
TITLE

Hello World!
This is an amazing program!

Go
Skip to Main Content
Search packages or symbols

Why Gosubmenu dropdown icon
Learn
Docssubmenu dropdown icon
Packages
Communitysubmenu dropdown icon
Notice The highest tagged major version is v2.
Discover Packages

github.com/jung-kurt/gofpdf

Go
gofpdf
package
module

Version: v1.16.2 Latest
Published: Nov 19, 2019
License: MIT
Imports: 31
Imported by: 945
Details
checked Valid go.mod file
checked Redistributable license
checked Tagged version
checked Stable version
Learn more about best practices
Repository
github.com/jung-kurt/gofpdf
Links
Open Source Insights Logo Open Source Insights
Jump to ...
README ¶
GoFPDF document generator
No Maintenance Intended MIT licensed Report GoDoc

Package gofpdf implements a PDF document generator with high level support for text, drawing and images.

Features
UTF-8 support
Choice of measurement unit, page format and margins

Page header and footer management
Automatic page breaks, line breaks, and text justification
Inclusion of JPEG, PNG, GIF, TIFF and basic path-only SVG images
Colors, gradients and alpha channel transparency
Outline bookmarks
Internal and external links
TrueType, Type1 and encoding support
Page compression
Lines, Bézier curves, arcs, and ellipses
Rotation, scaling, skewing, translation, and mirroring
Clipping
Document protection
Layers
Templates
Barcodes
Charting facility
Import PDFs as templates
gopdf has no dependencies other than the Go standard library. All tests pass on Linux, Mac and Windows platforms.

gopdf supports UTF-8 TrueType fonts and “right-to-left” languages. Note that Chinese, Japanese, and Korean characters may not be included in many general purpose fonts. For these languages, a specialized font (for example, NotoSansSC for simplified Chinese) can be used.

Also, support is provided to automatically translate UTF-8 runes to code page encodings for languages that have fewer than 256 glyphs.

We Are Closed

This repository will not be maintained, at least for some unknown duration. But it is hoped that gopdf has a bright future in the open source world. Due to Go’s promise of compatibility, gopdf should continue to function without modification for a longer time than would be the case with many other languages.

Forks should be based on the last viable commit. Tools such as active-forks can be used to select a fork that looks promising for your needs. If a particular fork looks like it has taken the lead in attracting followers, this README will be updated to point people in that direction.

The efforts of all contributors to this project have been deeply appreciated. Best wishes to all of you.

Installation

To install the package on your system, run

```
go get github.com/jung-kurt/gopdf
```

Later, to receive updates, run

```
go get -u -v github.com/jung-kurt/gopdf/...
```

Quick Start

The following Go code generates a simple PDF file.

```
pdf := gopdf.New("P", "mm", "A4", "")  
pdf.AddPage()  
pdf.SetFont("Arial", "B", 16)  
pdf.Cell(40, 10, "Hello, world")  
err := pdf.OutputFileAndClose("hello.pdf")
```

See the functions in the `fpdf_test.go` file (shown as examples in this documentation) for more advanced PDF examples.

Errors

If an error occurs in an Fpdf method, an internal error field is set. After this occurs, Fpdf method calls typically return without performing any operations and the error state is retained. This error management scheme facilitates PDF generation since individual method calls do not need to be examined for failure; it is generally sufficient to wait until after `Output()` is called. For the same reason, if an error occurs in the calling application during PDF generation, it may be desirable for the application to transfer the error to the Fpdf instance by calling the `SetError()` method or the `SetErrorf()` method. At any time during the life cycle of the Fpdf instance, the error state can be determined with a call to `Ok()` or `Err()`. The error itself can be retrieved with a call to `Error()`.

Conversion Notes

This package is a relatively straightforward translation from the original FPDF library written in PHP (despite the caveat in the introduction to Effective Go). The API names have been retained even though the Go idiom would suggest otherwise (for example, `pdf.GetX()` is used rather than simply `pdf.X()`). The similarity of the two libraries makes the original FPDF website a good source of information. It includes a forum and FAQ.

However, some internal changes have been made. Page content is built up using buffers (of type `bytes.Buffer`) rather than repeated string concatenation. Errors are handled as explained above rather than panicking. Output is generated through an interface of type `io.Writer` or `io.WriteCloser`. A number of the original PHP methods behave differently based on the type of the arguments that are passed to them; in these cases additional methods have been exported to provide similar functionality. Font definition files are produced in JSON rather than PHP.

Example PDFs

A side effect of running `go test ./...` is the production of a number of example PDFs. These can be found in the `gofpdf/pdf` directory after the tests complete.

Please note that these examples run in the context of a test. In order run an example as a standalone application, youâ€™ll need to examine `fpdf_test.go` for some helper routines, for example `exampleFilename()` and `summary()`.

Example PDFs can be compared with reference copies in order to verify that they have been generated as expected. This comparison will be performed if a PDF with the same name as the example PDF is placed in the `gofpdf/pdf/reference` directory and if the third argument to `ComparePDFFiles()` in `internal/example/example.go` is true. (By default it is false.) The routine that summarizes an example will look for this file and, if found, will call `ComparePDFFiles()` to check the example PDF for equality with its reference PDF. If differences exist between the two files they will be printed to standard output and the test will fail. If the reference file is missing, the comparison is considered to succeed. In order to successfully compare two PDFs, the placement of internal resources must be consistent and the internal creation timestamps must be the same. To do this, the methods `SetCatalogSort()` and `SetCreationDate()` need to be called for both files. This is done automatically for all examples.

Nonstandard Fonts

Nothing special is required to use the standard PDF fonts (`courier`, `helvetica`, `times`, `zapfdingbats`) in your documents other than calling `SetFont()`.

You should use `AddUTF8Font()` or `AddUTF8FontFromBytes()` to add a TrueType UTF-8 encoded font. Use `RTL()` and `LTR()` methods switch between â€™right-to-leftâ€™ and â€™left-to-rightâ€™ mode.

In order to use a different non-UTF-8 TrueType or Type1 font, you will need to generate a font definition file

and, if the font will be embedded into PDFs, a compressed version of the font file. This is done by calling the `MakeFont` function or using the included `makefont` command line utility. To create the utility, `cd` into the `makefont` subdirectory and run `go build`. This will produce a standalone executable named `makefont`. Select the appropriate encoding file from the font subdirectory and run the command as in the following example.

```
./makefont --embed --enc=./font/cp1252.map --dst=./font ./font/calligra.ttf
```

In your PDF generation code, call `AddFont()` to load the font and, as with the standard fonts, `SetFont()` to begin using it. Most examples, including the package example, demonstrate this method. Good sources of free, open-source fonts include Google Fonts and DejaVu Fonts.

Related Packages

The `draw2d` package is a two dimensional vector graphics library that can generate output in different forms. It uses `gofpdf` for its document production mode.

Contributing Changes

`gofpdf` is a global community effort and you are invited to make it even better. If you have implemented a new feature or corrected a problem, please consider contributing your change to the project. A contribution that does not directly pertain to the core functionality of `gofpdf` should be placed in its own directory directly beneath the `contrib` directory.

Here are guidelines for making submissions. Your change should

- be compatible with the MIT License

- be properly documented

- be formatted with `go fmt`

- include an example in `fpdf_test.go` if appropriate

- conform to the standards of `golint` and `go vet`, that is, `golint` and `go vet` should not generate any warnings

- not diminish test coverage

Pull requests are the preferred means of accepting your changes.

License

`gofpdf` is released under the MIT License. It is copyrighted by Kurt Jung and the contributors acknowledged below.

Acknowledgments

This package's code and documentation are closely derived from the FPDF library created by Olivier Plathey, and a number of font and image resources are copied directly from it. Bruno Michel has provided valuable assistance with the code. Drawing support is adapted from the FPDF geometric figures script by David Hernandez Sanz. Transparency support is adapted from the FPDF transparency script by Martin Hall-May. Support for gradients and clipping is adapted from FPDF scripts by Andreas Warmser. Support for outline bookmarks is adapted from Olivier Plathey by Manuel Cornes. Layer support is adapted from Olivier Plathey. Support for transformations is adapted from the FPDF transformation script by Moritz Wagner and Andreas Warmser. PDF protection is adapted from the work of Klemen Vodopivec for the FPDF product. Lawrence Kesteloot provided code to allow an image's extent to be determined prior to placement. Support for vertical alignment within a cell was provided by Stefan Schroeder. Ivan Daniluk generalized the font and image loading code to use the Reader interface while maintaining backward compatibility. Anthony Starks provided code for the Polygon function. Robert Lillack provided the Beziergon function and corrected some naming issues with the internal curve function. Claudio Felber provided implementations for dashed line drawing and generalized font loading. Stani Michiels provided support for multi-segment path drawing with smooth line joins, line join styles, enhanced fill modes, and has helped greatly with package presentation and tests. Templating is adapted by Marcus Downing from the FPDF_Tpl library created by Jan Slabon and Setasign. Jelmer Snoeck contributed packages that generate a variety of


barcodes and help with registering images on the web. Jelmer Snoek and Guillermo Pascual augmented the basic HTML functionality with aligned text. Kent Quirk implemented backwards-compatible support for reading DPI from images that support it, and for setting DPI manually and then having it properly taken into account when calculating image size. Paulo Coutinho provided support for static embedded fonts. Dan Meyers added support for embedded JavaScript. David Fish added a generic alias-replacement function to enable, among other things, table of contents functionality. Andy Bakun identified and corrected a problem in which the internal catalogs were not sorted stably. Paul Montag added encoding and decoding functionality for templates, including images that are embedded in templates; this allows templates to be stored independently of gofpdf. Paul also added support for page boxes used in printing PDF documents. Wojciech Matusiak added supported for word spacing. Artem Korotkiy added support of UTF-8 fonts. Dave Barnes added support for imported objects and templates. Brigham Thompson added support for rounded rectangles. Joe Westcott added underline functionality and optimized image storage. Benoit KUGLER contributed support for rectangles with corners of unequal radius, modification times, and for file attachments and annotations.


Roadmap

Remove all legacy code page font support; use UTF-8 exclusively

Improve test coverage as reported by the coverage tool.

Expand $\frac{3}{4}$

Documentation 

Overview 

Features

We Are Closed

Installation

Quick Start

Errors

Conversion Notes

Example PDFs

Nonstandard Fonts

Related Packages

Contributing Changes

License

Acknowledgments

Roadmap

Package gofpdf implements a PDF document generator with high level support for text, drawing and images.

Features 

- UTF-8 support

- Choice of measurement unit, page format and margins

- Page header and footer management

- Automatic page breaks, line breaks, and text justification

- Inclusion of JPEG, PNG, GIF, TIFF and basic path-only SVG images

- Colors, gradients and alpha channel transparency

- Outline bookmarks

- Internal and external links

- TrueType, Type1 and encoding support

- Page compression
- Lines, Bézier curves, arcs, and ellipses
- Rotation, scaling, skewing, translation, and mirroring
- Clipping
- Document protection
- Layers
- Templates
- Barcodes
- Charting facility
- Import PDFs as templates

gofpdf has no dependencies other than the Go standard library. All tests pass on Linux, Mac and Windows platforms.

gofpdf supports UTF-8 TrueType fonts and "right-to-left" languages. Note that Chinese, Japanese, and Korean characters may not be included in many general purpose fonts. For these languages, a specialized font (for example, NotoSansSC for simplified Chinese) can be used.

Also, support is provided to automatically translate UTF-8 runes to code page encodings for languages that have fewer than 256 glyphs.

We Are Closed

This repository will not be maintained, at least for some unknown duration. But it is hoped that gofpdf has a bright future in the open source world. Due to Go's promise of compatibility, gofpdf should continue to function without modification for a longer time than would be the case with many other languages.

Forks should be based on the last viable commit. Tools such as active-forks can be used to select a fork that looks promising for your needs. If a particular fork looks like it has taken the lead in attracting followers, this README will be updated to point people in that direction.

The efforts of all contributors to this project have been deeply appreciated. Best wishes to all of you.

Installation

To install the package on your system, run

```
go get github.com/jung-kurt/gofpdf
```

Later, to receive updates, run

```
go get -u -v github.com/jung-kurt/gofpdf/...
```

Quick Start

The following Go code generates a simple PDF file.

```
pdf := gofpdf.New("P", "mm", "A4", "")
```

```
pdf.AddPage()
pdf.SetFont("Arial", "B", 16)
pdf.Cell(40, 10, "Hello, world")
err := pdf.OutputFileAndClose("hello.pdf")
```

See the functions in the `fpdf_test.go` file (shown as examples in this documentation) for more advanced PDF examples.

Errors ¶

If an error occurs in an `Fpdf` method, an internal error field is set. After this occurs, `Fpdf` method calls typically return without performing any operations and the error state is retained. This error management scheme facilitates PDF generation since individual method calls do not need to be examined for failure; it is generally sufficient to wait until after `Output()` is called. For the same reason, if an error occurs in the calling application during PDF generation, it may be desirable for the application to transfer the error to the `Fpdf` instance by calling the `SetError()` method or the `SetErrorf()` method. At any time during the life cycle of the `Fpdf` instance, the error state can be determined with a call to `Ok()` or `Err()`. The error itself can be retrieved with a call to `Error()`.

Conversion Notes ¶

This package is a relatively straightforward translation from the original `FPDF` library written in PHP (despite the caveat in the introduction to *Effective Go*). The API names have been retained even though the Go idiom would suggest otherwise (for example, `pdf.GetX()` is used rather than simply `pdf.X()`). The similarity of the two libraries makes the original `FPDF` website a good source of information. It includes a forum and FAQ.

However, some internal changes have been made. Page content is built up using buffers (of type `bytes.Buffer`) rather than repeated string concatenation. Errors are handled as explained above rather than panicking. Output is generated through an interface of type `io.Writer` or `io.WriteCloser`. A number of the original PHP methods behave differently based on the type of the arguments that are passed to them; in these cases additional methods have been exported to provide similar functionality. Font definition files are produced in JSON rather than PHP.

Example PDFs ¶

A side effect of running `go test ./...` is the production of a number of example PDFs. These can be found in the `gofpdf/pdf` directory after the tests complete.

Please note that these examples run in the context of a test. In order run an example as a standalone application, youâ€™ll need to examine `fpdf_test.go` for some helper routines, for example `exampleFilename()` and `summary()`.

Example PDFs can be compared with reference copies in order to verify that they have been generated as expected. This comparison will be performed if a PDF with the same name as the example PDF is placed in the `gofpdf/pdf/reference` directory and if the third argument to `ComparePDFFiles()` in `internal/example/example.go` is `true`. (By default it is `false`.) The routine that summarizes an example will look for this file and, if found, will call `ComparePDFFiles()` to check the example PDF for equality with its reference PDF. If differences exist between the two files they will be printed to standard output and the test will fail. If the reference file is missing, the comparison is considered to succeed. In order to successfully compare two PDFs, the placement of internal resources must be consistent and the internal creation timestamps must be the same. To do this, the methods `SetCatalogSort()` and `SetCreationDate()` need to be called for both files. This is done automatically for all examples.

Nonstandard Fonts ¶

Nothing special is required to use the standard PDF fonts (`courier`, `helvetica`, `times`, `zapfdingbats`) in your documents other than calling `SetFont()`.

You should use `AddUTF8Font()` or `AddUTF8FontFromBytes()` to add a TrueType UTF-8 encoded font. Use `RTL()` and `LTR()` methods switch between “right-to-left” and “left-to-right” mode.

In order to use a different non-UTF-8 TrueType or Type1 font, you will need to generate a font definition file and, if the font will be embedded into PDFs, a compressed version of the font file. This is done by calling the `MakeFont` function or using the included `makefont` command line utility. To create the utility, `cd` into the `makefont` subdirectory and run “`go build`”. This will produce a standalone executable named `makefont`. Select the appropriate encoding file from the font subdirectory and run the command as in the following example.

```
./makefont --embed --enc=./font/cp1252.map --dst=./font ../font/calligra.ttf
```

In your PDF generation code, call `AddFont()` to load the font and, as with the standard fonts, `SetFont()` to begin using it. Most examples, including the package example, demonstrate this method. Good sources of free, open-source fonts include Google Fonts and DejaVu Fonts.

Related Packages ¶

The `draw2d` package is a two dimensional vector graphics library that can generate output in different forms. It uses `gofpdf` for its document production mode.

