# What Is JMuPdf?

JMuPdf is a java library for rendering PDF, XPS and CBZ (Comic Book) documents. It can render pages and export them into various formats as well as generate a java buffered image. It has the ability to render the whole page or sections (cropping). It can also extract page text, links, outlines, etc.

## Supported Color Spaces

1. ARGB, ARGB_PRE, RGB, BGR, Gray-scale, and Binary (Black & White)
2. Optional dithering available for binary images

## Supported Export Formats

- JMuPdf can convert rendered pages to TIF, Multi-Page TIF, JPG, BMP, PNG, PNM, PAM, and PBM files.
- Supported TIF Compressions
    - CCITT, LZW, Deflate, JPeg, ZLib, Packbits, and Deflate are all supported.

## Java Buffered Images

The toolkit can also create java buffered images of pages so that the developer can further extend the libraries functionality. The following types are supported: RGB, ARGB, ARGB_PRE, BGR, GRAY, and BINARY.

## Improving Image Quality

There are settings that can be set to help improve quality such as anti-aliasing levels and gamma correction.

## Supported Operating Systems

1. Windows x86 and x64
2. Linux x86 and x64

## How It Works

JMuPdf provides native binaries for Windows and Linux environments. The native binaries are based on the MuPDF rendering library from Artifex Software, Inc. MuPDF is a lightweight toolkit written in portable C.
Basically JMuPdf exposes MuPDF's functionality via JNI wrappers. JMuPdf is designed with the following in mind: robustness, speed, and efficiency. There has been a great deal of thought and bench-marking in considering how to implement the JNI code.

## Why Use JNI?

First, I don't like reinventing the wheel. Why try to rewrite something that is already fantastic. It's like trying to improve on perfection. It's just a waste of time.
Second, imagine trying to rewrite the whole thing in Java. It probably could be done but then it would be a maintenance nightmare trying to keep new features in sync.
Third, there is a lot of hype about whether or not it is a good idea to code JNI. I've read many articles where programmers get all hung up when they realize they have a DLL or SO dependency. These guys are "purists" but what they need to realize is that the JVM is riddled with JNI code and have many O/S binaries it depends on. Actually the core of Java depends on it. So much for that. :-)
Fourth, it's said that "crossing over" from Java to JNI has an impact on performance. I am sorry but that's an "old wives tale". It might have been true say 10 years ago, but not today. Programming techniques and processing power have come a long way since the advent of index cards. There is very little impact on Java-to-JNI invocations. So little that it is barely measurable especially if you make use of the proper JNI functions and cleanup after yourself.
Ultimately if you write bad code it doesn't matter what language you write it in, it's just not going to perform (period) :-)