step 1: Define Your Licensing and Budgetary Stance.

- **Question:** Are you building a proprietary, closed-source commercial product?
  - **If YES:** You must either use a permissively licensed library (Apache PDFBox) or purchase a commercial license. The GPL/AGPL licenses of Poppler and MuPDF are not suitable for proprietary distribution.
  - **If NO (building an open-source project):** Evaluate the license compatibility. Apache 2.0 (PDFBox) is the most flexible. GPL (Poppler) and AGPL (MuPDF) impose copyleft obligations on your project.

- **Question:** What is your budget for this component?
  - **Zero/Minimal:** Apache PDFBox is your only option for proprietary use.
  - **Moderate:** A commercial license for MuPDF may be a cost-effective way to get top-tier performance.
  - **Substantial:** The full commercial SDKs from Foxit, Apryse, and Adobe PDFL are all within scope.

Step 2: Define Your Performance Requirements.

- **Question:** Is your application a high-throughput server, or is low-latency processing critical?
  - **If YES:** Prioritize native C/C++ libraries. MuPDF (with a commercial license) is the top candidate, followed by Foxit, Apryse, and Adobe PDFL. Apache PDFBox's performance may not be sufficient.
  - **If NO:** The performance of Apache PDFBox is likely acceptable, and its other benefits (license, API stability) may outweigh the speed difference.

Step 3: Define Your Need for Low-Level Control vs. API Stability.

- **Question:** Do you require deep, structural manipulation of PDF objects?
  - **If YES:** You need a library with a strong, stable low-level API.
    - For maximum stability and guaranteed compatibility, choose **Adobe PDFL**.
    - For a stable, well-documented API in a Java environment, choose **Apache PDFBox**.
    - For a stable, high-performance API in a C environment, choose **MuPDF** (with a commercial license).
    - **Avoid Poppler's internal API** for production systems unless you are prepared for the high maintenance risk of its "unstable" status.

By systematically addressing these three questions—License, Performance, and Stability—an organization can navigate the complex PDF library landscape and select a toolchain that is not only technically sound but also strategically aligned with its business goals, risk tolerance, and budget.

Works cited

1. SDK Developer Kit | PDF Library | Adobe Acrobat Services, accessed July 17, 2025, https://developer.adobe.com/document-services/
2. Licensing and Usage Limits - Adobe Developer, accessed July 17, 2025, https://developer.adobe.com/document-services/docs/overview/limits/
3. Adobe PDF Library SDK Datasheet - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdflsdk/pdfldatasheet.pdf

4. Understanding our PDF SDK Pricing and Licensing - Datalogics, accessed July 17, 2025, https://www.datalogics.com/understanding-sdk-pricing
5. PDFL Overview - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdflsdk/
6. Poppler (software) - Wikipedia, accessed July 17, 2025, https://en.wikipedia.org/wiki/Poppler_(software)
7. Re: Acrobat SDK vs. PDF Library SDK (especially for AcroForms) - Adobe Community, accessed July 17, 2025, https://community.adobe.com/t5/acrobat-sdk-discussions/acrobat-sdk-vs-pdf-library-sdk-especially-for-acroforms/m-p/8474888
8. License for SDK use in company's internal application - Adobe Community, accessed July 17, 2025, https://community.adobe.com/t5/acrobat/license-for-sdk-use-in-company-s-internal-application/m-p/8955725
9. PDF Library API Reference - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdflsdk/apireference/index.html
10. All Layers - PDF Library API Reference - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdflsdk/apireference/package-summary.html
11. COS_Layer package - PDF Library API Reference - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdflsdk/apireference/COS_Layer/package-detail.html
12. Working with Cos Objects - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/library/plugin/Plugins_Cos.html
13. All Objects - PDF Library API Reference - Adobe Open Source, accessed July 17, 2025, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdflsdk/apireference/class-summary.html
14. PDFTron - PDF SDK: Fully Customizable, Add 100s of Features - XLsoft, accessed July 17, 2025, https://www.xlsoft.com/en/products/pdftron/index.html
15. Apryse PDF SDK | Customizable PDF functionalities for all platforms, accessed July 17, 2025, https://apryse.com/products/core-sdk
16. PDFTron SDK API documentation, accessed July 17, 2025, https://sdk.apryse.com/api/PDFTronSDK/index.html
17. Apryse documentation, accessed July 17, 2025, https://docs.apryse.com/
18. Viewing documents offline with WebViewer - Apryse documentation, accessed July 17, 2025, https://docs.apryse.com/web/guides/offline-loading
19. PDFTron WebViewer Namespace: Core - Apryse Documentation, accessed July 17, 2025, https://sdk.apryse.com/api/web/8.1/Core.html
20. Steps to upgrade to PDFNet SDK v.4 - Google Groups, accessed July 17, 2025, https://groups.google.com/g/pdfnet-sdk/c/AtSqWg5KKIk