Contributing Changes ¶

`gofpdf` is a global community effort and you are invited to make it even better. If you have implemented a new feature or corrected a problem, please consider contributing your change to the project. A contribution that does not directly pertain to the core functionality of `gofpdf` should be placed in its own directory directly beneath the `contrib` directory.

Here are guidelines for making submissions. Your change should

- be compatible with the MIT License
- be properly documented
- be formatted with `go fmt`
- include an example in `fpdf_test.go` if appropriate
- conform to the standards of `golint` and `go vet`, that is, `golint .` and `go vet .` should not generate any warnings
- not diminish test coverage

Pull requests are the preferred means of accepting your changes.

License ¶

`gofpdf` is released under the MIT License. It is copyrighted by Kurt Jung and the contributors acknowledged below.

Acknowledgments ¶

This package’s code and documentation are closely derived from the FPDF library created by Olivier Plathey, and a number of font and image resources are copied directly from it. Bruno Michel has provided valuable assistance with the code. Drawing support is adapted from the FPDF geometric figures script by David Hernández Sanz. Transparency support is adapted from the FPDF transparency script by Martin Hall-May. Support for gradients and clipping is adapted from FPDF scripts by Andreas Wärmser. Support for outline bookmarks is adapted from Olivier Plathey by Manuel Cornes. Layer support is adapted from Olivier Plathey. Support for transformations is adapted from the FPDF transformation script by Moritz

Wagner and Andreas WÄ¼rmser. PDF protection is adapted from the work of Klemen Vodopivec for the FPDF product. Lawrence Kesteloot provided code to allow an imageâ€™s extent to be determined prior to placement. Support for vertical alignment within a cell was provided by Stefan Schroeder. Ivan Daniluk generalized the font and image loading code to use the Reader interface while maintaining backward compatibility. Anthony Starks provided code for the Polygon function. Robert Lillack provided the Bezierngon function and corrected some naming issues with the internal curve function. Claudio Felber provided implementations for dashed line drawing and generalized font loading. Stani Michiels provided support for multi-segment path drawing with smooth line joins, line join styles, enhanced fill modes, and has helped greatly with package presentation and tests. Templating is adapted by Marcus Downing from the FPDF_Tpl library created by Jan Slabon and Setasign. Jelmer Snoeck contributed packages that generate a variety of barcodes and help with registering images on the web. Jelmer Snoek and Guillermo Pascual augmented the basic HTML functionality with aligned text. Kent Quirk implemented backwards-compatible support for reading DPI from images that support it, and for setting DPI manually and then having it properly taken into account when calculating image size. Paulo Coutinho provided support for static embedded fonts. Dan Meyers added support for embedded JavaScript. David Fish added a generic alias-replacement function to enable, among other things, table of contents functionality. Andy Bakun identified and corrected a problem in which the internal catalogs were not sorted stably. Paul Montag added encoding and decoding functionality for templates, including images that are embedded in templates; this allows templates to be stored independently of gofpdf. Paul also added support for page boxes used in printing PDF documents. Wojciech Matusiak added supported for word spacing. Artem Korotkiy added support of UTF-8 fonts. Dave Barnes added support for imported objects and templates. Brigham Thompson added support for rounded rectangles. Joe Westcott added underline functionality and optimized image storage. Benoit KUGLER contributed support for rectangles with corners of unequal radius, modification times, and for file attachments and annotations.

Roadmap

- Remove all legacy code page font support; use UTF-8 exclusively
- Improve test coverage as reported by the coverage tool.

Example

Index

Constants

```
func CompareBytes(sl1, sl2 []byte, printDiff bool) (err error)
func ComparePDFFiles(file1Str, file2Str string, printDiff bool) (err error)
func ComparePDFs(rdr1, rdr2 io.Reader, printDiff bool) (err error)
func MakeFont(fontFileStr, encodingFileStr, dstDirStr string, msgWriter io.Writer, ...) error
func SetDefaultCatalogSort(flag bool)
func SetDefaultCompression(compress bool)
func SetDefaultCreationDate(tm time.Time)
func SetDefaultModificationDate(tm time.Time)
func TickmarkPrecision(div float64) int
func Tickmarks(min, max float64) (list []float64, precision int)
func UTF8CutFont(inBuf []byte, cutset string) (outBuf []byte)
func UnicodeTranslator(r io.Reader) (f func(string) string, err error)
func UnicodeTranslatorFromFile(fileStr string) (f func(string) string, err error)
type Attachment
type FontDescType
type FontLoader
type Fpdf
func New(orientationStr, unitStr, sizeStr, fontDirStr string) (f *Fpdf)
func NewCustom(init *InitType) (f *Fpdf)
func (f *Fpdf) AddAttachmentAnnotation(a *Attachment, x, y, w, h float64)
func (f *Fpdf) AddFont(familyStr, styleStr, fileStr string)
```

```
func (f *Fpdf) AddFontFromBytes(familyStr, styleStr string, jsonFileBytes, zFileBytes []byte)
func (f *Fpdf) AddFontFromReader(familyStr, styleStr string, r io.Reader)
func (f *Fpdf) AddLayer(name string, visible bool) (layerID int)
func (f *Fpdf) AddLink() int
func (f *Fpdf) AddPage()
func (f *Fpdf) AddPageFormat(orientationStr string, size SizeType)
func (f *Fpdf) AddSpotColor(nameStr string, c, m, y, k byte)
func (f *Fpdf) AddUTF8Font(familyStr, styleStr, fileStr string)
func (f *Fpdf) AddUTF8FontFromBytes(familyStr, styleStr string, utf8Bytes []byte)
func (f *Fpdf) AliasNbPages(aliasStr string)
func (f *Fpdf) Arc(x, y, rx, ry, degRotate, degStart, degEnd float64, styleStr string)
func (f *Fpdf) ArcTo(x, y, rx, ry, degRotate, degStart, degEnd float64)
func (f *Fpdf) BeginLayer(id int)
func (f *Fpdf) Beziergon(points []PointType, styleStr string)
func (f *Fpdf) Bookmark(txtStr string, level int, y float64)
func (f *Fpdf) Cell(w, h float64, txtStr string)
func (f *Fpdf) CellFormat(w, h float64, txtStr, borderStr string, ln int, alignStr string, fill bool, ...)
func (f *Fpdf) Cellf(w, h float64, fmtStr string, args ...interface{ })
func (f *Fpdf) Circle(x, y, r float64, styleStr string)
func (f *Fpdf) ClearError()
func (f *Fpdf) ClipCircle(x, y, r float64, outline bool)
func (f *Fpdf) ClipEllipse(x, y, rx, ry float64, outline bool)
func (f *Fpdf) ClipEnd()
func (f *Fpdf) ClipPolygon(points []PointType, outline bool)
func (f *Fpdf) ClipRect(x, y, w, h float64, outline bool)
func (f *Fpdf) ClipRoundedRect(x, y, w, h, r float64, outline bool)
func (f *Fpdf) ClipRoundedRectExt(x, y, w, h, rTL, rTR, rBR, rBL float64, outline bool)
func (f *Fpdf) ClipText(x, y float64, txtStr string, outline bool)
func (f *Fpdf) Close()
func (f *Fpdf) ClosePath()
func (f *Fpdf) CreateTemplate(fn func(*Tpl)) Template
func (f *Fpdf) CreateTemplateCustom(corner PointType, size SizeType, fn func(*Tpl)) Template
func (f *Fpdf) Curve(x0, y0, cx, cy, x1, y1 float64, styleStr string)
func (f *Fpdf) CurveBezierCubic(x0, y0, cx0, cy0, cx1, cy1, x1, y1 float64, styleStr string)
func (f *Fpdf) CurveBezierCubicTo(cx0, cy0, cx1, cy1, x, y float64)
func (f *Fpdf) CurveCubic(x0, y0, cx0, cy0, x1, y1, cx1, cy1 float64, styleStr string)
func (f *Fpdf) CurveTo(cx, cy, x, y float64)
func (f *Fpdf) DrawPath(styleStr string)
func (f *Fpdf) Ellipse(x, y, rx, ry, degRotate float64, styleStr string)
func (f *Fpdf) EndLayer()
func (f *Fpdf) Err() bool
func (f *Fpdf) Error() error
func (f *Fpdf) GetAlpha() (alpha float64, blendModeStr string)
func (f *Fpdf) GetAutoPageBreak() (auto bool, margin float64)
func (f *Fpdf) GetCellMargin() float64
func (f *Fpdf) GetConversionRatio() float64
func (f *Fpdf) GetDrawColor() (int, int, int)
func (f *Fpdf) GetDrawSpotColor() (name string, c, m, y, k byte)
func (f *Fpdf) GetFillColor() (int, int, int)
func (f *Fpdf) GetFillSpotColor() (name string, c, m, y, k byte)
func (f *Fpdf) GetFontDesc(familyStr, styleStr string) FontDescType
func (f *Fpdf) GetFontSize() (ptSize, unitSize float64)
func (f *Fpdf) GetImageInfo(imageStr string) (info *ImageInfoType)
```

func (f *Fpdf) GetLineWidth() float64
func (f *Fpdf) GetMargins() (left, top, right, bottom float64)
func (f *Fpdf) GetPageSize() (width, height float64)
func (f *Fpdf) GetPageSizeStr(sizeStr string) (size SizeType)
func (f *Fpdf) GetStringSymbolWidth(s string) int
func (f *Fpdf) GetStringWidth(s string) float64
func (f *Fpdf) GetTextColor() (int, int, int)
func (f *Fpdf) GetTextSpotColor() (name string, c, m, y, k byte)
func (f *Fpdf) GetX() float64
func (f *Fpdf) GetXY() (float64, float64)
func (f *Fpdf) GetY() float64
func (f *Fpdf) HTMLBasicNew() (html HTMLBasicType)
func (f *Fpdf) Image(imageNameStr string, x, y, w, h float64, flow bool, tp string, link int, ...)
func (f *Fpdf) ImageOptions(imageNameStr string, x, y, w, h float64, flow bool, options ImageOptions, ...)
func (f *Fpdf) ImageTypeFromMime(mimeStr string) (tp string)
func (f *Fpdf) ImportObjPos(objPos map[string]map[int]string)
func (f *Fpdf) ImportObjects(objs map[string][]byte)
func (f *Fpdf) ImportTemplates(tpls map[string]string)
func (f *Fpdf) LTR()
func (f *Fpdf) Line(x1, y1, x2, y2 float64)
func (f *Fpdf) LineTo(x, y float64)
func (f *Fpdf) LinearGradient(x, y, w, h float64, r1, g1, b1, r2, g2, b2 int, x1, y1, x2, y2 float64)
func (f *Fpdf) Link(x, y, w, h float64, link int)
func (f *Fpdf) LinkString(x, y, w, h float64, linkStr string)
func (f *Fpdf) Ln(h float64)
func (f *Fpdf) MoveTo(x, y float64)
func (f *Fpdf) MultiCell(w, h float64, txtStr, borderStr, alignStr string, fill bool)
func (f *Fpdf) Ok() bool
func (f *Fpdf) OpenLayerPane()
func (f *Fpdf) Output(w io.Writer) error
func (f *Fpdf) OutputAndClose(w io.WriteCloser) error
func (f *Fpdf) OutputFileAndClose(fileStr string) error
func (f *Fpdf) PageCount() int
func (f *Fpdf) PageNo() int
func (f *Fpdf) PageSize(pageNum int) (wd, ht float64, unitStr string)
func (f *Fpdf) PointConvert(pt float64) (u float64)
func (f *Fpdf) PointToUnitConvert(pt float64) (u float64)
func (f *Fpdf) Polygon(points []PointType, styleStr string)
func (f *Fpdf) RTL()
func (f *Fpdf) RadialGradient(x, y, w, h float64, r1, g1, b1, r2, g2, b2 int, x1, y1, x2, y2, r float64)
func (f *Fpdf) RawWriteBuf(r io.Reader)
func (f *Fpdf) RawWriteStr(str string)
func (f *Fpdf) Rect(x, y, w, h float64, styleStr string)
func (f *Fpdf) RegisterAlias(alias, replacement string)
func (f *Fpdf) RegisterImage(fileStr, tp string) (info *ImageInfoType)
func (f *Fpdf) RegisterImageOptions(fileStr string, options ImageOptions) (info *ImageInfoType)
func (f *Fpdf) RegisterImageOptionsReader(imgName string, options ImageOptions, r io.Reader) (info *ImageInfoType)
func (f *Fpdf) RegisterImageReader(imgName, tp string, r io.Reader) (info *ImageInfoType)
func (f *Fpdf) RoundedRect(x, y, w, h, r float64, corners string, stylestr string)
func (f *Fpdf) RoundedRectExt(x, y, w, h, rTL, rTR, rBR, rBL float64, stylestr string)
func (f *Fpdf) SVGBasicWrite(sb *SVGBasicType, scale float64)
func (f *Fpdf) SetAcceptPageBreakFunc(fnc func() bool)

```
func (f *Fpdf) SetAlpha(alpha float64, blendModeStr string)
func (f *Fpdf) SetAttachments(as []Attachment)
func (f *Fpdf) SetAuthor(authorStr string, isUTF8 bool)
func (f *Fpdf) SetAutoPageBreak(auto bool, margin float64)
func (f *Fpdf) SetCatalogSort(flag bool)
func (f *Fpdf) SetCellMargin(margin float64)
func (f *Fpdf) SetCompression(compress bool)
func (f *Fpdf) SetCreationDate(tm time.Time)
func (f *Fpdf) SetCreator(creatorStr string, isUTF8 bool)
func (f *Fpdf) SetDashPattern(dashArray []float64, dashPhase float64)
func (f *Fpdf) SetDisplayMode(zoomStr, layoutStr string)
func (f *Fpdf) SetDrawColor(r, g, b int)
func (f *Fpdf) SetDrawSpotColor(nameStr string, tint byte)
func (f *Fpdf) SetError(err error)
func (f *Fpdf) SetErrorf(fmtStr string, args ...interface{ })
func (f *Fpdf) SetFillColor(r, g, b int)
func (f *Fpdf) SetFillSpotColor(nameStr string, tint byte)
func (f *Fpdf) SetFont(familyStr, styleStr string, size float64)
func (f *Fpdf) SetFontLoader(loader FontLoader)
func (f *Fpdf) SetFontLocation(fontDirStr string)
func (f *Fpdf) SetFontSize(size float64)
func (f *Fpdf) SetFontStyle(styleStr string)
func (f *Fpdf) SetFontUnitSize(size float64)
func (f *Fpdf) SetFooterFunc(fnc func())
func (f *Fpdf) SetFooterFuncLpi(fnc func(lastPage bool))
func (f *Fpdf) SetHeaderFunc(fnc func())
func (f *Fpdf) SetHeaderFuncMode(fnc func(), homeMode bool)
func (f *Fpdf) SetHomeXY()
func (f *Fpdf) SetJavascript(script string)
func (f *Fpdf) SetKeywords(keywordsStr string, isUTF8 bool)
func (f *Fpdf) SetLeftMargin(margin float64)
func (f *Fpdf) SetLineCapStyle(styleStr string)
func (f *Fpdf) SetLineJoinStyle(styleStr string)
func (f *Fpdf) SetLineWidth(width float64)
func (f *Fpdf) SetLink(link int, y float64, page int)
func (f *Fpdf) SetMargins(left, top, right float64)
func (f *Fpdf) SetModificationDate(tm time.Time)
func (f *Fpdf) SetPage(pageNum int)
func (f *Fpdf) SetPageBox(t string, x, y, wd, ht float64)
func (f *Fpdf) SetPageBoxRec(t string, pb PageBox)
func (f *Fpdf) SetProducer(producerStr string, isUTF8 bool)
func (f *Fpdf) SetProtection(actionFlag byte, userPassStr, ownerPassStr string)
func (f *Fpdf) SetRightMargin(margin float64)
func (f *Fpdf) SetSubject(subjectStr string, isUTF8 bool)
func (f *Fpdf) SetTextColor(r, g, b int)
func (f *Fpdf) SetTextRenderingMode(mode int)
func (f *Fpdf) SetTextSpotColor(nameStr string, tint byte)
func (f *Fpdf) SetTitle(titleStr string, isUTF8 bool)
func (f *Fpdf) SetTopMargin(margin float64)
func (f *Fpdf) SetUnderlineThickness(thickness float64)
func (f *Fpdf) SetWordSpacing(space float64)
func (f *Fpdf) SetX(x float64)
func (f *Fpdf) SetXY(x, y float64)
```

```

func (f *Fpdf) SetXmpMetadata(xmpStream []byte)
func (f *Fpdf) SetY(y float64)
func (f *Fpdf) SplitLines(txt []byte, w float64) [][]byte
func (f *Fpdf) SplitText(txt string, w float64) (lines []string)
func (f *Fpdf) String() string
func (f *Fpdf) SubWrite(ht float64, str string, subFontSize, subOffset float64, link int, ...)
func (f *Fpdf) Text(x, y float64, txtStr string)
func (f *Fpdf) Transform(tm TransformMatrix)
func (f *Fpdf) TransformBegin()
func (f *Fpdf) TransformEnd()
func (f *Fpdf) TransformMirrorHorizontal(x float64)
func (f *Fpdf) TransformMirrorLine(angle, x, y float64)
func (f *Fpdf) TransformMirrorPoint(x, y float64)
func (f *Fpdf) TransformMirrorVertical(y float64)
func (f *Fpdf) TransformRotate(angle, x, y float64)
func (f *Fpdf) TransformScale(scaleWd, scaleHt, x, y float64)
func (f *Fpdf) TransformScaleX(scaleWd, x, y float64)
func (f *Fpdf) TransformScaleXY(s, x, y float64)
func (f *Fpdf) TransformScaleY(scaleHt, x, y float64)
func (f *Fpdf) TransformSkew(angleX, angleY, x, y float64)
func (f *Fpdf) TransformSkewX(angleX, x, y float64)
func (f *Fpdf) TransformSkewY(angleY, x, y float64)
func (f *Fpdf) TransformTranslate(tx, ty float64)
func (f *Fpdf) TransformTranslateX(tx float64)
func (f *Fpdf) TransformTranslateY(ty float64)
func (f *Fpdf) UnicodeTranslatorFromDescriptor(cpStr string) (rep func(string) string)
func (f *Fpdf) UnitToPointConvert(u float64) (pt float64)
func (f *Fpdf) UseImportedTemplate(tplName string, scaleX float64, scaleY float64, tX float64, tY float64)
func (f *Fpdf) UseTemplate(t Template)
func (f *Fpdf) UseTemplateScaled(t Template, corner PointType, size SizeType)
func (f *Fpdf) Write(h float64, txtStr string)
func (f *Fpdf) WriteAligned(width, lineHeight float64, textStr, alignStr string)
func (f *Fpdf) WriteLinkID(h float64, displayStr string, linkID int)
func (f *Fpdf) WriteLinkString(h float64, displayStr, targetStr string)
func (f *Fpdf) Writetf(h float64, fmtStr string, args ...interface{ })
type FpdfTpl
func (t *FpdfTpl) Bytes() []byte
func (t *FpdfTpl) FromPage(page int) (Template, error)
func (t *FpdfTpl) FromPages() []Template
func (t *FpdfTpl) GobDecode(buf []byte) error
func (t *FpdfTpl) GobEncode() ([]byte, error)
func (t *FpdfTpl) ID() string
func (t *FpdfTpl) Images() map[string]*ImageInfoType
func (t *FpdfTpl) NumPages() int
func (t *FpdfTpl) Serialize() ([]byte, error)
func (t *FpdfTpl) Size() (corner PointType, size SizeType)
func (t *FpdfTpl) Templates() []Template
type GridType
func NewGrid(x, y, w, h float64) (grid GridType)
func (g GridType) Grid(pdf *Fpdf)
func (g GridType) Ht(dataHt float64) float64
func (g GridType) HtAbs(dataHt float64) float64
func (g GridType) Plot(pdf *Fpdf, xMin, xMax float64, count int, fnc func(x float64) (y float64))

