Discover Packages >
github.com/SebastiaanKlippert/go-wkhtmltopdf

wkhtmltopdf

Version: v1.7.2 latest | Published: Sep 26, 2021 | License: MIT | Imports: 14 | Imported by: 45

package module

Details
Valid go.mod file
Redistributable license
Tagged version
Stable version
Learn more

Repository
github.com/SebastiaanKlippert/go-wkhtmltopdf

Links
Report a Vulnerability
Open Source Insights

README

README

reference | go report A | codebeat B | codecov 90%

Ubuntu build & test passing | 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 acurate 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 muliple 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

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 if 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.

<> Documentation

Overview

Package wkhtmltopdf contains wrappers around the wkhtmltopdf commandline tool

Index

Examples

NewPDFGenerator
NewPDFGeneratorFromJSON

Constants

View Source

```
const (
 OrientationLandscape = "Landscape" // Landscape mode
 OrientationPortrait  = "Portrait"  // Portrait mode
)
```

Constants for orientation modes

View Source

```
const (
 PageSizeA0        = "A0"        // 841 x 1189 mm
 PageSizeA1        = "A1"        // 594 x 841 mm
 PageSizeA2        = "A2"        // 420 x 594 mm
 PageSizeA3        = "A3"        // 297 x 420 mm
 PageSizeA4        = "A4"        // 210 x 297 mm, 8.26
 PageSizeA5        = "A5"        // 148 x 210 mm
 PageSizeA6        = "A6"        // 105 x 148 mm
 PageSizeA7        = "A7"        // 74 x 105 mm
 PageSizeA8        = "A8"        // 52 x 74 mm
 PageSizeA9        = "A9"        // 37 x 52 mm
 PageSizeB0        = "B0"        // 1000 x 1414 mm
 PageSizeB1        = "B1"        // 707 x 1000 mm
 PageSizeB10       = "B10"       // 31 x 44 mm
 PageSizeB2        = "B2"        // 500 x 707 mm
 PageSizeB3        = "B3"        // 353 x 500 mm
 PageSizeB4        = "B4"        // 250 x 353 mm
 PageSizeB5        = "B5"        // 176 x 250 mm, 6.93
 PageSizeB6        = "B6"        // 125 x 176 mm
 PageSizeB7        = "B7"        // 88 x 125 mm
 PageSizeB8        = "B8"        // 62 x 88 mm
 PageSizeB9        = "B9"        // 33 x 62 mm
 PageSizeC5E       = "C5E"       // 163 x 229 mm
 PageSizeComm10E   = "Comm10E"   // 105 x 241 mm, U.S. Common 10 Envelope
 PageSizeCustom    = "Custom"    // Unknown, or a user defined size.
 PageSizeDLE       = "DLE"       // 110 x 220 mm
 PageSizeExecutive = "Executive" // 7.5 x 10 inches, 190.5 x 254 mm
 PageSizeFolio     = "Folio"     // 210 x 330 mm
 PageSizeLedger    = "Ledger"    // 431.8 x 279.4 mm
 PageSizeLegal     = "Legal"     // 8.5 x 14 inches, 215.9 x 355.6 mm
 PageSizeLetter    = "Letter"    // 8.5 x 11 inches, 215.9 x 279.4 mm
 PageSizeTabloid   = "Tabloid"   // 279.4 x 431.8 mm
)
```

Constants for page sizes

Variables

This section is empty.

Functions

func GetPath

```
func GetPath() string
```

GetPath gets the path to wkhtmltopdf

func SetPath

```
func SetPath(path string)
```

SetPath sets the path to wkhtmltopdf

Types

type PDFGenerator

```
type PDFGenerator struct {
 Cover      cover
 TOC        toc
 OutputFile string //filename to write to, default empty (writes to internal buffer)
 // contains filtered or unexported fields
}
```

PDFGenerator is the main wkhtmltopdf struct, always use NewPDFGenerator to obtain a new PDFGenerator struct

func NewPDFGenerator

```
func NewPDFGenerator() (*PDFGenerator, error)
```

NewPDFGenerator returns a new PDFGenerator struct with all options created and checks if wkhtmltopdf can be found on the system

▶ Example

func NewPDFGeneratorFromJSON added in v1.2.0

```
func NewPDFGeneratorFromJSON(jsonReader io.Reader) (*PDFGenerator, error)
```

NewPDFGeneratorFromJSON creates a new PDFGenerator and restores all the settings and pages from a JSON byte slice which should be created using PDFGenerator.ToJSON().