21. README.md - ApryseSDK/pdftron-react-native - GitHub, accessed July 17, 2025, https://github.com/ApryseSDK/pdftron-react-native/blob/master/README.md
22. Foxit PDF SDK for Windows - Foxit Developers | PDF SDK technology, accessed July 17, 2025, https://developers.foxit.com/products/windows/
23. Developer Guide for Foxit PDF SDK for Node.js (10.0), accessed July 17, 2025, https://developers.foxit.com/developer-hub/document/developer-guide-pdf-sdk-node-js-10-0/
24. Developer Guide for Foxit PDF SDK for Java API (10.1), accessed July 17, 2025, https://developers.foxit.com/developer-hub/document/developer-guide-pdf-sdk-java/
25. Developer Guide for Foxit PDF SDK for .NET (10.1), accessed July 17, 2025, https://developers.foxit.com/developer-hub/document/developer-guide-pdf-sdk-net/
26. Developer Guide for Foxit PDF SDK for C (10.1), accessed July 17, 2025, https://developers.foxit.com/developer-hub/document/developer-guide-pdf-sdk-c
27. Foxit PDF SDK, accessed July 17, 2025, http://cdn01.foxitsoftware.com/pub/foxit/manual/en_us/FoxitPDFSDK5_3_DeveloperGuideforWin10_UWP.pdf
28. Remote Desktop Services Deployments of Foxit PDF Editor, accessed July 17, 2025, https://kb.foxit.com/s/articles/4841713129876-Remote-Desktop-Services-Deployments-of-Foxit-PDF-Editor
29. When To Use Which Command Prompt Line for Offline Activation - Foxit Support, accessed July 17, 2025, https://kb.foxit.com/s/articles/7181601954452-When-To-Use-Which-Command-Prompt-Line-for-Offline-Activation
30. Foxit PDF Editor Activation Methods, accessed July 17, 2025, https://kb.foxit.com/s/articles/360058834332-Foxit-PDF-Editor-Activation-Methods
31. .NET SDKs End User License Agreement | Foxit PDF Library, accessed July 17, 2025, https://developers.foxit.com/net-sdk-end-user-license-agreement/
32. How to Use Foxit PDF SDK for Web Offline | JavaScript based PDF Viewer, accessed July 17, 2025, https://developers.foxit.com/developer-hub/document/use-foxit-pdf-sdk-web-offline/
33. Foxit is phasing out perpetual licenses : r/sysadmin - Reddit, accessed July 17, 2025, https://www.reddit.com/r/sysadmin/comments/1l7rfeh/foxit_is_phasing_out_perpetual_licenses/
34. Foxit Resource Hub, accessed July 17, 2025, https://www.foxit.com/resource-hub/?page=1&search=&type=34
35. Foxit PDF SDK (dotnet), accessed July 17, 2025, http://cdn01.foxitsoftware.com/pub/foxit/manual/en_us/FoxitPDFSDK5_2_DeveloperGuidefor.Net.pdf

36. MuPDF - Wikipedia, accessed July 17, 2025, https://en.wikipedia.org/wiki/MuPDF
37. Poppler vs MuPDF - Johnny Huang, accessed July 17, 2025, https://hzqtc.github.io/2012/04/poppler-vs-mupdf.html
38. is MuPdf library faster than xpdf/poppler at rendering images from pdf pages?, accessed July 17, 2025, https://stackoverflow.com/questions/7322768/is-mupdf-library-faster-than-xpdf-poppler-at-rendering-images-from-pdf-pages
39. PDF rendering engine performance and fidelity comparison - Hyland Connect, accessed July 17, 2025, https://connect.hyland.com/t5/alfresco-blog/pdf-rendering-engine-performance-and-fidelity-comparison/ba-p/125428
40. Text Extraction Using PyMuPDF - Medium, accessed July 17, 2025, https://medium.com/@pymupdf/text-extraction-using-pymupdf-4572b97c6a58
41. MuPDF: The ultimate library for managing PDF documents, accessed July 17, 2025, https://mupdf.com/
42. Low-Level Interfaces - PyMuPDF 1.26.3 documentation, accessed July 17, 2025, https://pymupdf.readthedocs.io/en/latest/recipes-low-level-interfaces.html
43. MuPDF Core, accessed July 17, 2025, https://mupdf.com/core
44. MuPDF Android SDK - Debian Sources, accessed July 17, 2025, https://sources.debian.org/data/main/m/mupdf/1.17.0%2Bds1-2/docs/android-sdk.html
45. Activity · ArtifexSoftware/mupdf - GitHub, accessed July 17, 2025, https://github.com/ArtifexSoftware/mupdf/activity
46. ArtifexSoftware/mupdf - GitHub, accessed July 17, 2025, https://github.com/ArtifexSoftware/mupdf
47. Poppler-24.08.0 - Linux From Scratch!, accessed July 17, 2025, https://www.linuxfromscratch.org/blfs/view/12.2/general/poppler.html
48. How-To Work With Poppler Utility Library (PDF Tool) - Foxtrot Alliance, accessed July 17, 2025, https://foxtrot-alliance.zendesk.com/hc/en-us/articles/360025802252
49. Poppler-0.85.0 - Linux From Scratch!, accessed July 17, 2025, https://www.linuxfromscratch.org/blfs/view/9.1/general/poppler.html
50. Poppler - library for rendering PDF files, and examining or modifying their structure, accessed July 17, 2025, https://www.linuxlinks.com/poppler-pdf-rendering-library/
51. Poppler, accessed July 17, 2025, https://poppler.freedesktop.org/
52. janl/poppler - GitHub, accessed July 17, 2025, https://github.com/janl/poppler//
53. innodatalabs/poppler - GitHub, accessed July 17, 2025, https://github.com/innodatalabs/poppler
54. Distrotech/poppler-data: Mirror of git://anongit.freedesktop.org/poppler/poppler-data - GitHub, accessed July 17, 2025, https://github.com/Distrotech/poppler-data
55. Activity · blackbeam/poppler-simple - GitHub, accessed July 17, 2025, https://github.com/blackbeam/poppler-simple/activity
56. Download Poppler binaries packaged for Windows with dependencies - GitHub,

