



 SebastiaanKlippert / go-wkhtmltopdf Public

Notifications

Fork 125









Star 896

<>

- Code
- Issues 6
- Pull requests
- Actions
- Projects
- Security
- Insights

 master ·

 152 commits

-  [.github/workflows](#)
[use only stable go version for actions](#)
February 11, 2023 13:26
-  [testdata](#)
[add options with units \(#107\)](#)
February 11, 2023 13:21
-  [.gitignore](#)
[rewrite path check to lock only once](#)
November 5, 2018 17:44
-  [LICENSE](#)
[Initial commit](#)
November 21, 2015 21:44
-  [README.md](#)
[Document changes in relative path discovery in...](#)
November 22, 2022 21:55
-  [go.mod](#)
[Use of exec.ErrDot requires Go 1.19+ \(#104\)](#)
December 8, 2022 09:16
-  [go.sum](#)
[update dependencies](#)
May 31, 2022 21:23
-  [json.go](#)

[update comment](#)

March 11, 2020 10:11

[json_test.go](#)[add options with units \(#107\)](#)

February 11, 2023 13:21

[options.go](#)[add options with units \(#107\)](#)

February 11, 2023 13:21

[release_test.go](#)[added new options for 0.12.6 and some fixes](#)

July 23, 2020 23:15

[samplesample_test..](#)[Updating "Zoom" parameter in sample](#)

November 28, 2018 23:16

[wkhtmltopdf.go](#)[add options with units \(#107\)](#)

February 11, 2023 13:21

[wkhtmltopdf_test....](#)[write the PDF for the unit test](#)

February 11, 2023 13:40



README.md

 [reference](#)  [go report](#)  [codebeat](#)  [codecov](#)  93% [Ubuntu build & test](#)   [macOS build & test](#)  passing

go-wkhtmltopdf

Golang commandline wrapper for wkhtmltopdf

See <http://wkhtmltopdf.org/index.html> for wkhtmltopdf docs.

What and why

We needed a way to generate PDF documents from Go. These vary from invoices with highly customizable lay-outs to reports with tables, graphs and images. In our opinion the best way to do this was by using HTML/CSS templates as source for our PDFs. Using CSS print media types and millimeters instead of pixel units we can generate very accurate PDF documents using wkhtmltopdf.

go-wkhtmltopdf is a pure Golang wrapper around the wkhtmltopdf command line utility.

It has all options typed out as struct members which makes it very easy to use if you use an IDE with code completion and it has type safety for all options. For example you can set general options like

```
pdfg.Dpi.Set(600)
pdfg.NoCollate.Set(false)
pdfg.PageSize.Set(PageSizeA4)
pdfg.MarginBottom.Set(40)
```

The same goes for adding pages, settings page options, TOC options per page etc.

It takes care of setting the correct order of options as these can become very long with multiple pages where you have page and TOC options for each page.

Secondly it makes usage in server-type applications easier, every instance (PDF process) has its own output buffer which contains the PDF output and you can feed one input document from an `io.Reader` (using `stdin` in `wkhtmltopdf`). You can combine any number of external HTML documents (HTTP(S) links) with at most one HTML document from `stdin` and set options for each input document.

Note: You can also ignore the internal buffer and let `wkhtmltopdf` write directly to disk if required for large files, or use the [SetOutput](#) method to pass any `io.Writer`.

For us this is one of the easiest ways to generate PDF documents from Go(lang) and performance is very acceptable.

Installation

go get or use a Go dependency manager of your liking.

```
go get -u github.com/SebastiaanKlippert/go-wkhtmltopdf
```

`go-wkhtmltopdf` finds the path to `wkhtmltopdf` by

- first looking in the current dir
- looking in the `PATH` and `PATHEXT` environment dirs
- using the `WKHTMLTOPDF_PATH` environment dir

Warning: Running executables from the current path is no longer possible in Go 1.19, see https://pkg.go.dev/os/exec@master#hdr-Executables_in_the_current_directory

If you need to set your own `wkhtmltopdf` path or want to change it during execution, you can call `SetPath()`.

