Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

June 2026. Vol. 5. No. 3; e-ISSN: 2808-4519

Development of a Python-Based Prepress Application for Document Color and Size Conversion in Print-Ready Files

Muamar Abid Rabbani[1]*, Yoga Putra Pratama[2], Sahiba Sahila[3]

[1,2,3]*Politeknik Negeri Jakarta*

muamar.abid.rabbani.tgp22@mhsw.pnj.ac.id[1*], yoga.putra.pratama@grafika.pnj.ac.id[2], sahiba.sahila@lecturer.pnj.ac.id[3]

Abstract

The prepress stage plays a critical role in ensuring the feasibility of design documents before entering the printing process, helping prevent printing defects. However, the conventional checking and processing of print-ready documents are still performed manually, leading to time inefficiency. At the same time, commercially available software in the current industry tends to be costly and possesses complex system architectures. Therefore, this study aims to design a simpler Python-based prepress application to validate color modes and perform automatic paper-size conversions. The method employed is experimental, utilizing a Research and Development (R&D) approach. The program was designed using the Python programming language, leveraging the PyMuPDF library for internal metadata extraction and CustomTkinter for the graphical user interface (GUI) design. Testing of the conversion results was conducted by comparing parameters against Adobe Photoshop, alongside assessing the system's computational processing speed within the application. The results indicate that the application successfully extracted document metadata, converted color modes into pure CMYK, and adjusted paper dimensions accurately. This study concludes that the developed Python-based preflight application is capable of streamlining the workflow of print-ready document inspections, minimizing the risk of human error, and enhancing prepress production time efficiency for small to medium-scale printing industries.

Keywords: *Prepress, Application, PDF, Adobe Photoshop*

1. Introduction

The modern printing industry linearly involves three interconnected stages: prepress, press, and finishing. Among these, the prepress stage plays the most crucial role in verifying the feasibility of design documents before entering mass production, as the smallest detailed error at this phase can trigger significant print quality defects and substantial material losses. [1]. Although document quality inspection is critical, manual checking methods are deemed inefficient due to being time-consuming, while commercially available automated software in the current market suffers from expensive licensing costs and high system complexity for novice operators [2]. To bridge this gap, this study focuses on developing a much simpler, standalone inspection application utilizing the Python programming language. By minimizing the system architecture to focus solely on core parameters such as pure CMYK color space validation, paper dimension adjustments, and document resolution verification, this application is designed to streamline the prepress workflow, reduce production lead time, and mitigate the risk of human error in small to medium-scale printing industries.

2. Theoretical Basic

2.1. Prepress

Prepress is a very important phase in the printing process as it involves the digital preparation of the files to be printed according to the requirements of the printing technology used [1]. Inspection standardization within production activities plays a vital role in the graphic arts industry. The absence of standardized benchmarks during the prepress phase is frequently the primary trigger for various technical defects, such as geometric inconsistencies and color deviations. The lack of clear visual inspection guidelines not only significantly increases the risks of production rejects and customer complaints, but also contributes to a high average defect rate during the prepress process, which reaches up to 15% in the packaging and label manufacturing sector [3].

2.2. Python

Python is a programming language first developed in the late 1980s by Guido van Rossum, a Dutch research programmer [4]. Python serves as a foundation for building automated systems capable of conducting in-depth analysis of digital document metadata and processing them into print-ready files.

2.3. Portable Document Format (PDF)

Portable Document Format (PDF) is a digital file format widely utilized across the globe for electronic document exchange, academic publications, and the graphic arts industry due to its capability to maintain content integrity regardless of the software or operating system used. Within the digital printing workflow, PDF serves as the primary container carrying all technical information necessary for the production process, where its processing efficiency is highly dependent on the optimization of the imposition workflow and the compatibility of elements within the document to achieve consistent print quality [5].

2.4. PyMuPDF

PyMuPDF is a high-level Python library designed for exceptionally fast and efficient PDF document processing [6]. It facilitates the extraction of critical metadata, such as color space detection, image resolution, and font embedding status, which are highly crucial to ensuring the printability of a document [7].

2.5. CustomTkinter

CustomTkinter is a modern extension of the standard Tkinter library that provides a more aesthetically pleasing and professional graphical user interface (GUI). This library enables developers to create desktop applications that support contemporary features, such as dark mode, alongside dynamically customizable visual elements [8] [9]. Within this study, CustomTkinter plays a vital role in building the control interface, which facilitates prepress operators in managing documents and monitoring production status indicators in real-time.

