

Java



Perform OCR



HOME &gt; GUIDES &gt; JAVA &gt; OCR &gt; PERFORM OCR

# Perform OCR on PDFs in Java



COPY PAGE



This guide demonstrates how to perform optical character recognition (OCR) with Nutrient Java SDK to extract text from PDFs.

## Unlock the power of OCR for document processing

Nutrient Java SDK's OCR component enables seamless text recognition, making scanned documents and image-based PDFs fully interactive. By converting inaccessible text into searchable and selectable content, OCR enhances document usability, supports automated data extraction, and integrates smoothly with other PDF processing features. You can optimize workflows by customizing OCR parameters, such as processing specific page ranges or selecting recognition languages.

For more in-depth understanding of OCR and its applications, refer to the Java OCR SDK guide on [extracting text from PDFs](#).

## Enabling OCR in your Java application

To use OCR functionality in Nutrient Java SDK, include an additional package dependency separate from the core SDK. If your project is already configured with the [Nutrient repository](#), adding OCR support requires a dependency update. For projects that cannot



ASK AI

use Maven, manual integration is also available through the [Nutrient Portal](#). Proper integration ensures seamless access to advanced text recognition and document processing features.

For more information, refer to the guide on [how to integrate OCR SDK into a Java application](#).

## Performing OCR on a PDF

The `OcrProcessor` class handles OCR processing in Nutrient Java SDK as shown below:

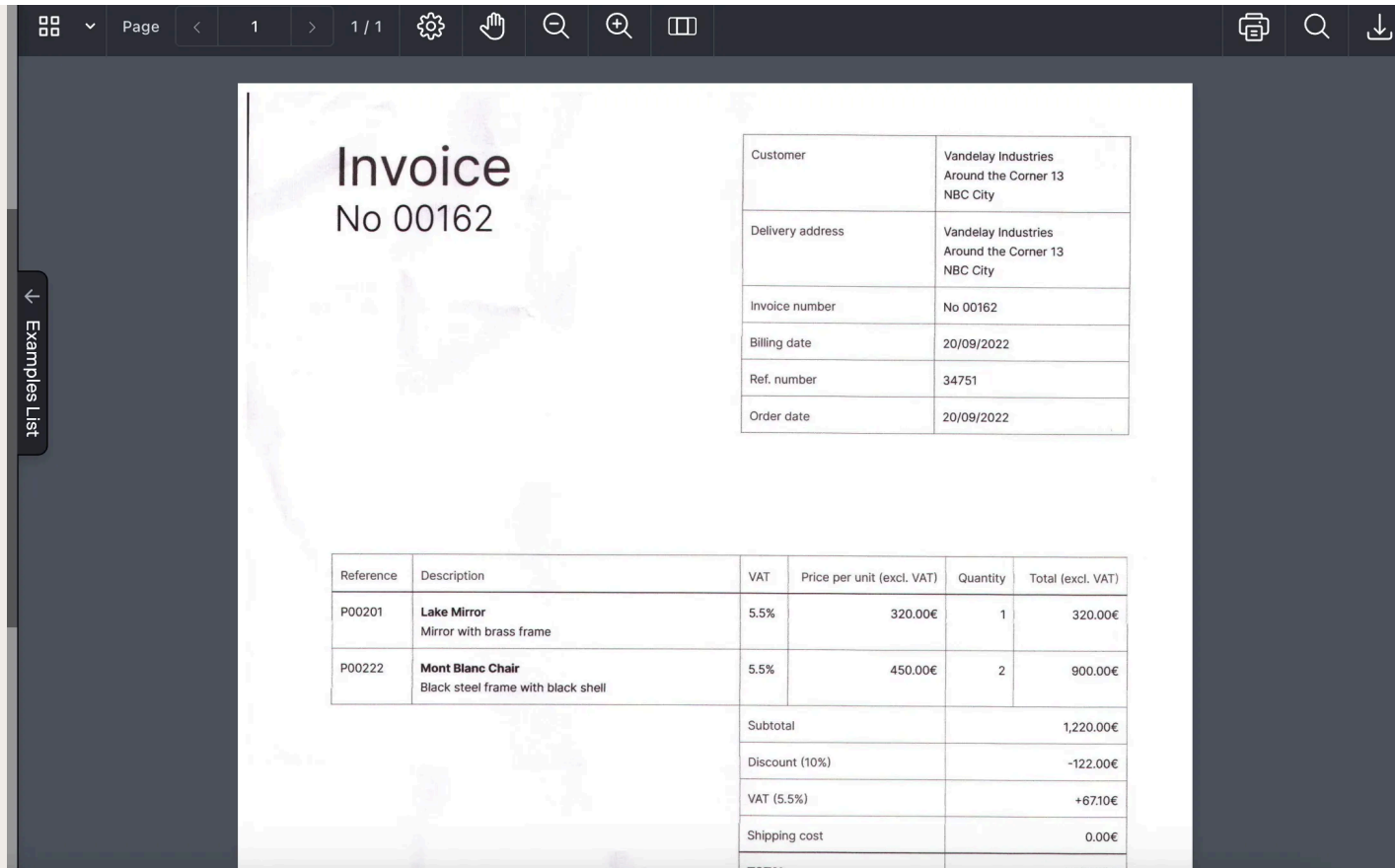
```
// Specify the file to perform OCR on and open it.
File file = new File("ocrExample.pdf");
PdfDocument pdfDocument = PdfDocument.open(new FileDataProvider(file));

// Create a file to write the new document that will contain the OCR data extracted
File outputFile = File.createTempFile("ocrOutput", ".pdf");
FileDataProvider outputDataProvider = new FileDataProvider(outputFile);

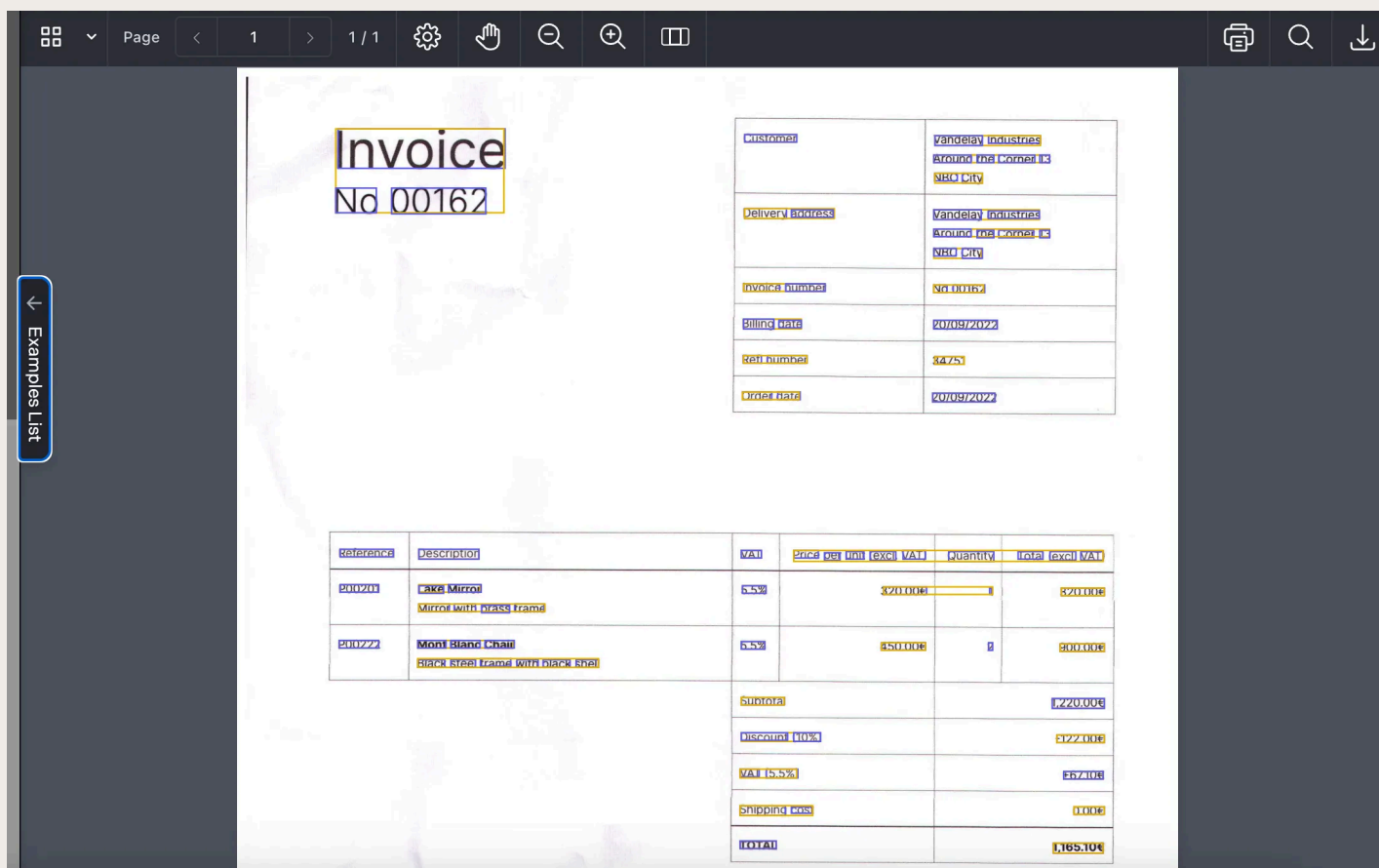
// Create the processor with the open document using the processor `Builder` and
OcrProcessor ocrProcessor = new OcrProcessor.Builder(pdfDocument).build();
ocrProcessor.performOcr(outputDataProvider);
```