```

```

func (g GridType) Pos(xRel, yRel float64) (x, y float64)
func (g *GridType) TickmarksContainX(min, max float64)
func (g *GridType) TickmarksContainY(min, max float64)
func (g *GridType) TickmarksExtentX(min, div float64, count int)
func (g *GridType) TickmarksExtentY(min, div float64, count int)
func (g GridType) Wd(dataWd float64) float64
func (g GridType) WdAbs(dataWd float64) float64
func (g GridType) X(dataX float64) float64
func (g GridType) XRange() (min, max float64)
func (g GridType) XY(dataX, dataY float64) (x, y float64)
func (g GridType) Y(dataY float64) float64
func (g GridType) YRange() (min, max float64)
type HTMLBasicSegmentType
func HTMLBasicTokenize(htmlStr string) (list []HTMLBasicSegmentType)
type HTMLBasicType
func (html *HTMLBasicType) Write(lineHt float64, htmlStr string)
type ImageInfoType
func (info *ImageInfoType) Extent() (wd, ht float64)
func (info *ImageInfoType) GobDecode(buf []byte) (err error)
func (info *ImageInfoType) GobEncode() (buf []byte, err error)
func (info *ImageInfoType) Height() float64
func (info *ImageInfoType) SetDpi(dpi float64)
func (info *ImageInfoType) Width() float64
type ImageOptions
type InitType
type PageBox
type Pdf
type PointType
func (p *PointType) Transform(x, y float64) PointType
func (p PointType) XY() (float64, float64)
type RGBAType
type RGBType
type SVGBasicSegmentType
type SVGBasicType
func SVGBasicFileParse(svgFileStr string) (sig SVGBasicType, err error)
func SVGBasicParse(buf []byte) (sig SVGBasicType, err error)
type SizeType
func (s *SizeType) Orientation() string
func (s *SizeType) ScaleBy(factor float64) SizeType
func (s *SizeType) ScaleToHeight(height float64) SizeType
func (s *SizeType) ScaleToWidth(width float64) SizeType
type StateType
func StateGet(pdf *Fpdf) (st StateType)
func (st StateType) Put(pdf *Fpdf)
type Template
func CreateTemplate(corner PointType, size SizeType, unitStr, fontDirStr string, fn func(*Tpl)) Template
func CreateTpl(corner PointType, size SizeType, orientationStr, unitStr, fontDirStr string, ...) Template
func DeserializeTemplate(b []byte) (Template, error)
type TickFormatFncType
type Tpl
type TransformMatrix
type TtfType
func TtfParse(fileStr string) (TtfRec TtfType, err error)

```

Examples ¶

Package

Fpdf.AddAttachmentAnnotation
Fpdf.AddFont
Fpdf.AddFontFromBytes
Fpdf.AddLayer
Fpdf.AddPage
Fpdf.AddSpotColor
Fpdf.AddUTF8Font
Fpdf.Bezierngon
Fpdf.Bookmark
Fpdf.Cell (Strikeout)
Fpdf.CellFormat (Align)
Fpdf.CellFormat (Codepage)
Fpdf.CellFormat (Codepageescape)
Fpdf.CellFormat (Tables)
Fpdf.Circle
Fpdf.ClipRect
Fpdf.ClipText
Fpdf.CreateTemplate
Fpdf.DrawPath
Fpdf.GetStringWidth
Fpdf.HTMLBasicNew
Fpdf.Image
Fpdf.ImageOptions
Fpdf.LinearGradient
Fpdf.MoveTo
Fpdf.MultiCell
Fpdf.PageSize
Fpdf.Polygon
Fpdf.Rect
Fpdf.RegisterAlias
Fpdf.RegisterAlias (Utf8)
Fpdf.RegisterImage
Fpdf.RegisterImageOptionsReader
Fpdf.RegisterImageReader
Fpdf.RoundedRect
Fpdf.SVGBasicWrite
Fpdf.SetAcceptPageBreakFunc
Fpdf.SetAlpha
Fpdf.SetAttachments
Fpdf.SetFillColor
Fpdf.SetFontLoader
Fpdf.SetJavascript
Fpdf.SetKeywords
Fpdf.SetLeftMargin
Fpdf.SetLineJoinStyle
Fpdf.SetModificationDate
Fpdf.SetPage
Fpdf.SetPageBox
Fpdf.SetProtection
Fpdf.SetTextRenderingMode
Fpdf.SetUnderlineThickness

Fpdf.SplitLines
Fpdf.SplitLines (Tables)
Fpdf.SubWrite
Fpdf.TransformBegin
Fpdf.TransformRotate
Fpdf.WriteAligned
NewGrid
TtfParse
UTF8CutFont
Constants ¶
View Source

```
const (  
    // OrientationPortrait represents the portrait orientation.  
    OrientationPortrait = "portrait"  
  
    // OrientationLandscape represents the landscape orientation.  
    OrientationLandscape = "landscape"  
)
```

View Source

```
const (  
    // UnitPoint represents the size unit point  
    UnitPoint = "pt"  
    // UnitMillimeter represents the size unit millimeter  
    UnitMillimeter = "mm"  
    // UnitCentimeter represents the size unit centimeter  
    UnitCentimeter = "cm"  
    // UnitInch represents the size unit inch  
    UnitInch = "inch"  
)
```

View Source

```
const (  
    // PageSizeA3 represents DIN/ISO A3 page size  
    PageSizeA3 = "A3"  
    // PageSizeA4 represents DIN/ISO A4 page size  
    PageSizeA4 = "A4"  
    // PageSizeA5 represents DIN/ISO A5 page size  
    PageSizeA5 = "A5"  
    // PageSizeLetter represents US Letter page size  
    PageSizeLetter = "Letter"  
    // PageSizeLegal represents US Legal page size  
    PageSizeLegal = "Legal"  
)
```

View Source

```
const (  
    // BorderNone set no border  
    BorderNone = ""  
    // BorderFull sets a full border  
    BorderFull = "1"  
    // BorderLeft sets the border on the left side  
    BorderLeft = "L"  
    // BorderTop sets the border at the top  
    BorderTop = "T"  
    // BorderRight sets the border on the right side
```



```
BorderRight = "R"  
// BorderBottom sets the border on the bottom  
BorderBottom = "B"
```

)

[View Source](#)

```
const (  
// LineBreakNone disables linebreak  
LineBreakNone = 0  
// LineBreakNormal enables normal linebreak  
LineBreakNormal = 1  
// LineBreakBelow enables linebreak below  
LineBreakBelow = 2
```

)

[View Source](#)

```
const (  
// AlignLeft left aligns the cell  
AlignLeft = "L"  
// AlignRight right aligns the cell  
AlignRight = "R"  
// AlignCenter centers the cell  
AlignCenter = "C"  
// AlignTop aligns the cell to the top  
AlignTop = "T"  
// AlignBottom aligns the cell to the bottom  
AlignBottom = "B"  
// AlignMiddle aligns the cell to the middle  
AlignMiddle = "M"  
// AlignBaseline aligns the cell to the baseline  
AlignBaseline = "B"
```

)

[View Source](#)

```
const (  
// FontFlagFixedPitch is set if all glyphs have the same width (as  
// opposed to proportional or variable-pitch fonts, which have  
// different widths).  
FontFlagFixedPitch = 1 << 0  
// FontFlagSerif is set if glyphs have serifs, which are short  
// strokes drawn at an angle on the top and bottom of glyph stems.  
// (Sans serif fonts do not have serifs.)  
FontFlagSerif = 1 << 1  
// FontFlagSymbolic is set if font contains glyphs outside the  
// Adobe standard Latin character set. This flag and the  
// Nonsymbolic flag shall not both be set or both be clear.  
FontFlagSymbolic = 1 << 2  
// FontFlagScript is set if glyphs resemble cursive handwriting.  
FontFlagScript = 1 << 3  
// FontFlagNonsymbolic is set if font uses the Adobe standard  
// Latin character set or a subset of it.  
FontFlagNonsymbolic = 1 << 5  
// FontFlagItalic is set if glyphs have dominant vertical strokes  
// that are slanted.  
FontFlagItalic = 1 << 6  
// FontFlagAllCap is set if font contains no lowercase letters;
```

```
// typically used for display purposes, such as for titles or
// headlines.
FontFlagAllCap = 1 << 16
// SmallCap is set if font contains both uppercase and lowercase
// letters. The uppercase letters are similar to those in the
// regular version of the same typeface family. The glyphs for the
// lowercase letters have the same shapes as the corresponding
// uppercase letters, but they are sized and their proportions
// adjusted so that they have the same size and stroke weight as
// lowercase glyphs in the same typeface family.
SmallCap = 1 << 18
// ForceBold determines whether bold glyphs shall be painted with
// extra pixels even at very small text sizes by a conforming
// reader. If the ForceBold flag is set, features of bold glyphs
// may be thickened at small text sizes.
ForceBold = 1 << 18
)
Font flags for FontDescType.Flags as defined in the pdf specification.
```

View Source

```
const (
    CnProtectPrint    = 4
    CnProtectModify   = 8
    CnProtectCopy     = 16
    CnProtectAnnotForms = 32
)
Advisory bitflag constants that control document activities
```

Variables ¶

This section is empty.

Functions ¶

func CompareBytes ¶

func CompareBytes(s1, s2 []byte, printDiff bool) (err error)

CompareBytes compares the bytes referred to by s1 with those referred to by s2. Nil is returned if the buffers are equal, otherwise an error.

func ComparePDFFiles ¶

func ComparePDFFiles(file1Str, file2Str string, printDiff bool) (err error)

ComparePDFFiles reads and compares the full contents of the two specified files byte-for-byte. Nil is returned if the file contents are equal, or if the second file is missing, otherwise an error.

func ComparePDFs ¶

func ComparePDFs(rdr1, rdr2 io.Reader, printDiff bool) (err error)

ComparePDFs reads and compares the full contents of the two specified readers byte-for-byte. Nil is returned if the buffers are equal, otherwise an error.

func MakeFont ¶

func MakeFont(fontFileStr, encodingFileStr, dstDirStr string, msgWriter io.Writer, embed bool) error

MakeFont generates a font definition file in JSON format. A definition file of this type is required to use non-core fonts in the PDF documents that gofpdf generates. See the makefont utility in the gofpdf package for a command line interface to this function.

fontFileStr is the name of the TrueType file (extension .ttf), OpenType file (extension .otf) or binary Type1 file (extension .pfb) from which to generate a definition file. If an OpenType file is specified, it must be one that is based on TrueType outlines, not PostScript outlines; this cannot be determined from the file extension alone. If a Type1 file is specified, a metric file with the same pathname except with the extension .afm must be present.

encodingFileStr is the name of the encoding file that corresponds to the font.

dstDirStr is the name of the directory in which to save the definition file and, if embed is true, the compressed font file.

msgWriter is the writer that is called to display messages throughout the process. Use nil to turn off messages.

embed is true if the font is to be embedded in the PDF files.

func SetDefaultCatalogSort Â¶

func SetDefaultCatalogSort(flag bool)

SetDefaultCatalogSort sets the default value of the catalog sort flag that will be used when initializing a new Fpdf instance. See SetCatalogSort() for more details.

func SetDefaultCompression Â¶

func SetDefaultCompression(compress bool)

SetDefaultCompression controls the default setting of the internal compression flag. See SetCompression() for more details. Compression is on by default.

func SetDefaultCreationDate Â¶

func SetDefaultCreationDate(tm time.Time)

SetDefaultCreationDate sets the default value of the document creation date that will be used when initializing a new Fpdf instance. See SetCreationDate() for more details.

func SetDefaultModificationDate Â¶

added in v1.16.0

func SetDefaultModificationDate(tm time.Time)

SetDefaultModificationDate sets the default value of the document modification date that will be used when initializing a new Fpdf instance. See SetCreationDate() for more details.

func TickmarkPrecision Â¶

added in v1.0.1

func TickmarkPrecision(div float64) int

TickmarkPrecision returns an appropriate precision value for label formatting.

func Tickmarks Â¶

added in v1.0.1

func Tickmarks(min, max float64) (list []float64, precision int)

Tickmarks returns a slice of tickmarks appropriate for a chart axis and an appropriate precision for formatting purposes. The values min and max will be contained within the tickmark range.

func UTF8CutFont Â¶

added in v1.5.0

func UTF8CutFont(inBuf []byte, cutset string) (outBuf []byte)

UTF8CutFont is a utility function that generates a TrueType font composed only of the runes included in cutset. The rune glyphs are copied from This function is demonstrated in ExampleUTF8CutFont().