Usage

See testfile `wkhtmltopdf_test.go` for more complex options, a common use case test is in `simplesample_test.go`

```
package wkhtmltopdf

import (
    "fmt"
    "log"
)

func ExampleNewPDFGenerator() {

    // Create new PDF generator
    pdfg, err := NewPDFGenerator()
    if err != nil {
        log.Fatal(err)
    }

    // Set global options
    pdfg.Dpi.Set(300)
    pdfg.Orientation.Set(OrientationLandscape)
    pdfg.Grayscale.Set(true)

    // Create a new input page from an URL
    page := NewPage("https://godoc.org/github.com/SebastiaanKlippert/go-wkhtmltopdf")

    // Set options for this page
    page.FooterRight.Set("[page]")
    page.FooterFontSize.Set(10)
    page.Zoom.Set(0.95)

    // Add to document
    pdfg.AddPage(page)

    // Create PDF document in internal buffer
    err = pdfg.Create()
    if err != nil {
        log.Fatal(err)
    }

    // Write buffer contents to file on disk
    err = pdfg.WriteFile("./simplesample.pdf")
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Done")
    // Output: Done
}
```

As mentioned before, you can provide one document from stdin, this is done by using a [PageReader](#) object as input to `AddPage`. This is best constructed with [NewPageReader](#) and will accept any `io.Reader` so this can be used with files from disk (`os.File`) or memory (`bytes.Buffer`) etc.

A simple example snippet:

```
html := "<html>Hi</html>"
pdfgen.AddPage(NewPageReader(strings.NewReader(html)))
```

Saving to and loading from JSON

The package now has the possibility to save the PDF Generator object as JSON and to create a new PDF Generator from a JSON file. All options and pages are saved in JSON, pages added using `NewPageReader` are read to memory before saving and then saved as Base64 encoded strings in the JSON file.

This is useful to prepare a PDF file and generate the actual PDF elsewhere, for example on AWS Lambda. To create PDF Generator on the client, where wkhtmltopdf might not be present, function `NewPDFPreparer` can be used.

Use `NewPDFPreparer` to create a PDF Generator object on the client and `NewPDFGeneratorFromJSON` to reconstruct it on the server.

```
// Client code
pdfg := NewPDFPreparer()
htmlfile, err := ioutil.ReadFile("testdata/htmlsimple.html")
if err != nil {
    log.Fatal(err)
}

pdfg.AddPage(NewPageReader(bytes.NewReader(htmlfile)))
pdfg.Dpi.Set(600)

// The contents of htmlsimple.html are saved as base64 string in the JSON file
jb, err := pdfg.ToJSON()
if err != nil {
    log.Fatal(err)
}

// Server code
pdfgFromJSON, err := NewPDFGeneratorFromJSON(bytes.NewReader(jb))
if err != nil {
    log.Fatal(err)
}

err = pdfgFromJSON.Create()
if err != nil {
    log.Fatal(err)
}
```

For an example of running this in AWS Lambda see <https://github.com/SebastiaanKlippert/go-wkhtmltopdf-lambda>

Speed

The speed is pretty much determined by wkhtmltopdf itself, or if you use external source URLs, the time it takes to get and render the source HTML.

The go wrapper time is negligible with around 0.04ms for parsing an above average number of commandline options.

Benchmarks are included.

About

Golang commandline wrapper for wkhtmltopdf

[go](#) [html](#) [golang](#) [pdf](#) [wkhtmltopdf](#) [pdf-document](#) [html-to-pdf](#) [pdf-generation](#)

[Readme](#)


[MIT license](#)

[896 stars](#)

[10 watching](#)

[125 forks](#)









Releases 17

 v1.9.0 Latest
Feb 11, 2023
[+ 16 releases](#)

Packages

No packages published

Used by 421



+ 413

Contributors 10



Languages

 Go 100.0%
 © 2023 GitHub,
Inc.
[Terms](#)
[Privacy](#)
[Security](#)
[Status](#)
[Docs](#)

[Contact GitHub](#)

[Pricing](#)

[API](#)

[Training](#)

[Blog](#)

[About](#)