The code above processes the `ocrExample.pdf` file and saves the OCR-enhanced output to `ocrOutput.pdf`. The resulting document converts previously unsearchable, image-based text (such as scanned documents) into selectable and annotatable text.

### PDF before performing OCR



## PDF after performing OCR



## Limiting OCR to specific pages

To improve efficiency, limit OCR processing to specific pages, as shown below:

```
// Create the processor with the open document and perform OCR only on page `0`.
Set<Integer> pages = new HashSet<>();
pages.add(0);
OcrProcessor ocrProcessor = new OcrProcessor.Builder(pdfDocument)
    .setPages(pages)
    .build();
ocrProcessor.performOcr(outputDataProvider);
```

The approach above significantly reduces processing time for large documents by applying OCR to necessary pages only.

## Specifying the OCR language

For accurate text recognition, set the document's language in `OcrProcessor`. By default, Nutrient Java SDK uses English, but you can specify other languages as needed:

```
// Create the processor with the open document and perform OCR using the Finnish
OcrProcessor ocrProcessor = new OcrProcessor.Builder(pdfDocument)
    .setLanguage(OcrLanguage.Finnish)
    .build();
ocrProcessor.performOcr(outputDataProvider);
```

For a complete list of supported languages, refer to the [OCR language support](#) guide. Customizing the OCR language enhances accuracy for multilingual documents.

---

### Was this helpful?

✓ YES

✗ NO

---