Example ¶

```
func UnicodeTranslator ¶
```

```
func UnicodeTranslator(r io.Reader) (f func(string) string, err error)
```

UnicodeTranslator returns a function that can be used to translate, where possible, utf-8 strings to a form that is compatible with the specified code page. The returned function accepts a string and returns a string.

r is a reader that should read a buffer made up of content lines that pertain to the code page of interest. Each line is made up of three whitespace separated fields. The first begins with "!" and is followed by two hexadecimal digits that identify the glyph position in the code page of interest. The second field begins with "U+" and is followed by the unicode code point value. The third is the glyph name. A number of these code page map files are packaged with the gfpdf library in the font directory.

An error occurs only if a line is read that does not conform to the expected format. In this case, the returned function is valid but does not perform any rune translation.

```
func UnicodeTranslatorFromFile ¶
```

```
func UnicodeTranslatorFromFile(fileStr string) (f func(string) string, err error)
```

UnicodeTranslatorFromFile returns a function that can be used to translate, where possible, utf-8 strings to a form that is compatible with the specified code page. See UnicodeTranslator for more details.

fileStr identifies a font descriptor file that maps glyph positions to names.

If an error occurs reading the file, the returned function is valid but does not perform any rune translation.

Types ¶

```
type Attachment ¶
```

added in v1.15.0

```
type Attachment struct {
```

```
    Content []byte
```

```
// Filename is the displayed name of the attachment
```

```
Filename string
```

```
// Description is only displayed when using AddAttachmentAnnotation(),
```

```
// and might be modified by the pdf reader.
```

```
Description string
```

```
// contains filtered or unexported fields
```

```
}
```

Attachment defines a content to be included in the pdf, in one of the following ways :

associated with the document as a whole : see SetAttachments()

accessible via a link localized on a page : see AddAttachmentAnnotation()

```
type FontDescType ¶
```

```
type FontDescType struct {
```

```
// The maximum height above the baseline reached by glyphs in this
```

```
// font (for example for "S"). The height of glyphs for accented
```

```
// characters shall be excluded.
```

```
Ascent int
```

```
// The maximum depth below the baseline reached by glyphs in this
```

```
// font. The value shall be a negative number.
```

```
Descent int
```

```
// The vertical coordinate of the top of flat capital letters,
```

```
// measured from the baseline (for example "H").
```

CapHeight int

// A collection of flags defining various characteristics of the
// font. (See the FontFlag* constants.)

Flags int

// A rectangle, expressed in the glyph coordinate system, that
// shall specify the font bounding box. This should be the smallest
// rectangle enclosing the shape that would result if all of the
// glyphs of the font were placed with their origins coincident
// and then filled.

FontBBox fontBoxType

// The angle, expressed in degrees counterclockwise from the
// vertical, of the dominant vertical strokes of the font. (The
// 9-oâ€™clock position is 90 degrees, and the 3-oâ€™clock position
// is â€“90 degrees.) The value shall be negative for fonts that
// slope to the right, as almost all italic fonts do.

ItalicAngle int

// The thickness, measured horizontally, of the dominant vertical
// stems of glyphs in the font.

StemV int

// The width to use for character codes whose widths are not
// specified in a font dictionaryâ€™s Widths array. This shall have
// a predictable effect only if all such codes map to glyphs whose
// actual widths are the same as the value of the MissingWidth
// entry. (Default value: 0.)

MissingWidth int

}

FontDescType (font descriptor) specifies metrics and other attributes of a font, as distinct from the metrics of individual glyphs (as defined in the pdf specification).

type FontLoader Â¶

type FontLoader interface {

Open(name string) (io.Reader, error)

}

FontLoader is used to read fonts (JSON font specification and zlib compressed font binaries) from arbitrary locations (e.g. files, zip files, embedded font resources).

Open provides an io.Reader for the specified font file (.json or .z). The file name never includes a path. Open returns an error if the specified file cannot be opened.

type Fpdf Â¶

type Fpdf struct {

// contains filtered or unexported fields

}

Fpdf is the principal structure for creating a single PDF document

func New Â¶

func New(orientationStr, unitStr, sizeStr, fontDirStr string) (f *Fpdf)

New returns a pointer to a new Fpdf instance. Its methods are subsequently called to produce a single PDF document.

orientationStr specifies the default page orientation. For portrait mode, specify "P" or "Portrait". For landscape mode, specify "L" or "Landscape". An empty string will be replaced with "P".

unitStr specifies the unit of length used in size parameters for elements other than fonts, which are always measured in points. Specify "pt" for point, "mm" for millimeter, "cm" for centimeter, or "in" for inch. An empty string will be replaced with "mm".

sizeStr specifies the page size. Acceptable values are "A3", "A4", "A5", "Letter", "Legal", or "Tabloid". An empty string will be replaced with "A4".

fontDirStr specifies the file system location in which font resources will be found. An empty string is replaced with ".". This argument only needs to reference an actual directory if a font other than one of the core fonts is used. The core fonts are "courier", "helvetica" (also called "arial"), "times", and "zapfdingbats" (also called "symbol").

func NewCustom ¶

func NewCustom(init *InitType) (f *Fpdf)

NewCustom returns a pointer to a new Fpdf instance. Its methods are subsequently called to produce a single PDF document. NewCustom() is an alternative to New() that provides additional customization. The PageSize() example demonstrates this method.

func (*Fpdf) AddAttachmentAnnotation ¶

added in v1.15.0

func (f *Fpdf) AddAttachmentAnnotation(a *Attachment, x, y, w, h float64)

AddAttachmentAnnotation puts a link on the current page, on the rectangle defined by `x`, `y`, `w`, `h`. This link points towards the content defined in `a`, which is embedded in the document. Note that no drawing is done by this method : a method like `Cell()` or `Rect()` should be called to indicate to the reader that there is a link here. Requiring a pointer to an Attachment avoids useless copies in the resulting pdf: attachment pointing to the same data will have their content only be included once, and be shared amongst all links. Be aware that not all PDF readers support annotated attachments. See the AddAttachmentAnnotation example for a demonstration of this method.

Example ¶

func (*Fpdf) AddFont ¶

func (f *Fpdf) AddFont(familyStr, styleStr, fileStr string)

AddFont imports a TrueType, OpenType or Type1 font and makes it available. It is necessary to generate a font definition file first with the makefont utility. It is not necessary to call this function for the core PDF fonts (courier, helvetica, times, zapfdingbats).

The JSON definition file (and the font file itself when embedding) must be present in the font directory. If it is not found, the error "Could not include font definition file" is set.

family specifies the font family. The name can be chosen arbitrarily. If it is a standard family name, it will override the corresponding font. This string is used to subsequently set the font with the SetFont method.

style specifies the font style. Acceptable values are (case insensitive) the empty string for regular style, "B" for bold, "I" for italic, or "BI" or "IB" for bold and italic combined.

fileStr specifies the base name with ".json" extension of the font definition file to be added. The file will be loaded from the font directory specified in the call to New() or SetFontLocation().

Example ¶

func (*Fpdf) AddFontFromBytes ¶

func (f *Fpdf) AddFontFromBytes(familyStr, styleStr string, jsonFileBytes, zFileBytes []byte)

AddFontFromBytes imports a TrueType, OpenType or Type1 font from static bytes within the executable and makes it available for use in the generated document.

family specifies the font family. The name can be chosen arbitrarily. If it is a standard family name, it will override the corresponding font. This string is used to subsequently set the font with the `SetFont` method.

style specifies the font style. Acceptable values are (case insensitive) the empty string for regular style, "B" for bold, "I" for italic, or "BI" or "IB" for bold and italic combined.

jsonFileBytes contain all bytes of JSON file.

zFileBytes contain all bytes of Z file.

Example ¶

```
func (*Fpdf) AddFontFromReader ¶
```

```
func (f *Fpdf) AddFontFromReader(familyStr, styleStr string, r io.Reader)
```

AddFontFromReader imports a TrueType, OpenType or Type1 font and makes it available using a reader that satisfies the `io.Reader` interface. See `AddFont` for details about `familyStr` and `styleStr`.

```
func (*Fpdf) AddLayer ¶
```

```
func (f *Fpdf) AddLayer(name string, visible bool) (layerID int)
```

AddLayer defines a layer that can be shown or hidden when the document is displayed. `name` specifies the layer name that the document reader will display in the layer list. `visible` specifies whether the layer will be initially visible. The return value is an integer ID that is used in a call to `BeginLayer()`.

Example ¶

```
func (*Fpdf) AddLink ¶
```

```
func (f *Fpdf) AddLink() int
```

AddLink creates a new internal link and returns its identifier. An internal link is a clickable area which directs to another place within the document. The identifier can then be passed to `Cell()`, `Write()`, `Image()` or `Link()`. The destination is defined with `SetLink()`.

```
func (*Fpdf) AddPage ¶
```

```
func (f *Fpdf) AddPage()
```

AddPage adds a new page to the document. If a page is already present, the `Footer()` method is called first to output the footer. Then the page is added, the current position set to the top-left corner according to the left and top margins, and `Header()` is called to display the header.

The font which was set before calling is automatically restored. There is no need to call `SetFont()` again if you want to continue with the same font. The same is true for colors and line width.

The origin of the coordinate system is at the top-left corner and increasing ordinates go downwards.

See `AddPageFormat()` for a version of this method that allows the page size and orientation to be different than the default.

Example ¶

```
func (*Fpdf) AddPageFormat ¶
```

```
func (f *Fpdf) AddPageFormat(orientationStr string, size.SizeType)
```

AddPageFormat adds a new page with non-default orientation or size. See `AddPage()` for more details.

See `New()` for a description of `orientationStr`.

size specifies the size of the new page in the units established in `New()`.

The `PageSize()` example demonstrates this method.

```
func (*Fpdf) AddSpotColor Â¶
```

added in v1.0.1

```
func (f *Fpdf) AddSpotColor(nameStr string, c, m, y, k byte)
```

`AddSpotColor` adds an ink-based CMYK color to the `goFPDF` instance and associates it with the specified name. The individual components specify percentages ranging from 0 to 100. Values above this are quietly capped to 100. An error occurs if the specified name is already associated with a color.

Example Â¶

```
func (*Fpdf) AddUTF8Font Â¶
```

added in v1.3.0

```
func (f *Fpdf) AddUTF8Font(familyStr, styleStr, fileStr string)
```

`AddUTF8Font` imports a TrueType font with utf-8 symbols and makes it available. It is necessary to generate a font definition file first with the `makefont` utility. It is not necessary to call this function for the core PDF fonts (courier, helvetica, times, zapfdingbats).

The JSON definition file (and the font file itself when embedding) must be present in the font directory. If it is not found, the error "Could not include font definition file" is set.

`family` specifies the font family. The name can be chosen arbitrarily. If it is a standard family name, it will override the corresponding font. This string is used to subsequently set the font with the `SetFont` method.

`style` specifies the font style. Acceptable values are (case insensitive) the empty string for regular style, "B" for bold, "I" for italic, or "BI" or "IB" for bold and italic combined.

`fileStr` specifies the base name with ".json" extension of the font definition file to be added. The file will be loaded from the font directory specified in the call to `New()` or `SetFontLocation()`.

Example Â¶

```
func (*Fpdf) AddUTF8FontFromBytes Â¶
```

added in v1.3.0

```
func (f *Fpdf) AddUTF8FontFromBytes(familyStr, styleStr string, utf8Bytes []byte)
```

`AddUTF8FontFromBytes` imports a TrueType font with utf-8 symbols from static bytes within the executable and makes it available for use in the generated document.

`family` specifies the font family. The name can be chosen arbitrarily. If it is a standard family name, it will override the corresponding font. This string is used to subsequently set the font with the `SetFont` method.

`style` specifies the font style. Acceptable values are (case insensitive) the empty string for regular style, "B" for bold, "I" for italic, or "BI" or "IB" for bold and italic combined.

`jsonFileBytes` contain all bytes of JSON file.

`zFileBytes` contain all bytes of Z file.

```
func (*Fpdf) AliasNbPages Â¶
```

```
func (f *Fpdf) AliasNbPages(aliasStr string)
```

`AliasNbPages` defines an alias for the total number of pages. It will be substituted as the document is closed. An empty string is replaced with the string "{nb}".

See the example for `AddPage()` for a demonstration of this method.


```
func (*Fpdf) Arc Â¶
```

```
func (f *Fpdf) Arc(x, y, rx, ry, degRotate, degStart, degEnd float64, styleStr string)
```

Arc draws an elliptical arc centered at point (x, y). rx and ry specify its horizontal and vertical radii.

degRotate specifies the angle that the arc will be rotated. degStart and degEnd specify the starting and ending angle of the arc. All angles are specified in degrees and measured counter-clockwise from the 3 o'clock position.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color, line width, and cap style centered on the arc's path. Filling uses the current fill color.

The Circle() example demonstrates this method.

```
func (*Fpdf) ArcTo Â¶
```

```
func (f *Fpdf) ArcTo(x, y, rx, ry, degRotate, degStart, degEnd float64)
```

ArcTo draws an elliptical arc centered at point (x, y). rx and ry specify its horizontal and vertical radii. If the start of the arc is not at the current position, a connecting line will be drawn.

degRotate specifies the angle that the arc will be rotated. degStart and degEnd specify the starting and ending angle of the arc. All angles are specified in degrees and measured counter-clockwise from the 3 o'clock position.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color, line width, and cap style centered on the arc's path. Filling uses the current fill color.

The MoveTo() example demonstrates this method.

```
func (*Fpdf) BeginLayer Â¶
```

```
func (f *Fpdf) BeginLayer(id int)
```

BeginLayer is called to begin adding content to the specified layer. All content added to the page between a call to BeginLayer and a call to EndLayer is added to the layer specified by id. See AddLayer for more details.

```
func (*Fpdf) Beziern Â¶
```

```
func (f *Fpdf) Beziern(points []PointType, styleStr string)
```

Beziern draws a closed figure defined by a series of cubic Bézier curve segments. The first point in the slice defines the starting point of the figure. Each three following points p1, p2, p3 represent a curve segment to the point p3 using p1 and p2 as the Bézier control points.

The x and y fields of the points use the units established in New().

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color and line width centered on the ellipse's perimeter. Filling uses the current fill color.

```
Example Â¶
```

```
func (*Fpdf) Bookmark Â¶
```

```
func (f *Fpdf) Bookmark(txtStr string, level int, y float64)
```

Bookmark sets a bookmark that will be displayed in a sidebar outline. txtStr is the title of the bookmark. level specifies the level of the bookmark in the outline; 0 is the top level, 1 is just below, and so on. y specifies the vertical position of the bookmark destination in the current page; -1 indicates the current position.

Example (Cell) ¶

```
func (*Fpdf) Cell ¶
```

```
func (f *Fpdf) Cell(w, h float64, txtStr string)
```

Cell is a simpler version of CellFormat with no fill, border, links or special alignment. The Cell_strikeout() example demonstrates this method.

Example (Strikeout) ¶

```
func (*Fpdf) CellFormat ¶
```

```
func (f *Fpdf) CellFormat(w, h float64, txtStr, borderStr string, ln int,
    alignStr string, fill bool, link int, linkStr string)
```

CellFormat prints a rectangular cell with optional borders, background color and character string. The upper-left corner of the cell corresponds to the current position. The text can be aligned or centered. After the call, the current position moves to the right or to the next line. It is possible to put a link on the text.

An error will be returned if a call to SetFont() has not already taken place before this method is called.

If automatic page breaking is enabled and the cell goes beyond the limit, a page break is done before outputting.

w and h specify the width and height of the cell. If w is 0, the cell extends up to the right margin. Specifying 0 for h will result in no output, but the current position will be advanced by w.

txtStr specifies the text to display.

borderStr specifies how the cell border will be drawn. An empty string indicates no border, "1" indicates a full border, and one or more of "L", "T", "R" and "B" indicate the left, top, right and bottom sides of the border.

ln indicates where the current position should go after the call. Possible values are 0 (to the right), 1 (to the beginning of the next line), and 2 (below). Putting 1 is equivalent to putting 0 and calling Ln() just after.

alignStr specifies how the text is to be positioned within the cell. Horizontal alignment is controlled by including "L", "C" or "R" (left, center, right) in alignStr. Vertical alignment is controlled by including "T", "M", "B" or "A" (top, middle, bottom, baseline) in alignStr. The default alignment is left middle.

fill is true to paint the cell background or false to leave it transparent.

link is the identifier returned by AddLink() or 0 for no internal link.

linkStr is a target URL or empty for no external link. A non-zero value for link takes precedence over linkStr.

