

[DOCS](#)[CONTACT SALES](#)[Processor](#)[Get Started](#)[GETTING STARTED](#)[PROCESSOR](#)

# Getting started with Processor

[GOLANG](#)

PSPDFKit Processor has been deprecated and replaced by [Document Engine](#). To migrate to Document Engine and unlock advanced document processing capabilities, refer to our [migration guide](#). Learn more about these enhancements on our [blog](#).

This guide walks you through the steps necessary to start PSPDFKit Processor. It also shows you how to use it to process documents. By the end, you'll be able to merge two PDF documents into one using Processor's HTTP API with Golang.

## Requirements

PSPDFKit Processor runs on a variety of platforms. The following operating systems are supported:

macOS Ventura, Monterey, Mojave, Catalina, or Big Sur

Windows 10 Pro, Home, Education, or Enterprise 64-bit

[ASK AI](#)

Ubuntu, Fedora, Debian, or CentOS. Ubuntu and Debian derivatives such as Kubuntu or Xubuntu are supported as well. Currently only 64-bit Intel (x86\_64) processors are supported.

Regardless of your operating system, you'll need at least 4 GB of RAM.

## 1 Installing Docker

PSPDFKit Processor is distributed as a Docker container. To run it on your computer, you need to install a Docker runtime distribution for your operating system.

MACOS

WINDOWS

LINUX

Install and start Docker Desktop for Mac. Refer to the Docker website for instructions.

## 2 Starting PSPDFKit Processor

First, open your terminal emulator.

MACOS

WINDOWS

LINUX

Use the terminal emulator integrated with your code editor or IDE. Alternatively, you can use `Terminal.app` or `iTerm2`.

Now run the following command:

```
docker run --rm -t -p 5000:5000 pspdfkit/processor:2023.11.1
```

This command might take a while to run, depending on your internet connection speed. Wait until you see a message like this in the terminal:

[info] 2023-02-05 18:56:45.286 Running PSPDFKit Processor version 2023

The PSPDFKit Processor is now up and running!

### 3 Installing Golang

The interaction with PSPDFKit Processor happens via its HTTP API. Documents and commands are sent in API request calls, and the resulting files are received in API response calls. API calls are invoked from the Go package, so you need to install Golang.

To install Golang:

- 1 Follow the instructions for your operating system on Golang's Download and Installation Page.
- 2 Go to any directory in your system using your terminal. Create a new directory called `merging-pdfs` and go to the newly created directory:

```
1 mkdir merging-pdfs
2 cd merging-pdfs
```

- 3 Create a new `go` module by running the command below from the `merging-pdfs` directory. Replace `YOUR-GITHUB-USERNAME` with your actual GitHub username:

```
go mod init github.com/YOUR-GITHUB-USERNAME/merging-pdfs
```

- 4 Create a new file in the directory called `merge.go` and add the following content:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7
```

```
fmt.Println("Hello World")
```

- 5 Run the file from your terminal with `go run merge.go` to make sure everything is working properly.

## 4 Merging PDFs with Golang

If you don't have any sample documents, download and use these files: `cover.pdf` and `document.pdf`

Replace the contents of the `merge.go` file with the code below. Replace `/path/to/cover.pdf` in lines 37 and 46, and replace `/path/to/document.pdf` in lines 59 and 68 with the actual paths to the example documents on your machine:

```
1 package main
2
3 import (
4     "bytes"
5     "fmt"
6     "io"
7     "log"
8     "mime/multipart"
9     "net/http"
10    "os"
11    "path/filepath"
12 )
13
14 func main() {
15     instructions := `
16     {
17         "parts": [
18             {
19                 "file": "cover"
20             },
21             {
22                 "file": "document"
23             }
24         ]
25     }
26 `
27
28     payload := &bytes.Buffer{}
29     writer := multipart.NewWriter(payload)
30
31     err := writer.WriteField("instructions", instructions)
32     if err != nil {
```