▶ Example

func NewPDFPreparer added in v1.2.0

```
func NewPDFPreparer() *PDFGenerator
```

NewPDFPreparer returns a PDFGenerator object without looking for the wkhtmltopdf executable file. This is useful to prepare a PDF file that is generated elsewhere and you just want to save the config as JSON. Note that Create() can not be called on this object unless you call SetPath yourself.

func (*PDFGenerator) AddPage

```
func (pdfg *PDFGenerator) AddPage(p PageProvider)
```

AddPage adds a new input page to the document. A page is an input HTML page, it can span multiple pages in the output document. It is a Page when read from file or URL or a PageReader when read from memory.

func (*PDFGenerator) ArgString

```
func (pdfg *PDFGenerator) ArgString() string
```

ArgString returns Args as a single string

func (*PDFGenerator) Args

```
func (pdfg *PDFGenerator) Args() []string
```

Args returns the commandline arguments as a string slice

func (*PDFGenerator) Buffer

```
func (pdfg *PDFGenerator) Buffer() *bytes.Buffer
```

Buffer returns the embedded output buffer used if OutputFile is empty

func (*PDFGenerator) Bytes

```
func (pdfg *PDFGenerator) Bytes() []byte
```

Bytes returns the output byte slice from the output buffer used if OutputFile is empty

func (*PDFGenerator) Create

```
func (pdfg *PDFGenerator) Create() error
```

Create creates the PDF document and stores it in the internal buffer if no error is returned

func (*PDFGenerator) CreateContext added in v1.7.0

```
func (pdfg *PDFGenerator) CreateContext(ctx context.Context) error
```

CreateContext is Create with a context passed to exec.CommandContext when calling wkhtmltopdf

func (*PDFGenerator) ResetPages added in v1.4.1

```
func (pdfg *PDFGenerator) ResetPages()
```

ResetPages drops all pages previously added by AddPage or SetPages. This allows reuse of current instance of PDFGenerator with all of it's configuration preserved.

func (*PDFGenerator) SetOutput added in v1.3.0

```
func (pdfg *PDFGenerator) SetOutput(w io.Writer)
```

SetOutput sets the output to write the PDF to, when this method is called, the internal buffer will not be used, so the Bytes(), Buffer() and WriteFile() methods will not work.

func (*PDFGenerator) SetPages

```
func (pdfg *PDFGenerator) SetPages(p []PageProvider)
```

SetPages resets all pages

func (*PDFGenerator) SetStderr added in v1.5.0

```
func (pdfg *PDFGenerator) SetStderr(w io.Writer)
```

SetStderr sets the output writer for Stderr when running the wkhtmltopdf command. You only need to call this when you want to print the output of wkhtmltopdf (like the progress messages in verbose mode). If not called, or if w is nil, the output of Stderr is kept in an internal buffer and returned as error message if there was an error when calling wkhtmltopdf.

func (*PDFGenerator) ToJSON added in v1.2.0

```
func (pdfg *PDFGenerator) ToJSON() ([]byte, error)
```

ToJSON creates JSON of the complete representation of the PDFGenerator. It also saves all pages. For a PageReader page, the content

is stored as a Base64 string in the JSON.

func (*PDFGenerator) WriteFile

```
func (pdfg *PDFGenerator) WriteFile(filename string) error
```

WriteFile writes the contents of the output buffer to a file

type Page

```
type Page struct {
 Input string
 PageOptions
}
```

Page is the input struct for each page

func NewPage

```
func NewPage(input string) *Page
```

NewPage creates a new input page from a local or web resource (filepath or URL)

func (*Page) Args

```
func (p *Page) Args() []string
```

Args returns the argument slice and is part of the page interface

func (*Page) InputFile

```
func (p *Page) InputFile() string
```

InputFile returns the input string and is part of the page interface

func (*Page) Reader

```
func (p *Page) Reader() io.Reader
```

Reader returns the io.Reader and is part of the page interface

type PageOptions

```
type PageOptions struct {
 // contains filtered or unexported fields
}
```

PageOptions are options for each input page

func NewPageOptions

```
func NewPageOptions() PageOptions
```

NewPageOptions returns a new PageOptions struct with all options

func (*PageOptions) Args

```
func (po *PageOptions) Args() []string
```

Args returns the argument slice

type PageProvider added in v1.6.1

```
type PageProvider interface {
 Args() []string
 InputFile() string
 Reader() io.Reader
}
```

PageProvider is the interface which provides a single input page. Implemented by Page and PageReader.

type PageReader

```
type PageReader struct {
 Input io.Reader
 PageOptions
}
```

PageReader is one input page (a HTML document) that is read from an io.Reader You can add only one Page from a reader

func NewPageReader

```
func NewPageReader(input io.Reader) *PageReader
```

NewPageReader creates a new PageReader from an io.Reader

func (*PageReader) Args

```
func (pr *PageReader) Args() []string
```

Args returns the argument slice and is part of the page interface

func (*PageReader) InputFile

```
func (pr *PageReader) InputFile() string
```

InputFile returns the input string and is part of the page interface

func (*PageReader) Reader

```
func (pr *PageReader) Reader() io.Reader
```

Reader returns the io.Reader and is part of the page interface

Source Files

View all

json.go
options.go
wkhtmltopdf.go