Example (Align) ¶

Example (Codepage) ¶

Example (Codepageescape) ¶

Example (Tables) ¶

```
func (*Fpdf) Cellf ¶
```

```
func (f *Fpdf) Cellf(w, h float64, fmtStr string, args ...interface{ })
```

Cellf is a simpler printf-style version of CellFormat with no fill, border, links or special alignment. See documentation for the fmt package for details on fmtStr and args.

```
func (*Fpdf) Circle ¶
```

```
func (f *Fpdf) Circle(x, y, r float64, styleStr string)
```

Circle draws a circle centered on point (x, y) with radius r.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color and line width centered on the circle's perimeter. Filling uses the current fill color.

Example \hat{A}

```
func (*Fpdf) ClearError  $\hat{A}$ 
```

```
func (f *Fpdf) ClearError()
```

ClearError unsets the internal Fpdf error. This method should be used with care, as an internal error condition usually indicates an unrecoverable problem with the generation of a document. It is intended to deal with cases in which an error is used to select an alternate form of the document.

```
func (*Fpdf) ClipCircle  $\hat{A}$ 
```

```
func (f *Fpdf) ClipCircle(x, y, r float64, outline bool)
```

ClipCircle begins a circular clipping operation. The circle is centered at (x, y) and has radius r. outline is true to draw a border with the current draw color and line width centered on the circle's perimeter. Only the outer half of the border will be shown. After calling this method, all rendering operations (for example, Image(), LinearGradient(), etc) will be clipped by the specified circle. Call ClipEnd() to restore unclipped operations.

The ClipText() example demonstrates this method.

```
func (*Fpdf) ClipEllipse  $\hat{A}$ 
```

```
func (f *Fpdf) ClipEllipse(x, y, rx, ry float64, outline bool)
```

ClipEllipse begins an elliptical clipping operation. The ellipse is centered at (x, y). Its horizontal and vertical radii are specified by rx and ry. outline is true to draw a border with the current draw color and line width centered on the ellipse's perimeter. Only the outer half of the border will be shown. After calling this method, all rendering operations (for example, Image(), LinearGradient(), etc) will be clipped by the specified ellipse. Call ClipEnd() to restore unclipped operations.

This ClipText() example demonstrates this method.

```
func (*Fpdf) ClipEnd  $\hat{A}$ 
```

```
func (f *Fpdf) ClipEnd()
```

ClipEnd ends a clipping operation that was started with a call to ClipRect(), ClipRoundedRect(), ClipText(), ClipEllipse(), ClipCircle() or ClipPolygon(). Clipping operations can be nested. The document cannot be successfully output while a clipping operation is active.

The ClipText() example demonstrates this method.

```
func (*Fpdf) ClipPolygon  $\hat{A}$ 
```

```
func (f *Fpdf) ClipPolygon(points []PointType, outline bool)
```

ClipPolygon begins a clipping operation within a polygon. The figure is defined by a series of vertices specified by points. The x and y fields of the points use the units established in New(). The last point in the slice will be implicitly joined to the first to close the polygon. outline is true to draw a border with the current draw color and line width centered on the polygon's perimeter. Only the outer half of the border will be shown. After calling this method, all rendering operations (for example, Image(), LinearGradient(), etc) will be clipped by the specified polygon. Call ClipEnd() to restore unclipped operations.

The ClipText() example demonstrates this method.

```
func (*Fpdf) ClipRect  $\hat{A}$ 
```

```
func (f *Fpdf) ClipRect(x, y, w, h float64, outline bool)
```

ClipRect begins a rectangular clipping operation. The rectangle is of width w and height h. Its upper left corner is positioned at point (x, y). outline is true to draw a border with the current draw color and line width

centered on the rectangle's perimeter. Only the outer half of the border will be shown. After calling this method, all rendering operations (for example, Image(), LinearGradient(), etc) will be clipped by the specified rectangle. Call ClipEnd() to restore unclipped operations.

This ClipText() example demonstrates this method.

Example Â¶

```
func (*Fpdf) ClipRoundedRect Â¶
```

```
func (f *Fpdf) ClipRoundedRect(x, y, w, h, r float64, outline bool)
```

ClipRoundedRect begins a rectangular clipping operation. The rectangle is of width w and height h. Its upper left corner is positioned at point (x, y). The rounded corners of the rectangle are specified by radius r. outline is true to draw a border with the current draw color and line width centered on the rectangle's perimeter. Only the outer half of the border will be shown. After calling this method, all rendering operations (for example, Image(), LinearGradient(), etc) will be clipped by the specified rectangle. Call ClipEnd() to restore unclipped operations.

This ClipText() example demonstrates this method.

```
func (*Fpdf) ClipRoundedRectExt Â¶
```

added in v1.14.0

```
func (f *Fpdf) ClipRoundedRectExt(x, y, w, h, rTL, rTR, rBR, rBL float64, outline bool)
```

ClipRoundedRectExt behaves the same as ClipRoundedRect() but supports a different radius for each corner, given by rTL (top-left), rTR (top-right) rBR (bottom-right), rBL (bottom-left). See ClipRoundedRect() for more details. This method is demonstrated in the ClipText() example.

```
func (*Fpdf) ClipText Â¶
```

```
func (f *Fpdf) ClipText(x, y float64, txtStr string, outline bool)
```

ClipText begins a clipping operation in which rendering is confined to the character string specified by txtStr. The origin (x, y) is on the left of the first character at the baseline. The current font is used. outline is true to draw a border with the current draw color and line width centered on the perimeters of the text characters. Only the outer half of the border will be shown. After calling this method, all rendering operations (for example, Image(), LinearGradient(), etc) will be clipped. Call ClipEnd() to restore unclipped operations.

Example Â¶

```
func (*Fpdf) Close Â¶
```

```
func (f *Fpdf) Close()
```

Close terminates the PDF document. It is not necessary to call this method explicitly because Output(), OutputAndClose() and OutputFileAndClose() do it automatically. If the document contains no page, AddPage() is called to prevent the generation of an invalid document.

```
func (*Fpdf) ClosePath Â¶
```

```
func (f *Fpdf) ClosePath()
```

ClosePath creates a line from the current location to the last MoveTo point (if not the same) and mark the path as closed so the first and last lines join nicely.

The MoveTo() example demonstrates this method.

```
func (*Fpdf) CreateTemplate Â¶
```

```
func (f *Fpdf) CreateTemplate(fn func(*Tpl)) Template
```

CreateTemplate defines a new template using the current page size.

Example Â¶

```
func (*Fpdf) CreateTemplateCustom Â¶
```

func (f *Fpdf) CreateTemplateCustom(corner PointType, size SizeType, fn func(*Tpl)) Template
CreateTemplateCustom starts a template, using the given bounds.

func (*Fpdf) Curve \hat{A}

func (f *Fpdf) Curve(x0, y0, cx, cy, x1, y1 float64, styleStr string)

Curve draws a single-segment quadratic Bézier curve. The curve starts at the point (x0, y0) and ends at the point (x1, y1). The control point (cx, cy) specifies the curvature. At the start point, the curve is tangent to the straight line between the start point and the control point. At the end point, the curve is tangent to the straight line between the end point and the control point.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color, line width, and cap style centered on the curve's path. Filling uses the current fill color.

The Circle() example demonstrates this method.

func (*Fpdf) CurveBezierCubic \hat{A}

func (f *Fpdf) CurveBezierCubic(x0, y0, cx0, cy0, cx1, cy1, x1, y1 float64, styleStr string)

CurveBezierCubic draws a single-segment cubic Bézier curve. The curve starts at the point (x0, y0) and ends at the point (x1, y1). The control points (cx0, cy0) and (cx1, cy1) specify the curvature. At the start point, the curve is tangent to the straight line between the start point and the control point (cx0, cy0). At the end point, the curve is tangent to the straight line between the end point and the control point (cx1, cy1).

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color, line width, and cap style centered on the curve's path. Filling uses the current fill color.

This routine performs the same function as CurveCubic() but uses standard argument order.

The Circle() example demonstrates this method.

func (*Fpdf) CurveBezierCubicTo \hat{A}

func (f *Fpdf) CurveBezierCubicTo(cx0, cy0, cx1, cy1, x, y float64)

CurveBezierCubicTo creates a single-segment cubic Bézier curve. The curve starts at the current stylus location and ends at the point (x, y). The control points (cx0, cy0) and (cx1, cy1) specify the curvature. At the current stylus, the curve is tangent to the straight line between the current stylus location and the control point (cx0, cy0). At the end point, the curve is tangent to the straight line between the end point and the control point (cx1, cy1).

The MoveTo() example demonstrates this method.

func (*Fpdf) CurveCubic \hat{A}

func (f *Fpdf) CurveCubic(x0, y0, cx0, cy0, x1, y1, cx1, cy1 float64, styleStr string)

CurveCubic draws a single-segment cubic Bézier curve. This routine performs the same function as CurveBezierCubic() but has a nonstandard argument order. It is retained to preserve backward compatibility.

func (*Fpdf) CurveTo \hat{A}

func (f *Fpdf) CurveTo(cx, cy, x, y float64)

CurveTo creates a single-segment quadratic Bézier curve. The curve starts at the current stylus location and ends at the point (x, y). The control point (cx, cy) specifies the curvature. At the start point, the curve is tangent to the straight line between the current stylus location and the control point. At the end point, the curve is tangent to the straight line between the end point and the control point.

The MoveTo() example demonstrates this method.

```
func (*Fpdf) DrawPath Â¶
```

```
func (f *Fpdf) DrawPath(styleStr string)
```

DrawPath actually draws the path on the page.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Path-painting operators as defined in the PDF specification are also allowed: "S" (Stroke the path), "s" (Close and stroke the path), "f" (fill the path, using the nonzero winding number), "f*" (Fill the path, using the even-odd rule), "B" (Fill and then stroke the path, using the nonzero winding number rule), "B*" (Fill and then stroke the path, using the even-odd rule), "b" (Close, fill, and then stroke the path, using the nonzero winding number rule) and "b*" (Close, fill, and then stroke the path, using the even-odd rule). Drawing uses the current draw color, line width, and cap style centered on the path. Filling uses the current fill color.

The MoveTo() example demonstrates this method.

```
Example Â¶
```

```
func (*Fpdf) Ellipse Â¶
```

```
func (f *Fpdf) Ellipse(x, y, rx, ry, degRotate float64, styleStr string)
```

Ellipse draws an ellipse centered at point (x, y). rx and ry specify its horizontal and vertical radii.

degRotate specifies the counter-clockwise angle in degrees that the ellipse will be rotated.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color and line width centered on the ellipse's perimeter. Filling uses the current fill color.

The Circle() example demonstrates this method.

```
func (*Fpdf) EndLayer Â¶
```

```
func (f *Fpdf) EndLayer()
```

EndLayer is called to stop adding content to the currently active layer. See BeginLayer for more details.

```
func (*Fpdf) Err Â¶
```

```
func (f *Fpdf) Err() bool
```

Err returns true if a processing error has occurred.

```
func (*Fpdf) Error Â¶
```

```
func (f *Fpdf) Error() error
```

Error returns the internal Fpdf error; this will be nil if no error has occurred.

```
func (*Fpdf) GetAlpha Â¶
```

```
func (f *Fpdf) GetAlpha() (alpha float64, blendModeStr string)
```

GetAlpha returns the alpha blending channel, which consists of the alpha transparency value and the blend mode. See SetAlpha for more details.

```
func (*Fpdf) GetAutoPageBreak Â¶
```

added in v1.0.1

```
func (f *Fpdf) GetAutoPageBreak() (auto bool, margin float64)
```

GetAutoPageBreak returns true if automatic pages breaks are enabled, false otherwise. This is followed by the triggering limit from the bottom of the page. This value applies only if automatic page breaks are enabled.

func (*Fpdf) GetCellMargin Â¶

func (f *Fpdf) GetCellMargin() float64

GetCellMargin returns the cell margin. This is the amount of space before and after the text within a cell that's left blank, and is in units passed to New(). It defaults to 1mm.

func (*Fpdf) GetConversionRatio Â¶

func (f *Fpdf) GetConversionRatio() float64

GetConversionRatio returns the conversion ratio based on the unit given when creating the PDF.

func (*Fpdf) GetDrawColor Â¶

func (f *Fpdf) GetDrawColor() (int, int, int)

GetDrawColor returns the most recently set draw color as RGB components (0 - 255). This will not be the current value if a draw color of some other type (for example, spot) has been more recently set.

func (*Fpdf) GetDrawSpotColor Â¶

added in v1.0.1

func (f *Fpdf) GetDrawSpotColor() (name string, c, m, y, k byte)

GetDrawSpotColor returns the most recently used spot color information for drawing. This will not be the current drawing color if some other color type such as RGB is active. If no spot color has been set for drawing, zero values are returned.

func (*Fpdf) GetFillColor Â¶

func (f *Fpdf) GetFillColor() (int, int, int)

GetFillColor returns the most recently set fill color as RGB components (0 - 255). This will not be the current value if a fill color of some other type (for example, spot) has been more recently set.

func (*Fpdf) GetFillSpotColor Â¶

added in v1.0.1

func (f *Fpdf) GetFillSpotColor() (name string, c, m, y, k byte)

GetFillSpotColor returns the most recently used spot color information for fill output. This will not be the current fill color if some other color type such as RGB is active. If no fill spot color has been set, zero values are returned.

func (*Fpdf) GetFontDesc Â¶

func (f *Fpdf) GetFontDesc(familyStr, styleStr string) FontDescType

GetFontDesc returns the font descriptor, which can be used for example to find the baseline of a font. If familyStr is empty current font descriptor will be returned. See FontDescType for documentation about the font descriptor. See AddFont for details about familyStr and styleStr.

func (*Fpdf) GetFontSize Â¶

func (f *Fpdf) GetFontSize() (ptSize, unitSize float64)

GetFontSize returns the size of the current font in points followed by the size in the unit of measure specified in New(). The second value can be used as a line height value in drawing operations.

func (*Fpdf) GetImageInfo Â¶

func (f *Fpdf) GetImageInfo(imageStr string) (info *ImageInfoType)

GetImageInfo returns information about the registered image specified by imageStr. If the image has not been registered, nil is returned. The internal error is not modified by this method.

func (*Fpdf) GetLineWidth Â¶

func (f *Fpdf) GetLineWidth() float64

GetLineWidth returns the current line thickness.

func (*Fpdf) GetMargins Â¶

func (f *Fpdf) GetMargins() (left, top, right, bottom float64)

GetMargins returns the left, top, right, and bottom margins. The first three are set with the SetMargins() method. The bottom margin is set with the SetAutoPageBreak() method.

func (*Fpdf) GetPageSize Â¶

func (f *Fpdf) GetPageSize() (width, height float64)

GetPageSize returns the current page's width and height. This is the paper's size. To compute the size of the area being used, subtract the margins (see GetMargins()).

func (*Fpdf) GetPageSizeStr Â¶

added in v1.0.1

func (f *Fpdf) GetPageSizeStr(sizeStr string) (size SizeType)

GetPageSizeStr returns the SizeType for the given sizeStr (that is A4, A3, etc..)

func (*Fpdf) GetStringSymbolWidth Â¶

added in v1.3.0

func (f *Fpdf) GetStringSymbolWidth(s string) int

GetStringSymbolWidth returns the length of a string in glyph units. A font must be currently selected.

func (*Fpdf) GetStringWidth Â¶

func (f *Fpdf) GetStringWidth(s string) float64

GetStringWidth returns the length of a string in user units. A font must be currently selected.

Example Â¶

func (*Fpdf) GetTextColor Â¶

func (f *Fpdf) GetTextColor() (int, int, int)

GetTextColor returns the most recently set text color as RGB components (0 - 255). This will not be the current value if a text color of some other type (for example, spot) has been more recently set.

func (*Fpdf) GetTextSpotColor Â¶

added in v1.0.1

func (f *Fpdf) GetTextSpotColor() (name string, c, m, y, k byte)

GetTextSpotColor returns the most recently used spot color information for text output. This will not be the current text color if some other color type such as RGB is active. If no spot color has been set for text, zero values are returned.

func (*Fpdf) GetX Â¶

func (f *Fpdf) GetX() float64

GetX returns the abscissa of the current position.

Note: the value returned will be affected by the current cell margin. To account for this, you may need to either add the value returned by GetCellMargin() to it or call SetCellMargin(0) to remove the cell margin.

func (*Fpdf) GetXY Â¶

func (f *Fpdf) GetXY() (float64, float64)

GetXY returns the abscissa and ordinate of the current position.