3. Research Methods

This study applies an experimental method utilizing a Research and Development (R&D) approach, focusing on the design and development phases of the application. Upon completion of the system construction phase, testing was conducted to evaluate the application's functionality by comparing its output accuracy against Adobe Photoshop, alongside measuring the efficiency of the system's computational performance.

3.1. Application Workflow

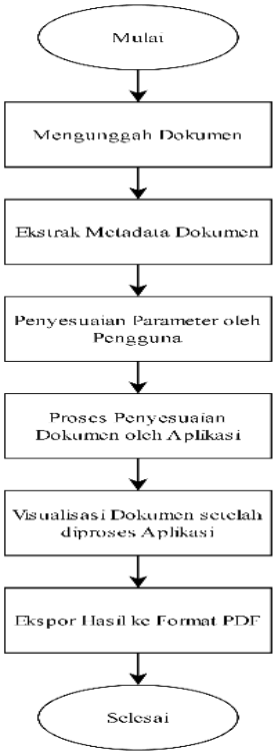

Fig. 1: Application Workflow

The application workflow begins with uploading a PDF document to automatically extract its metadata before operators select the target paper size and color mode. Based on these configurations, the system processes the file to generate a visual preview while simultaneously displaying critical technical specifications.

3.2. Application System Workflow

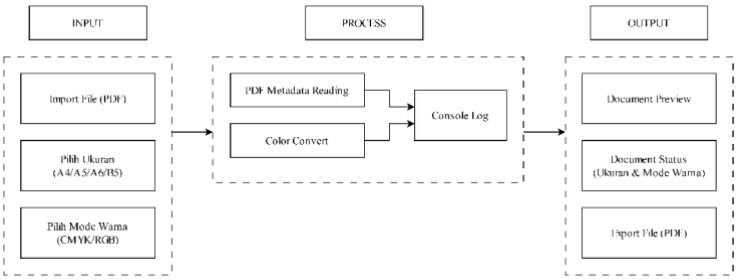

Fig. 2: System Block Diagram

The application architecture processes PDF inputs through automated metadata reading and CMYK color conversion based on user-defined dimensions and color parameters. The resulting system output delivers an immediate visual preview, a validation status report, and an exportable print-ready PDF file.

3.3. Graphical User Interface Design

The user interface was designed using Adobe Illustrator to visualize the application's layout and provide an operational overview for users. This design phase also included creating guide specifications for every button, indicator, and visual element within the interface.

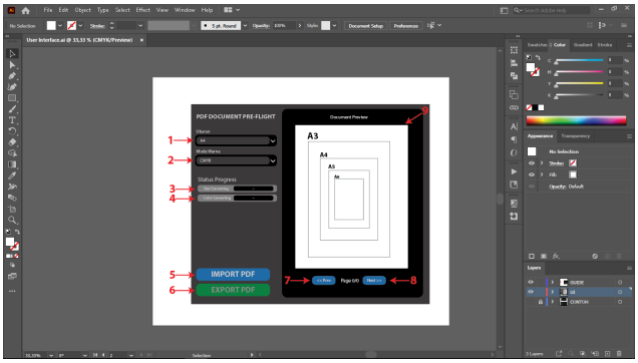

Fig. 3: GUI Designing in Adobe Illustrator

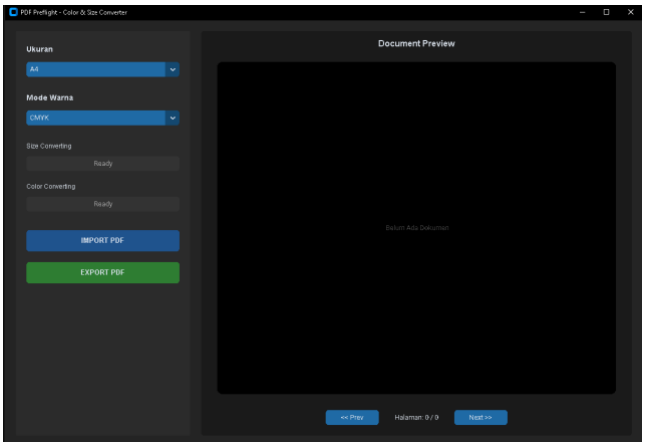

Fig. 4: GUI Design with CustomTkinter

Following the aesthetic design phase, utilizing Adobe Illustrator, the graphic assets were integrated into the application architecture using the CustomTkinter library. This transformation from static layouts into functional source code aims to realize a highly responsive digital user interface. This implementation ensures that the system is not only computationally robust but also delivers an intuitive and adaptive operational experience for the end-user.