```
33     log.Fatalln(err)
34 }
35
36 // Replace "/path/to/cover.pdf".
37 cover, err := os.Open("/path/to/cover.pdf")
38 defer cover.Close()
39
40 if err != nil {
41     log.Fatalln(err)
42     return
43 }
44
45 // Replace "/path/to/cover.pdf".
46 coverPart, err := writer.CreateFormFile("cover", filepath.Base("/pat
47 if err != nil {
48     log.Fatalln(err)
49     return
50 }
51
52 _, err = io.Copy(coverPart, cover)
53 if err != nil {
54     log.Fatalln(err)
55     return
56 }
57
58 // Replace "/path/to/document.pdf".
59 document, err := os.Open("/path/to/document.pdf")
60 defer document.Close()
61
62 if err != nil {
63     log.Fatalln(err)
64     return
65 }
66
67 // Replace "/path/to/document.pdf".
68 documentPart, err := writer.CreateFormFile("document", filepath.Base
69 if err != nil {
70     log.Fatalln(err)
71     return
72 }
73
74 _, err = io.Copy(documentPart, document)
75 if err != nil {
76     log.Fatalln(err)
77     return
78 }
79
80 err = writer.Close()
81 if err != nil {
82     fmt.Println(err)
83     return
84 }
85
```

```
86 processorUrl := "http://localhost:5000/build"
87 method := "POST"
88
89 client := &http.Client{}
90 req, err := http.NewRequest(method, processorUrl, payload)
91
92 if err != nil {
93     fmt.Println(err)
94     return
95 }
96 req.Header.Set("Content-Type", writer.FormDataContentType())
97 res, err := client.Do(req)
98 if err != nil {
99     fmt.Println(err)
100     return
101 }
102 defer res.Body.Close()
103
104 output, err := os.Create("result.pdf")
105
106 log.Print(res)
107 output.ReadFrom(res.Body)
108
109 if err != nil {
110     fmt.Println(err)
111     return
112 }
113 }
```

To run the code, ensure you're in the `merging-pdfs` directory and type the following command in your terminal:

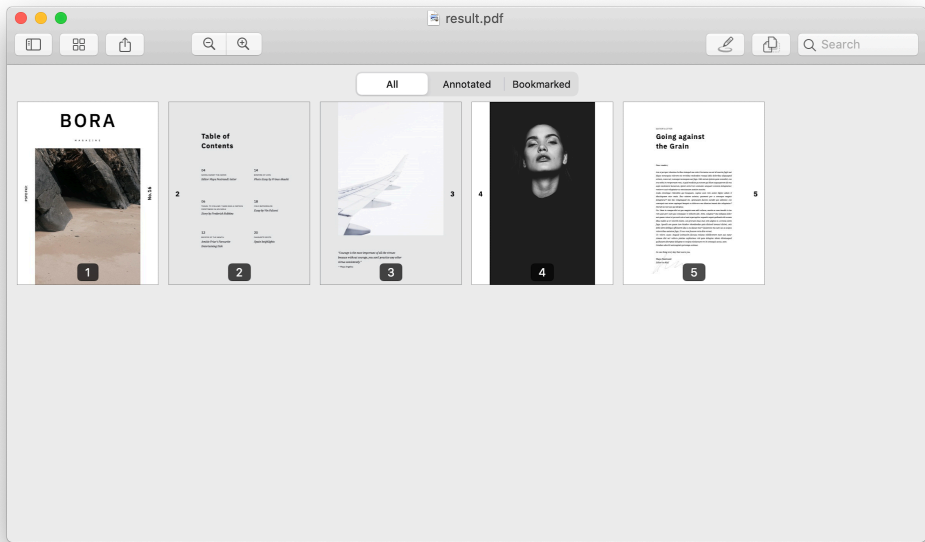
```
go run merge.go
```

Most of this code, up until the `client.Do(req)` statement, constructs a multipart request that's sent to Processor. It includes two files — in this case, `cover` and `document` — and a list of instructions for PSPDFKit Processor.

By default, PSPDFKit Processor's output (the `/build` endpoint) is the result of merging all input documents or parts of the instructions.

To learn more about the `/build` instructions, go to the [Processor API Reference](#) article.

The result of this code is a merged `result.pdf` file in the `merging-pdfs` directory.



---

Was this helpful?

YES

NO

---

Questions? [Contact us](#)