Note: the value returned for the abscissa will be affected by the current cell margin. To account for this, you may need to either add the value returned by GetCellMargin() to it or call SetCellMargin(0) to remove the cell margin.

func (*Fpdf) GetY Â¶

func (f *Fpdf) GetY() float64

GetY returns the ordinate of the current position.

func (*Fpdf) HTMLBasicNew Â¶

func (f *Fpdf) HTMLBasicNew() (html HTMLBasicType)

HTMLBasicNew returns an instance that facilitates writing basic HTML in the specified PDF file.

Example Â¶

func (*Fpdf) Image Â¶

func (f *Fpdf) Image(imageNameStr string, x, y, w, h float64, flow bool, tp string, link int, linkStr string)

Image puts a JPEG, PNG or GIF image in the current page.

Deprecated in favor of ImageOptions -- see that function for details on the behavior of arguments

Example Â¶

func (*Fpdf) ImageOptions Â¶

func (f *Fpdf) ImageOptions(imageNameStr string, x, y, w, h float64, flow bool, options ImageOptions, link int, linkStr string)

ImageOptions puts a JPEG, PNG or GIF image in the current page. The size it will take on the page can be specified in different ways. If both w and h are 0, the image is rendered at 96 dpi. If either w or h is zero, it will be calculated from the other dimension so that the aspect ratio is maintained. If w and/or h are -1, the dpi for that dimension will be read from the ImageInfoType object. PNG files can contain dpi information, and if present, this information will be populated in the ImageInfoType object and used in Width, Height, and Extent calculations. Otherwise, the SetDpi function can be used to change the dpi from the default of 72.

If w and h are any other negative value, their absolute values indicate their dpi extents.

Supported JPEG formats are 24 bit, 32 bit and gray scale. Supported PNG formats are 24 bit, indexed color, and 8 bit indexed gray scale. If a GIF image is animated, only the first frame is rendered. Transparency is supported. It is possible to put a link on the image.

imageNameStr may be the name of an image as registered with a call to either RegisterImageReader() or RegisterImage(). In the first case, the image is loaded using an io.Reader. This is generally useful when the image is obtained from some other means than as a disk-based file. In the second case, the image is loaded as a file. Alternatively, imageNameStr may directly specify a sufficiently qualified filename.

However the image is loaded, if it is used more than once only one copy is embedded in the file.

If x is negative, the current abscissa is used.

If flow is true, the current y value is advanced after placing the image and a page break may be made if necessary.

If link refers to an internal page anchor (that is, it is non-zero; see AddLink()), the image will be a clickable internal link. Otherwise, if linkStr specifies a URL, the image will be a clickable external link.

Example Â¶

func (*Fpdf) ImageTypeFromMime Â¶

func (f *Fpdf) ImageTypeFromMime(mimeStr string) (tp string)

ImageTypeFromMime returns the image type used in various image-related functions (for example, Image()) that is associated with the specified MIME type. For example, "jpg" is returned if mimeStr is "image/jpeg". An error is set if the specified MIME type is not supported.

func (*Fpdf) ImportObjPos Â¶

added in v1.4.1

func (f *Fpdf) ImportObjPos(objPos map[string]map[int]string)

ImportObjPos imports object hash positions from gofpdi

func (*Fpdf) ImportObjects Â¶

added in v1.4.1

func (f *Fpdf) ImportObjects(objs map[string][]byte)

ImportObjects imports objects from gofpdi into current document

func (*Fpdf) ImportTemplates Â¶

added in v1.4.1

func (f *Fpdf) ImportTemplates(tpls map[string]string)

ImportTemplates imports gofpdi template names into importedTplObjs for inclusion in the procset dictionary

func (*Fpdf) LTR Â¶

added in v1.3.0

func (f *Fpdf) LTR()

LTR disables right-to-left mode

func (*Fpdf) Line Â¶

func (f *Fpdf) Line(x1, y1, x2, y2 float64)

Line draws a line between points (x1, y1) and (x2, y2) using the current draw color, line width and cap style.

func (*Fpdf) LineTo Â¶

func (f *Fpdf) LineTo(x, y float64)

LineTo creates a line from the current stylus location to (x, y), which becomes the new stylus location. Note that this only creates the line in the path; it does not actually draw the line on the page.

The MoveTo() example demonstrates this method.

func (*Fpdf) LinearGradient Â¶

func (f *Fpdf) LinearGradient(x, y, w, h float64, r1, g1, b1, r2, g2, b2 int, x1, y1, x2, y2 float64)

LinearGradient draws a rectangular area with a blending of one color to another. The rectangle is of width w and height h. Its upper left corner is positioned at point (x, y).

Each color is specified with three component values, one each for red, green and blue. The values range from 0 to 255. The first color is specified by (r1, g1, b1) and the second color by (r2, g2, b2).

The blending is controlled with a gradient vector that uses normalized coordinates in which the lower left corner is position (0, 0) and the upper right corner is (1, 1). The vector's origin and destination are specified by the points (x1, y1) and (x2, y2). In a linear gradient, blending occurs perpendicularly to the vector. The vector does not necessarily need to be anchored on the rectangle edge. Color 1 is used up to the origin of the vector and color 2 is used beyond the vector's end point. Between the points the colors are gradually blended.

Example Â¶

func (*Fpdf) Link Â¶

func (f *Fpdf) Link(x, y, w, h float64, link int)

Link puts a link on a rectangular area of the page. Text or image links are generally put via Cell(), Write() or Image(), but this method can be useful for instance to define a clickable area inside an image. link is the value returned by AddLink().

func (*Fpdf) LinkString Â¶

```
func (f *Fpdf) LinkString(x, y, w, h float64, linkStr string)
```

LinkString puts a link on a rectangular area of the page. Text or image links are generally put via Cell(), Write() or Image(), but this method can be useful for instance to define a clickable area inside an image. linkStr is the target URL.

```
func (*Fpdf) Ln ¶
```

```
func (f *Fpdf) Ln(h float64)
```

Ln performs a line break. The current abscissa goes back to the left margin and the ordinate increases by the amount passed in parameter. A negative value of h indicates the height of the last printed cell.

This method is demonstrated in the example for MultiCell.

```
func (*Fpdf) MoveTo ¶
```

```
func (f *Fpdf) MoveTo(x, y float64)
```

MoveTo moves the stylus to (x, y) without drawing the path from the previous point. Paths must start with a MoveTo to set the original stylus location or the result is undefined.

Create a "path" by moving a virtual stylus around the page (with MoveTo, LineTo, CurveTo, CurveBezierCubicTo, ArcTo & ClosePath) then draw it or fill it in (with DrawPath). The main advantage of using the path drawing routines rather than multiple Fpdf.Line is that PDF creates nice line joins at the angles, rather than just overlaying the lines.

```
Example ¶
```

```
func (*Fpdf) MultiCell ¶
```

```
func (f *Fpdf) MultiCell(w, h float64, txtStr, borderStr, alignStr string, fill bool)
```

MultiCell supports printing text with line breaks. They can be automatic (as soon as the text reaches the right border of the cell) or explicit (via the \n character). As many cells as necessary are output, one below the other.

Text can be aligned, centered or justified. The cell block can be framed and the background painted. See CellFormat() for more details.

The current position after calling MultiCell() is the beginning of the next line, equivalent to calling CellFormat with ln equal to 1.

w is the width of the cells. A value of zero indicates cells that reach to the right margin.

h indicates the line height of each cell in the unit of measure specified in New().

Note: this method has a known bug that treats UTF-8 fonts differently than non-UTF-8 fonts. With UTF-8 fonts, all trailing newlines in txtStr are removed. With a non-UTF-8 font, if txtStr has one or more trailing newlines, only the last is removed. In the next major module version, the UTF-8 logic will be changed to match the non-UTF-8 logic. To prepare for that change, applications that use UTF-8 fonts and depend on having all trailing newlines removed should call strings.TrimRight(txtStr, "\r\n") before calling this method.

```
Example ¶
```

ExampleFpdf_MultiCell demonstrates word-wrapping, line justification and page-breaking.

```
package main
```

```
import (
```

```
    "fmt"
```

```
    "io/ioutil"
```

```
"github.com/jung-kurt/gofpdf"
"github.com/jung-kurt/gofpdf/internal/example"
)
```

```
func main() {
    pdf := gofpdf.New("P", "mm", "A4", "")
    titleStr := "20000 Leagues Under the Seas"
    pdf.SetTitle(titleStr, false)
    pdf.SetAuthor("Jules Verne", false)
    pdf.SetHeaderFunc(func() {
        // Arial bold 15
        pdf.SetFont("Arial", "B", 15)
        // Calculate width of title and position
        wd := pdf.GetStringWidth(titleStr) + 6
        pdf.SetX((210 - wd) / 2)
        // Colors of frame, background and text
        pdf.SetDrawColor(0, 80, 180)
        pdf.SetFillColor(230, 230, 0)
        pdf.SetTextColor(220, 50, 50)
        // Thickness of frame (1 mm)
        pdf.SetLineWidth(1)
        // Title
        pdf.CellFormat(wd, 9, titleStr, "1", 1, "C", true, 0, "")
        // Line break
        pdf.Ln(10)
    })
    pdf.SetFooterFunc(func() {
        // Position at 1.5 cm from bottom
        pdf.SetY(-15)
        // Arial italic 8
        pdf.SetFont("Arial", "I", 8)
        // Text color in gray
        pdf.SetTextColor(128, 128, 128)
        // Page number
        pdf.CellFormat(0, 10, fmt.Sprintf("Page %d", pdf.PageNo()),
            "", 0, "C", false, 0, "")
    })
    chapterTitle := func(chapNum int, titleStr string) {
        // // Arial 12
        pdf.SetFont("Arial", "", 12)
        // Background color
        pdf.SetFillColor(200, 220, 255)
        // Title
        pdf.CellFormat(0, 6, fmt.Sprintf("Chapter %d : %s", chapNum, titleStr),
            "", 1, "L", true, 0, "")
        // Line break
        pdf.Ln(4)
    }
    chapterBody := func(fileStr string) {
        // Read text file
        txtStr, err := ioutil.ReadFile(fileStr)
        if err != nil {
            pdf.SetError(err)
        }
    }
}
```

```

}
// Times 12
pdf.SetFont("Times", "", 12)
// Output justified text
pdf.MultiCell(0, 5, string(txtStr), "", "", false)
// Line break
pdf.Ln(-1)
// Mention in italics
pdf.SetFont("", "I", 0)
pdf.Cell(0, 5, "(end of excerpt)")
}
printChapter := func(chapNum int, titleStr, fileStr string) {
pdf.AddPage()
chapterTitle(chapNum, titleStr)
chapterBody(fileStr)
}
printChapter(1, "A RUNAWAY REEF", example.TextFile("20k_c1.txt"))
printChapter(2, "THE PROS AND CONS", example.TextFile("20k_c2.txt"))
fileStr := example.Filename("Fpdf_MultiCell")
err := pdf.OutputFileAndClose(fileStr)
example.Summary(err, fileStr)
}

```

Output:

Successfully generated pdf/Fpdf_MultiCell.pdf

Share

Format

Run

func (*Fpdf) Ok ¶

func (f *Fpdf) Ok() bool

Ok returns true if no processing errors have occurred.

func (*Fpdf) OpenLayerPane ¶

func (f *Fpdf) OpenLayerPane()

OpenLayerPane advises the document reader to open the layer pane when the document is initially displayed.

func (*Fpdf) Output ¶

func (f *Fpdf) Output(w io.Writer) error

Output sends the PDF document to the writer specified by w. No output will take place if an error has occurred in the document generation process. w remains open after this function returns. After returning, f is in a closed state and its methods should not be called.

func (*Fpdf) OutputAndClose ¶

func (f *Fpdf) OutputAndClose(w io.WriteCloser) error

OutputAndClose sends the PDF document to the writer specified by w. This method will close both f and w, even if an error is detected and no document is produced.

func (*Fpdf) OutputFileAndClose ¶

func (f *Fpdf) OutputFileAndClose(fileStr string) error

OutputFileAndClose creates or truncates the file specified by fileStr and writes the PDF document to it. This method will close f and the newly written file, even if an error is detected and no document is produced.

Most examples demonstrate the use of this method.

```
func (*Fpdf) PageCount Â¶
```

added in v1.1.0

```
func (f *Fpdf) PageCount() int
```

PageCount returns the number of pages currently in the document. Since page numbers in gofpdf are one-based, the page count is the same as the page number of the current last page.

```
func (*Fpdf) PageNo Â¶
```

```
func (f *Fpdf) PageNo() int
```

PageNo returns the current page number.

See the example for AddPage() for a demonstration of this method.

```
func (*Fpdf) PageSize Â¶
```

```
func (f *Fpdf) PageSize(pageNum int) (wd, ht float64, unitStr string)
```

PageSize returns the width and height of the specified page in the units established in New(). These return values are followed by the unit of measure itself. If pageNum is zero or otherwise out of bounds, it returns the default page size, that is, the size of the page that would be added by AddPage().

Example Â¶

```
func (*Fpdf) PointConvert Â¶
```

```
func (f *Fpdf) PointConvert(pt float64) (u float64)
```

PointConvert returns the value of pt, expressed in points (1/72 inch), as a value expressed in the unit of measure specified in New(). Since font management in Fpdf uses points, this method can help with line height calculations and other methods that require user units.

```
func (*Fpdf) PointToUnitConvert Â¶
```

```
func (f *Fpdf) PointToUnitConvert(pt float64) (u float64)
```

PointToUnitConvert is an alias for PointConvert.

```
func (*Fpdf) Polygon Â¶
```

```
func (f *Fpdf) Polygon(points []PointType, styleStr string)
```

Polygon draws a closed figure defined by a series of vertices specified by points. The x and y fields of the points use the units established in New(). The last point in the slice will be implicitly joined to the first to close the polygon.

styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color and line width centered on the ellipse's perimeter. Filling uses the current fill color.

Example Â¶

```
func (*Fpdf) RTL Â¶
```

added in v1.3.0

```
func (f *Fpdf) RTL()
```

RTL enables right-to-left mode

```
func (*Fpdf) RadialGradient Â¶
```

```
func (f *Fpdf) RadialGradient(x, y, w, h float64, r1, g1, b1, r2, g2, b2 int, x1, y1, x2, y2, r float64)
```

RadialGradient draws a rectangular area with a blending of one color to another. The rectangle is of width w and height h. Its upper left corner is positioned at point (x, y).

Each color is specified with three component values, one each for red, green and blue. The values range from

0 to 255. The first color is specified by (r1, g1, b1) and the second color by (r2, g2, b2).

The blending is controlled with a point and a circle, both specified with normalized coordinates in which the lower left corner of the rendered rectangle is position (0, 0) and the upper right corner is (1, 1). Color 1 begins at the origin point specified by (x1, y1). Color 2 begins at the circle specified by the center point (x2, y2) and radius r. Colors are gradually blended from the origin to the circle. The origin and the circle's center do not necessarily have to coincide, but the origin must be within the circle to avoid rendering problems.

The LinearGradient() example demonstrates this method.

```
func (*Fpdf) RawWriteBuf Â¶
```

```
func (f *Fpdf) RawWriteBuf(r io.Reader)
```

RawWriteBuf writes the contents of the specified buffer directly to the PDF generation buffer. This is a low-level function that is not required for normal PDF construction. An understanding of the PDF specification is needed to use this method correctly.

```
func (*Fpdf) RawWriteStr Â¶
```

```
func (f *Fpdf) RawWriteStr(str string)
```

RawWriteStr writes a string directly to the PDF generation buffer. This is a low-level function that is not required for normal PDF construction. An understanding of the PDF specification is needed to use this method correctly.

```
func (*Fpdf) Rect Â¶
```

```
func (f *Fpdf) Rect(x, y, w, h float64, styleStr string)
```

Rect outputs a rectangle of width w and height h with the upper left corner positioned at point (x, y).