3.4. Application Backend Development

The backend architecture was engineered to ensure functional interactivity by leveraging PyMuPDF and PyPDF2 for advanced PDF metadata extraction and dimensional conversion. Additionally, the Pillow library was integrated to bridge data processing between PyMuPDF and the CustomTkinter user interface.

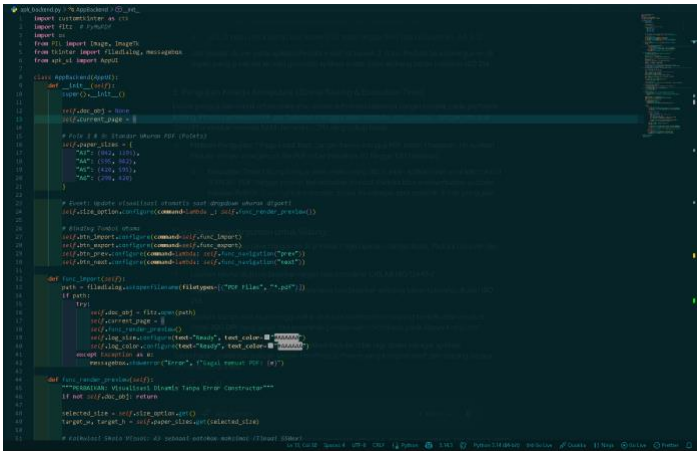

Fig. 5: Backend Code

This source code integrates a prepress visual preview functionality to deliver a real-time representation of the document contents to the operator. Furthermore, the system automates chromatic transformation mechanisms via pure color space conversion to guarantee print reproduction fidelity. The software also facilitates absolute geometric document scaling, which encompasses dimension standardization spanning from A3 to A6 variants.

3.4.1. Document Preview

a. Proportional Scale Calculation
The system automatically calculates the scale factor (SF) by comparing the maximum height threshold of the interface component (550 pixels) against the standard height of an A3 document (1191 points). This reference value is subsequently multiplied by the target paper dimensions selected by the user to produce mathematically precise width (*display_w*) and height (*display_h*) projections.

b. Widget Geometry Transformation
The visual component *(self.label_visual)* is dynamically reconfigured using the *.configure()* function to adjust the physical dimensions of the widget. This component is then positioned precisely at the center of the black control frame using relative coordinate parameters via *.place(relx=0.5, rely=0.5, anchor="center")*.

c. Raster Image Extraction
The active PDF page is extracted into a lightweight pixmap object through translative matrix multiplication between the widget size and the original dimensions of the page (*page.rect*). Utilizing the Pillow library, this binary object is converted into an *ImageTk.PhotoImage* format to be rendered on the application interface in real-time.

3.4.2. Size Conversion

a. Dimensional Parameter Standardization
The system provides a dictionary-based data repository (*self.paper_sizes*) that stores the absolute values of A3, A4, A5, and A6 paper sizes in DTP points (*1 inch = 72 points*). This constant serves as the primary reference for altering the document's geometric scale.

b. Target Document Initialization
Upon execution of the export function, the system instantiates a new, blank PDF document (*final_doc = fitz.open()*) to serve as the reconstruction container for the resized document.

c. Page Geometry Reconstruction
Through a looping process, the system generates new pages within the target file with absolutely defined width and height dimensions (*new_page*). The image content from the original page is then inserted to fit the boundaries of this new canvas using the *insert_image()* method.

3.4.3. Color Conversion

a. Target Color Space Mapping
During the file reconstruction process, the system reads the string variable from the color dropdown menu (*target_color*) to determine the target chromatic scheme to be applied to the printed document.

b. Chromatic Transformation & Compression
The application extracts page data using the pure color space *fitz.csCMYK* (or *fitz.csRGB* depending on the selection). The final file with the transformed color space is then saved and compressed using the deflate=True parameter to minimize storage file size without compromising print quality.