accessed July 17, 2025, https://github.com/oschwartz10612/poppler-windows
57. Apache PDFBox Java Tutorial: How to Generate PDFs | PDFBolt, accessed July 17, 2025, https://pdfbolt.com/blog/apache-pdfbox-java-tutorial-generate-pdfs
58. Apache PDFBox | A Java PDF Library, accessed July 17, 2025, https://pdfbox.apache.org/
59. A performance comparison of PDF text extraction libraries - Snowtide, accessed July 17, 2025, https://www.snowtide.com/performance
60. Performance iText vs.PdfBox (2014) - java - Stack Overflow, accessed July 17, 2025, https://stackoverflow.com/questions/22340674/performance-itext-vs-pdfbox-2014
61. Overview (Apache PDFBox 3.0.2 API) - javadoc.io, accessed July 17, 2025, https://javadoc.io/doc/org.apache.pdfbox/pdfbox/3.0.2/overview-summary.html
62. Apache PDFBox 3.0.5 API - javadoc.io, accessed July 17, 2025, https://javadoc.io/doc/org.apache.pdfbox/pdfbox/latest/index.html
63. Overview (Apache PDFBox 2.0.0 API), accessed July 17, 2025, https://pdfbox.apache.org/docs/2.0.0/javadocs/
64. Package org.apache.pdfbox.cos, accessed July 17, 2025, https://pdfbox.apache.org/docs/2.0.8/javadocs/org/apache/pdfbox/cos/package-summary.html
65. How to get the position of a field using pdfbox? - java - Stack Overflow, accessed July 17, 2025, https://stackoverflow.com/questions/14868059/how-to-get-the-position-of-a-field-using-pdfbox
66. Apache PDFBox Overview - Tutorialspoint, accessed July 17, 2025, https://www.tutorialspoint.com/pdfbox/pdfbox_overview.htm
67. PDFBox - Tutorialspoint, accessed July 17, 2025, https://www.tutorialspoint.com/pdfbox/pdfbox_tutorial.pdf
68. pdfbox-docs/content/3.0/faq.md at master · apache/pdfbox-docs · GitHub, accessed July 17, 2025, https://github.com/apache/pdfbox-docs/blob/master/content/3.0/faq.md
69. Mirror of Apache PDFBox - GitHub, accessed July 17, 2025, https://github.com/apache/pdfbox
70. PDFBox - Confluence Mobile - Apache Software Foundation, accessed July 17, 2025, https://cwiki.apache.org/confluence/display/INCUBATOR2/PDFBoxProposal
71. Activity · apache/pdfbox - GitHub, accessed July 17, 2025, https://github.com/apache/pdfbox/activity
72. Activity · apache/pdfbox-docs - GitHub, accessed July 17, 2025, https://github.com/apache/pdfbox-docs/activity?ref=master
73. Activity · apache/pdfbox-docs - GitHub, accessed July 17, 2025, https://github.com/apache/pdfbox-docs/activity
74. MuPDF.NET documentation, accessed July 17, 2025, https://mupdfnet.readthedocs.io/en/stable/
75. MuPDF.NET: The Sharp Addition to the MuPDF Family | Medium, accessed July 17, 2025,

https://medium.com/@pymupdf/mupdf-net-the-sharp-addition-to-the-mupdf-family-d6cbb6d8c73b
76. PDF Data Extraction Benchmark 2025: Comparing Docling, Unstructured, and LlamaParse for Document Processing Pipelines - Procycons, accessed July 17, 2025, https://procycons.com/en/blogs/pdf-data-extraction-benchmark/