It can be drawn (border only), filled (with no border) or both. styleStr can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color and line width centered on the rectangle's perimeter. Filling uses the current fill color.

```
Example Â¶
```

```
func (*Fpdf) RegisterAlias Â¶
```

added in v1.0.1

```
func (f *Fpdf) RegisterAlias(alias, replacement string)
```

RegisterAlias adds an (alias, replacement) pair to the document so we can replace all occurrences of that alias after writing but before the document is closed. Functions ExampleFpdf_RegisterAlias() and ExampleFpdf_RegisterAlias_utf8() in fpdf_test.go demonstrate this method.

```
Example Â¶
```

```
Example (Utf8) Â¶
```

```
func (*Fpdf) RegisterImage Â¶
```

```
func (f *Fpdf) RegisterImage(fileStr, tp string) (info *ImageInfoType)
```

RegisterImage registers an image, adding it to the PDF file but not adding it to the page. Use Image() with the same filename to add the image to the page. Note that Image() calls this function, so this function is only necessary if you need information about the image before placing it.

This function is now deprecated in favor of RegisterImageOptions. See Image() for restrictions on the image and the "tp" parameters.

```
Example Â¶
```

```
func (*Fpdf) RegisterImageOptions Â¶
```

```
func (f *Fpdf) RegisterImageOptions(fileStr string, options ImageOptions) (info *ImageInfoType)
```

RegisterImageOptions registers an image, adding it to the PDF file but not adding it to the page. Use Image()

with the same filename to add the image to the page. Note that `Image()` calls this function, so this function is only necessary if you need information about the image before placing it. See `Image()` for restrictions on the image and the "tp" parameters.

```
func (*Fpdf) RegisterImageOptionsReader Â¶
```

```
func (f *Fpdf) RegisterImageOptionsReader(imgName string, options ImageOptions, r io.Reader) (info *ImageInfoType)
```

`RegisterImageOptionsReader` registers an image, reading it from `Reader r`, adding it to the PDF file but not adding it to the page. Use `Image()` with the same name to add the image to the page. Note that `tp` should be specified in this case.

See `Image()` for restrictions on the image and the options parameters.

Example Â¶

```
func (*Fpdf) RegisterImageReader Â¶
```

```
func (f *Fpdf) RegisterImageReader(imgName, tp string, r io.Reader) (info *ImageInfoType)
```

`RegisterImageReader` registers an image, reading it from `Reader r`, adding it to the PDF file but not adding it to the page.

This function is now deprecated in favor of `RegisterImageOptionsReader`

Example Â¶

```
func (*Fpdf) RoundedRect Â¶
```

added in v1.6.0

```
func (f *Fpdf) RoundedRect(x, y, w, h, r float64, corners string, stylestr string)
```

`RoundedRect` outputs a rectangle of width `w` and height `h` with the upper left corner positioned at point `(x, y)`. It can be drawn (border only), filled (with no border) or both. `styleStr` can be "F" for filled, "D" for outlined only, or "DF" or "FD" for outlined and filled. An empty string will be replaced with "D". Drawing uses the current draw color and line width centered on the rectangle's perimeter. Filling uses the current fill color. The rounded corners of the rectangle are specified by radius `r`. `corners` is a string that includes "1" to round the upper left corner, "2" to round the upper right corner, "3" to round the lower right corner, and "4" to round the lower left corner. The `RoundedRect` example demonstrates this method.

Example Â¶

```
func (*Fpdf) RoundedRectExt Â¶
```

added in v1.14.0

```
func (f *Fpdf) RoundedRectExt(x, y, w, h, rTL, rTR, rBR, rBL float64, stylestr string)
```

`RoundedRectExt` behaves the same as `RoundedRect()` but supports a different radius for each corner. A zero radius means squared corner. See `RoundedRect()` for more details. This method is demonstrated in the `RoundedRect()` example.

```
func (*Fpdf) SVGBasicWrite Â¶
```

```
func (f *Fpdf) SVGBasicWrite(sb *SVGBasicType, scale float64)
```

`SVGBasicWrite` renders the paths encoded in the basic SVG image specified by `sb`. The scale value is used to convert the coordinates in the path to the unit of measure specified in `New()`. The current position (as set with a call to `SetXY()`) is used as the origin of the image. The current line cap style (as set with `SetLineCapStyle()`), line width (as set with `SetLineWidth()`), and draw color (as set with `SetDrawColor()`) are used in drawing the image paths.

Example Â¶

```
func (*Fpdf) SetAcceptPageBreakFunc Â¶
```

```
func (f *Fpdf) SetAcceptPageBreakFunc(fnc func() bool)
```

`SetAcceptPageBreakFunc` allows the application to control where page breaks occur.

fnc is an application function (typically a closure) that is called by the library whenever a page break condition is met. The break is issued if true is returned. The default implementation returns a value according to the mode selected by SetAutoPageBreak. The function provided should not be called by the application.

See the example for SetLeftMargin() to see how this function can be used to manage multiple columns.

Example ¶

```
func (*Fpdf) SetAlpha ¶
```

```
func (f *Fpdf) SetAlpha(alpha float64, blendModeStr string)
```

SetAlpha sets the alpha blending channel. The blending effect applies to text, drawings and images.

alpha must be a value between 0.0 (fully transparent) to 1.0 (fully opaque). Values outside of this range result in an error.

blendModeStr must be one of "Normal", "Multiply", "Screen", "Overlay", "Darken", "Lighten", "ColorDodge", "ColorBurn", "HardLight", "SoftLight", "Difference", "Exclusion", "Hue", "Saturation", "Color", or "Luminosity". An empty string is replaced with "Normal".

To reset normal rendering after applying a blending mode, call this method with alpha set to 1.0 and blendModeStr set to "Normal".

Example ¶

```
func (*Fpdf) SetAttachments ¶
```

added in v1.15.0

```
func (f *Fpdf) SetAttachments(as []Attachment)
```

SetAttachments writes attachments as embedded files (document attachment). These attachments are global, see AddAttachmentAnnotation() for a link anchored in a page. Note that only the last call of SetAttachments is useful, previous calls are discarded. Be aware that not all PDF readers support document attachments. See the SetAttachment example for a demonstration of this method.

Example ¶

```
func (*Fpdf) SetAuthor ¶
```

```
func (f *Fpdf) SetAuthor(authorStr string, isUTF8 bool)
```

SetAuthor defines the author of the document. isUTF8 indicates if the string is encoded in ISO-8859-1 (false) or UTF-8 (true).

```
func (*Fpdf) SetAutoPageBreak ¶
```

```
func (f *Fpdf) SetAutoPageBreak(auto bool, margin float64)
```

SetAutoPageBreak enables or disables the automatic page breaking mode. When enabling, the second parameter is the distance from the bottom of the page that defines the triggering limit. By default, the mode is on and the margin is 2 cm.

```
func (*Fpdf) SetCatalogSort ¶
```

```
func (f *Fpdf) SetCatalogSort(flag bool)
```

SetCatalogSort sets a flag that will be used, if true, to consistently order the document's internal resource catalogs. This method is typically only used for test purposes to facilitate PDF comparison.

```
func (*Fpdf) SetCellMargin ¶
```

```
func (f *Fpdf) SetCellMargin(margin float64)
```

SetCellMargin sets the cell margin. This is the amount of space before and after the text within a cell that's left blank, and is in units passed to New().

func (*Fpdf) SetCompression Â¶

func (f *Fpdf) SetCompression(compress bool)

SetCompression activates or deactivates page compression with zlib. When activated, the internal representation of each page is compressed, which leads to a compression ratio of about 2 for the resulting document. Compression is on by default.

func (*Fpdf) SetCreationDate Â¶

func (f *Fpdf) SetCreationDate(tm time.Time)

SetCreationDate fixes the document's internal CreationDate value. By default, the time when the document is generated is used for this value. This method is typically only used for testing purposes to facilitate PDF comparison. Specify a zero-value time to revert to the default behavior.

func (*Fpdf) SetCreator Â¶

func (f *Fpdf) SetCreator(creatorStr string, isUTF8 bool)

SetCreator defines the creator of the document. isUTF8 indicates if the string is encoded in ISO-8859-1 (false) or UTF-8 (true).

func (*Fpdf) SetDashPattern Â¶

func (f *Fpdf) SetDashPattern(dashArray []float64, dashPhase float64)

SetDashPattern sets the dash pattern that is used to draw lines. The dashArray elements are numbers that specify the lengths, in units established in New(), of alternating dashes and gaps. The dash phase specifies the distance into the dash pattern at which to start the dash. The dash pattern is retained from page to page. Call this method with an empty array to restore solid line drawing.

The Beziergon() example demonstrates this method.

func (*Fpdf) SetDisplayMode Â¶

func (f *Fpdf) SetDisplayMode(zoomStr, layoutStr string)

SetDisplayMode sets advisory display directives for the document viewer. Pages can be displayed entirely on screen, occupy the full width of the window, use real size, be scaled by a specific zooming factor or use viewer default (configured in the Preferences menu of Adobe Reader). The page layout can be specified so that pages are displayed individually or in pairs.

zoomStr can be "fullpage" to display the entire page on screen, "fullwidth" to use maximum width of window, "real" to use real size (equivalent to 100% zoom) or "default" to use viewer default mode.

layoutStr can be "single" (or "SinglePage") to display one page at once, "continuous" (or "OneColumn") to display pages continuously, "two" (or "TwoColumnLeft") to display two pages on two columns with odd-numbered pages on the left, or "TwoColumnRight" to display two pages on two columns with odd-numbered pages on the right, or "TwoPageLeft" to display pages two at a time with odd-numbered pages on the left, or "TwoPageRight" to display pages two at a time with odd-numbered pages on the right, or "default" to use viewer default mode.

func (*Fpdf) SetDrawColor Â¶

func (f *Fpdf) SetDrawColor(r, g, b int)

SetDrawColor defines the color used for all drawing operations (lines, rectangles and cell borders). It is expressed in RGB components (0 - 255). The method can be called before the first page is created. The value is retained from page to page.

func (*Fpdf) SetDrawSpotColor Â¶

added in v1.0.1

func (f *Fpdf) SetDrawSpotColor(nameStr string, tint byte)

SetDrawSpotColor sets the current draw color to the spot color associated with nameStr. An error occurs if the

name is not associated with a color. The value for tint ranges from 0 (no intensity) to 100 (full intensity). It is quietly bounded to this range.

```
func (*Fpdf) SetError Â¶
```

```
func (f *Fpdf) SetError(err error)
```

SetError sets an error to halt PDF generation. This may facilitate error handling by application. See also Ok(), Err() and Error().

```
func (*Fpdf) SetErrorf Â¶
```

```
func (f *Fpdf) SetErrorf(fmtStr string, args ...interface{ })
```

SetErrorf sets the internal Fpdf error with formatted text to halt PDF generation; this may facilitate error handling by application. If an error condition is already set, this call is ignored.

See the documentation for printing in the standard fmt package for details about fmtStr and args.

```
func (*Fpdf) SetFillColor Â¶
```

```
func (f *Fpdf) SetFillColor(r, g, b int)
```

SetFillColor defines the color used for all filling operations (filled rectangles and cell backgrounds). It is expressed in RGB components (0 -255). The method can be called before the first page is created and the value is retained from page to page.

```
Example Â¶
```

```
func (*Fpdf) SetFillSpotColor Â¶
```

added in v1.0.1

```
func (f *Fpdf) SetFillSpotColor(nameStr string, tint byte)
```

SetFillSpotColor sets the current fill color to the spot color associated with nameStr. An error occurs if the name is not associated with a color. The value for tint ranges from 0 (no intensity) to 100 (full intensity). It is quietly bounded to this range.

```
func (*Fpdf) SetFont Â¶
```

```
func (f *Fpdf) SetFont(familyStr, styleStr string, size float64)
```

SetFont sets the font used to print character strings. It is mandatory to call this method at least once before printing text or the resulting document will not be valid.

The font can be either a standard one or a font added via the AddFont() method or AddFontFromReader() method. Standard fonts use the Windows encoding cp1252 (Western Europe).

The method can be called before the first page is created and the font is kept from page to page. If you just wish to change the current font size, it is simpler to call SetFontSize().

Note: the font definition file must be accessible. An error is set if the file cannot be read.

familyStr specifies the font family. It can be either a name defined by AddFont(), AddFontFromReader() or one of the standard families (case insensitive): "Courier" for fixed-width, "Helvetica" or "Arial" for sans serif, "Times" for serif, "Symbol" or "ZapfDingbats" for symbolic.

styleStr can be "B" (bold), "I" (italic), "U" (underscore), "S" (strike-out) or any combination. The default value (specified with an empty string) is regular. Bold and italic styles do not apply to Symbol and ZapfDingbats.

size is the font size measured in points. The default value is the current size. If no size has been specified since the beginning of the document, the value taken is 12.

```
func (*Fpdf) SetFontLoader Â¶
```

```
func (f *Fpdf) SetFontLoader(loader FontLoader)
```

SetFontLoader sets a loader used to read font files (.json and .z) from an arbitrary source. If a font loader has been specified, it is used to load the named font resources when AddFont() is called. If this operation fails, an attempt is made to load the resources from the configured font directory (see SetFontLocation()).

Example ¶

```
func (*Fpdf) SetFontLocation ¶
```

```
func (f *Fpdf) SetFontLocation(fontDirStr string)
```

SetFontLocation sets the location in the file system of the font and font definition files.

```
func (*Fpdf) SetFontSize ¶
```

```
func (f *Fpdf) SetFontSize(size float64)
```

SetFontSize defines the size of the current font. Size is specified in points (1/ 72 inch). See also SetFontUnitSize().

```
func (*Fpdf) SetFontStyle ¶
```

added in v1.2.0

```
func (f *Fpdf) SetFontStyle(styleStr string)
```

SetFontStyle sets the style of the current font. See also SetFont()

```
func (*Fpdf) SetFontUnitSize ¶
```

```
func (f *Fpdf) SetFontUnitSize(size float64)
```

SetFontUnitSize defines the size of the current font. Size is specified in the unit of measure specified in New(). See also SetFontSize().

```
func (*Fpdf) SetFooterFunc ¶
```

```
func (f *Fpdf) SetFooterFunc(fnc func())
```

SetFooterFunc sets the function that lets the application render the page footer. The specified function is automatically called by AddPage() and Close() and should not be called directly by the application. The implementation in Fpdf is empty, so you have to provide an appropriate function if you want page footers. fnc will typically be a closure that has access to the Fpdf instance and other document generation variables. See SetFooterFuncLpi for a similar function that passes a last page indicator.

This method is demonstrated in the example for AddPage().

```
func (*Fpdf) SetFooterFuncLpi ¶
```

added in v1.0.1

```
func (f *Fpdf) SetFooterFuncLpi(fnc func(lastPage bool))
```

SetFooterFuncLpi sets the function that lets the application render the page footer. The specified function is automatically called by AddPage() and Close() and should not be called directly by the application. It is passed a boolean that is true if the last page of the document is being rendered. The implementation in Fpdf is empty, so you have to provide an appropriate function if you want page footers. fnc will typically be a closure that has access to the Fpdf instance and other document generation variables.

```
func (*Fpdf) SetHeaderFunc ¶
```

```
func (f *Fpdf) SetHeaderFunc(fnc func())
```

SetHeaderFunc sets the function that lets the application render the page header. The specified function is automatically called by AddPage() and should not be called directly by the application. The implementation in Fpdf is empty, so you have to provide an appropriate function if you want page headers. fnc will typically be a closure that has access to the Fpdf instance and other document generation variables.

A header is a convenient place to put background content that repeats on each page such as a watermark. When this is done, remember to reset the X and Y values so the normal content begins where expected.

Including a watermark on each page is demonstrated in the example for TransformRotate.

This method is demonstrated in the example for AddPage().

```
func (*Fpdf) SetHeaderFuncMode Â¶  
added in v1.0.1
```

```
func (f *Fpdf) SetHeaderFuncMode(fnc func(), homeMode bool)
```

SetHeaderFuncMode sets the function that lets the application render the page header. See SetHeaderFunc() for more details. The value for homeMode should be set to true to have the current position set to the left and top margin after the header function is called.

```
func (*Fpdf) SetHomeXY Â¶  
added in v1.0.1
```

```
func (f *Fpdf) SetHomeXY()
```

SetHomeXY is a convenience method that sets the current position to the left and top margins.

```
func (*Fpdf) SetJavascript Â¶  
added in v1.0.1
```

```
func (f *Fpdf) SetJavascript(script string)
```

SetJavascript adds Adobe JavaScript to the document.