4. Testing Result

4.1. Comparative Analysis with Adobe Photoshop

Comparative evaluation demonstrates that the visual fidelity of the application's output closely approaches the industry-standard quality produced by Adobe Photoshop. Meanwhile, validation testing of the document's geometric dimensional conversion achieved an absolute precision rate of 100%, matching the benchmarking software identically.

Table 1: Comparison with Adobe Photoshop

Document Sample	Parameter	Before	After	
			Prepress Application	Adobe Photoshop
 Sample 1	Color Mode	RGB	CMYK	CMYK
	Size	A4 (21 29,7 cm)	A5 (14,8 x 20,9 cm)	A5 (14,8 x 21 cm)
 Sample 2	Color Mode	RGB	CMYK	CMYK
	Size	A3 (29,7 x 42 cm)	A4 (20,9 x 29,6 cm)	A4 (21 x 29,7 cm)
 Sample 3	Color Mode	RGB	CMYK	CMYK
	Size	A4 (21 29,7 cm)	A3 (29,7 x 42 cm)	A3 (29,7 x 42 cm)

This achievement provides operators with a guarantee of reliability in maintaining the exact layout of prepress elements without visual distortion.

4.2. Computational Performance

Table 2: Computational Performance Result

Document No.	Page Count	Processing Time
Document 1	1	< 1 sec
Document 2	3	1-2 sec
Document 3	16	1-2 sec

Computational performance analysis indicates that the system maintains highly stable data processing efficiency, with execution times hovering at an average of 1 second per document, regardless of the variations in page counts tested. This constant processing speed significantly reduces the technical latency that typically bottlenecks traditional prepress production workflows. This responsive execution latency ultimately delivers a seamless, less stressful operational experience for operators within fast-paced industrial environments.

5. Conclusion

This study successfully developed a Python-based prepress application using PyMuPDF and CustomTkinter, streamlining the workflow for small to medium-scale printing industries . The testing results verified that the system achieves absolute precision (100% accuracy) in geometric document size conversions spanning from A3 to A6 variants while producing print-ready CMYK files that closely match industry-standard benchmarking software like Adobe Photoshop . Furthermore, the application demonstrates highly stable computational efficiency with a rapid execution speed averaging just 1 to 2 seconds per document, effectively mitigating human error and eliminating technical latency bottlenecking traditional prepress workflows . Ultimately, this standalone software provides an accessible, intuitive, and budget-friendly alternative that significantly enhances operational production time efficiency and operator comfort in fast-paced industrial environments.

References

[1] S. Dedijer *et al.*, “Reshaping The Future Of Prepress,” pp. 926–933, 2024.
[2] Y. P. Pratama, R. N. Kartika, P. F. Buwono, and M. A. Rabbani, “Analisis Deteksi Tepi Pada Pemindai Dokumen Menggunakan Kertas Thermal Berbasis Python Ide,” *JITET (Jurnal Inform. dan Tek. Elektro Ter.*, vol. 12, no. 3, pp. 2819–2824, 2024.
[3] F. F. Fajar, M. A. Yaqin, S. S. Nuraini, Trisna, and Y. Prastyo, “Analysis of Layout Design Discrepancies Generated in the Proofreading Process for Product Quality Improvement in the Label & Packaging Manufacturing Industry : Evidence from XYZ Company,” vol. 02, no. 12, pp. 526–542, 2025.
[4] B. Gustiandi, *Langkah Awal Menguasai Bahasa Pemrograman Python*. BRIN, 2023.
[5] E. K. Amissah, K. Nduro, and E. D. Kudjordjie, “Optimizing Workflow Efficiency in Digital Printing Imposition : A Visual Tutorial on Design Impact Printing Methods,” vol. 6, no. 2, pp. 1–28, 2024.
[6] N. S. Adhikari and S. Agarwal, “A Comparative Study of PDF Parsing Tools Across Diverse Document Categories,” 2025.
[7] S. Nurzhan, “Development of a linguistic support model for information retrieval for cloud library systems,” 2024.
[8] H. Seetha, V. Tiwari, K. R. Anugu, S. Makka, and R. Karnati, “A GUI Based Application for PDF Processing Tools Using Python & CustomTkinter,” no. January, 2023.
[9] O. O. Emmanuel, M. Z. Dorcas, O. F. Amrevuawho, A. P. Eleojo, P. O. Olawoye, and I. O. Joshua, “A Systematic Review of Python Libraries for Modern UI Development,” vol. 7, pp. 1745–1751, 2025.