Example Â¶

```
func (*Fpdf) SetKeywords Â¶
```

```
func (f *Fpdf) SetKeywords(keywordsStr string, isUTF8 bool)
```

SetKeywords defines the keywords of the document. keywordStr is a space-delimited string, for example "invoice August". isUTF8 indicates if the string is encoded

Example Â¶

```
func (*Fpdf) SetLeftMargin Â¶
```

```
func (f *Fpdf) SetLeftMargin(margin float64)
```

SetLeftMargin defines the left margin. The method can be called before creating the first page. If the current abscissa gets out of page, it is brought back to the margin.

Example Â¶

```
func (*Fpdf) SetLineCapStyle Â¶
```

```
func (f *Fpdf) SetLineCapStyle(styleStr string)
```

SetLineCapStyle defines the line cap style. styleStr should be "butt", "round" or "square". A square style projects from the end of the line. The method can be called before the first page is created. The value is retained from page to page.

```
func (*Fpdf) SetLineJoinStyle Â¶
```

```
func (f *Fpdf) SetLineJoinStyle(styleStr string)
```

SetLineJoinStyle defines the line cap style. styleStr should be "miter", "round" or "bevel". The method can be called before the first page is created. The value is retained from page to page.

Example Â¶

```
func (*Fpdf) SetLineWidth Â¶
```

```
func (f *Fpdf) SetLineWidth(width float64)
```

SetLineWidth defines the line width. By default, the value equals 0.2 mm. The method can be called before the first page is created. The value is retained from page to page.

```
func (*Fpdf) SetLink Â¶
```

```
func (f *Fpdf) SetLink(link int, y float64, page int)
```

SetLink defines the page and position a link points to. See AddLink().

```
func (*Fpdf) SetMargins Â¶
```

```
func (f *Fpdf) SetMargins(left, top, right float64)
```

SetMargins defines the left, top and right margins. By default, they equal 1 cm. Call this method to change them. If the value of the right margin is less than zero, it is set to the same as the left margin.

```
func (*Fpdf) SetModificationDate Â¶
```

added in v1.16.0

```
func (f *Fpdf) SetModificationDate(tm time.Time)
```

SetModificationDate fixes the document's internal ModDate value. See `SetCreationDate` for more details.

```
Example Â¶
```

```
func (*Fpdf) SetPage Â¶
```

added in v1.0.2

```
func (f *Fpdf) SetPage(pageNum int)
```

SetPage sets the current page to that of a valid page in the PDF document. pageNum is one-based. The SetPage() example demonstrates this method.

```
Example Â¶
```

```
func (*Fpdf) SetPageBox Â¶
```

added in v1.0.1

```
func (f *Fpdf) SetPageBox(t string, x, y, wd, ht float64)
```

SetPageBox sets the page box for the current page, and any following pages. Allowable types are trim, trimbox, crop, cropbox, bleed, bleedbox, art and artbox box types are case insensitive.

```
Example Â¶
```

```
func (*Fpdf) SetPageBoxRec Â¶
```

added in v1.0.1

```
func (f *Fpdf) SetPageBoxRec(t string, pb PageBox)
```

SetPageBoxRec sets the page box for the current page, and any following pages. Allowable types are trim, trimbox, crop, cropbox, bleed, bleedbox, art and artbox box types are case insensitive. See SetPageBox() for a method that specifies the coordinates and extent of the page box individually.

```
func (*Fpdf) SetProducer Â¶
```

added in v1.8.0

```
func (f *Fpdf) SetProducer(producerStr string, isUTF8 bool)
```

SetProducer defines the producer of the document. isUTF8 indicates if the string is encoded in ISO-8859-1 (false) or UTF-8 (true).

```
func (*Fpdf) SetProtection Â¶
```

```
func (f *Fpdf) SetProtection(actionFlag byte, userPassStr, ownerPassStr string)
```

SetProtection applies certain constraints on the finished PDF document.

actionFlag is a bitflag that controls various document operations. CnProtectPrint allows the document to be printed. CnProtectModify allows a document to be modified by a PDF editor. CnProtectCopy allows text and images to be copied into the system clipboard. CnProtectAnnotForms allows annotations and forms to be added by a PDF editor. These values can be combined by or-ing them together, for example, CnProtectCopy|CnProtectModify. This flag is advisory; not all PDF readers implement the constraints that this argument attempts to control.

userPassStr specifies the password that will need to be provided to view the contents of the PDF. The permissions specified by actionFlag will apply.

ownerPassStr specifies the password that will need to be provided to gain full access to the document regardless of the actionFlag value. An empty string for this argument will be replaced with a random value, effectively prohibiting full access to the document.

Example ¶

```
func (*Fpdf) SetRightMargin ¶
```

```
func (f *Fpdf) SetRightMargin(margin float64)
```

SetRightMargin defines the right margin. The method can be called before creating the first page.

```
func (*Fpdf) SetSubject ¶
```

```
func (f *Fpdf) SetSubject(subjectStr string, isUTF8 bool)
```

SetSubject defines the subject of the document. isUTF8 indicates if the string is encoded in ISO-8859-1 (false) or UTF-8 (true).

```
func (*Fpdf) SetTextColor ¶
```

```
func (f *Fpdf) SetTextColor(r, g, b int)
```

SetTextColor defines the color used for text. It is expressed in RGB components (0 - 255). The method can be called before the first page is created. The value is retained from page to page.

```
func (*Fpdf) SetTextRenderingMode ¶
```

added in v1.13.0

```
func (f *Fpdf) SetTextRenderingMode(mode int)
```

SetTextRenderingMode sets the rendering mode of following text. The mode can be as follows: 0: Fill text 1: Stroke text 2: Fill, then stroke text 3: Neither fill nor stroke text (invisible) 4: Fill text and add to path for clipping 5: Stroke text and add to path for clipping 6: Fills then stroke text and add to path for clipping 7: Add text to path for clipping This method is demonstrated in the SetTextRenderingMode example.

Example ¶

```
func (*Fpdf) SetTextSpotColor ¶
```

added in v1.0.1

```
func (f *Fpdf) SetTextSpotColor(nameStr string, tint byte)
```

SetTextSpotColor sets the current text color to the spot color associated with nameStr. An error occurs if the name is not associated with a color. The value for tint ranges from 0 (no intensity) to 100 (full intensity). It is quietly bounded to this range.

```
func (*Fpdf) SetTitle ¶
```

```
func (f *Fpdf) SetTitle(titleStr string, isUTF8 bool)
```

SetTitle defines the title of the document. isUTF8 indicates if the string is encoded in ISO-8859-1 (false) or UTF-8 (true).

```
func (*Fpdf) SetTopMargin ¶
```

```
func (f *Fpdf) SetTopMargin(margin float64)
```

SetTopMargin defines the top margin. The method can be called before creating the first page.

```
func (*Fpdf) SetUnderlineThickness ¶
```

added in v1.10.0

```
func (f *Fpdf) SetUnderlineThickness(thickness float64)
```

SetUnderlineThickness accepts a multiplier for adjusting the text underline thickness, defaulting to 1. See SetUnderlineThickness example.

Example ¶

```
func (*Fpdf) SetWordSpacing ¶
```

added in v1.0.1

func (f *Fpdf) SetWordSpacing(space float64)

SetWordSpacing sets spacing between words of following text. See the WriteAligned() example for a demonstration of its use.

func (*Fpdf) SetX Â¶

func (f *Fpdf) SetX(x float64)

SetX defines the abscissa of the current position. If the passed value is negative, it is relative to the right of the page.

func (*Fpdf) SetXY Â¶

func (f *Fpdf) SetXY(x, y float64)

SetXY defines the abscissa and ordinate of the current position. If the passed values are negative, they are relative respectively to the right and bottom of the page.

func (*Fpdf) SetXmpMetadata Â¶

added in v1.0.1

func (f *Fpdf) SetXmpMetadata(xmpStream []byte)

SetXmpMetadata defines XMP metadata that will be embedded with the document.

func (*Fpdf) SetY Â¶

func (f *Fpdf) SetY(y float64)

SetY moves the current abscissa back to the left margin and sets the ordinate. If the passed value is negative, it is relative to the bottom of the page.

func (*Fpdf) SplitLines Â¶

func (f *Fpdf) SplitLines(txt []byte, w float64) [][]byte

SplitLines splits text into several lines using the current font. Each line has its length limited to a maximum width given by w. This function can be used to determine the total height of wrapped text for vertical placement purposes.

This method is useful for codepage-based fonts only. For UTF-8 encoded text, use SplitText().

You can use MultiCell if you want to print a text on several lines in a simple way.

Example Â¶

Example (Tables) Â¶

func (*Fpdf) SplitText Â¶

added in v1.4.0

func (f *Fpdf) SplitText(txt string, w float64) (lines []string)

SplitText splits UTF-8 encoded text into several lines using the current font. Each line has its length limited to a maximum width given by w. This function can be used to determine the total height of wrapped text for vertical placement purposes.

func (*Fpdf) String Â¶

func (f *Fpdf) String() string

String satisfies the fmt.Stringer interface and summarizes the Fpdf instance.

func (*Fpdf) SubWrite Â¶

added in v1.0.1

func (f *Fpdf) SubWrite(ht float64, str string, subFontSize, subOffset float64, link int, linkStr string)

SubWrite prints text from the current position in the same way as Write(). ht is the line height in the unit of measure specified in New(). str specifies the text to write. subFontSize is the size of the font in points.

subOffset is the vertical offset of the text in points; a positive value indicates a superscript, a negative value indicates a subscript. link is the identifier returned by AddLink() or 0 for no internal link. linkStr is a target URL or empty for no external link. A non--zero value for link takes precedence over linkStr.

The SubWrite example demonstrates this method.

Example ¶

```
func (*Fpdf) Text ¶
```

```
func (f *Fpdf) Text(x, y float64, txtStr string)
```

Text prints a character string. The origin (x, y) is on the left of the first character at the baseline. This method permits a string to be placed precisely on the page, but it is usually easier to use Cell(), MultiCell() or Write() which are the standard methods to print text.

```
func (*Fpdf) Transform ¶
```

```
func (f *Fpdf) Transform(tm TransformMatrix)
```

Transform generally transforms the following text, drawings and images according to the specified matrix. It is typically easier to use the various methods such as TransformRotate() and TransformMirrorVertical() instead.

```
func (*Fpdf) TransformBegin ¶
```

```
func (f *Fpdf) TransformBegin()
```

TransformBegin sets up a transformation context for subsequent text, drawings and images. The typical usage is to immediately follow a call to this method with a call to one or more of the transformation methods such as TransformScale(), TransformSkew(), etc. This is followed by text, drawing or image output and finally a call to TransformEnd(). All transformation contexts must be properly ended prior to outputting the document.

Example ¶

```
func (*Fpdf) TransformEnd ¶
```

```
func (f *Fpdf) TransformEnd()
```

TransformEnd applies a transformation that was begun with a call to TransformBegin().

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformMirrorHorizontal ¶
```

```
func (f *Fpdf) TransformMirrorHorizontal(x float64)
```

TransformMirrorHorizontal horizontally mirrors the following text, drawings and images. x is the axis of reflection.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformMirrorLine ¶
```

```
func (f *Fpdf) TransformMirrorLine(angle, x, y float64)
```

TransformMirrorLine symmetrically mirrors the following text, drawings and images on the line defined by angle and the point (x, y). angles is specified in degrees and measured counter-clockwise from the 3 o'clock position.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformMirrorPoint ¶
```

```
func (f *Fpdf) TransformMirrorPoint(x, y float64)
```

TransformMirrorPoint symmetrically mirrors the following text, drawings and images on the point specified by (x, y).

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformMirrorVertical Â¶
```

```
func (f *Fpdf) TransformMirrorVertical(y float64)
```

TransformMirrorVertical vertically mirrors the following text, drawings and images. y is the axis of reflection.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformRotate Â¶
```

```
func (f *Fpdf) TransformRotate(angle, x, y float64)
```

TransformRotate rotates the following text, drawings and images around the center point (x, y). angle is specified in degrees and measured counter-clockwise from the 3 o'clock position.

The TransformBegin() example demonstrates this method.

```
Example Â¶
```

```
func (*Fpdf) TransformScale Â¶
```

```
func (f *Fpdf) TransformScale(scaleWd, scaleHt, x, y float64)
```

TransformScale generally scales the following text, drawings and images. scaleWd and scaleHt are the percentage scaling factors for width and height. (x, y) is center of scaling.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformScaleX Â¶
```

```
func (f *Fpdf) TransformScaleX(scaleWd, x, y float64)
```

TransformScaleX scales the width of the following text, drawings and images. scaleWd is the percentage scaling factor. (x, y) is center of scaling.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformScaleXY Â¶
```

```
func (f *Fpdf) TransformScaleXY(s, x, y float64)
```

TransformScaleXY uniformly scales the width and height of the following text, drawings and images. s is the percentage scaling factor for both width and height. (x, y) is center of scaling.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformScaleY Â¶
```

```
func (f *Fpdf) TransformScaleY(scaleHt, x, y float64)
```

TransformScaleY scales the height of the following text, drawings and images. scaleHt is the percentage scaling factor. (x, y) is center of scaling.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformSkew Â¶
```

```
func (f *Fpdf) TransformSkew(angleX, angleY, x, y float64)
```

TransformSkew generally skews the following text, drawings and images keeping the point (x, y) stationary. angleX ranges from -90 degrees (skew to the left) to 90 degrees (skew to the right). angleY ranges from -90 degrees (skew to the bottom) to 90 degrees (skew to the top).

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformSkewX Â¶
```

```
func (f *Fpdf) TransformSkewX(angleX, x, y float64)
```

TransformSkewX horizontally skews the following text, drawings and images keeping the point (x, y) stationary. angleX ranges from -90 degrees (skew to the left) to 90 degrees (skew to the right).

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformSkewY Â¶
```

```
func (f *Fpdf) TransformSkewY(angleY, x, y float64)
```

TransformSkewY vertically skews the following text, drawings and images keeping the point (x, y) stationary. angleY ranges from -90 degrees (skew to the bottom) to 90 degrees (skew to the top).

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformTranslate Â¶
```

```
func (f *Fpdf) TransformTranslate(tx, ty float64)
```

TransformTranslate moves the following text, drawings and images horizontally and vertically by the amounts specified by tx and ty.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformTranslateX Â¶
```

```
func (f *Fpdf) TransformTranslateX(tx float64)
```

TransformTranslateX moves the following text, drawings and images horizontally by the amount specified by tx.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) TransformTranslateY Â¶
```

```
func (f *Fpdf) TransformTranslateY(ty float64)
```

TransformTranslateY moves the following text, drawings and images vertically by the amount specified by ty.

The TransformBegin() example demonstrates this method.

```
func (*Fpdf) UnicodeTranslatorFromDescriptor Â¶
```

```
func (f *Fpdf) UnicodeTranslatorFromDescriptor(cpStr string) (rep func(string) string)
```

UnicodeTranslatorFromDescriptor returns a function that can be used to translate, where possible, utf-8 strings to a form that is compatible with the specified code page. See UnicodeTranslator for more details.

cpStr identifies a code page. A descriptor file in the font directory, set with the fontDirStr argument in the call to New(), should have this name plus the extension ".map". If cpStr is empty, it will be replaced with "cp1252", the gofpdf code page default.

If an error occurs reading the descriptor, the returned function is valid but does not perform any rune translation.

The CellFormat_codepage example demonstrates this method.

```
func (*Fpdf) UnitToPointConvert Â¶
```

```
func (f *Fpdf) UnitToPointConvert(u float64) (pt float64)
```

UnitToPointConvert returns the value of u, expressed in the unit of measure specified in New(), as a value expressed in points (1/72 inch). Since font management in Fpdf uses points, this method can help with setting font sizes based on the sizes of other non-font page elements.

func (*Fpdf) UseImportedTemplate Â¶

added in v1.4.1

func (f *Fpdf) UseImportedTemplate(tplName string, scaleX float64, scaleY float64, tX float64, tY float64)

UseImportedTemplate uses imported template from gofpdi. It draws imported PDF page onto page.

func (*Fpdf) UseTemplate Â¶

func (f *Fpdf) UseTemplate(t Template)

UseTemplate adds a template to the current page or another template, using the size and position at which it was originally written.

func (*Fpdf) UseTemplateScaled Â¶

func (f *Fpdf) UseTemplateScaled(t Template, corner PointType, size SizeType)

UseTemplateScaled adds a template to the current page or another template, using the given page coordinates.

func (*Fpdf) Write Â¶

func (f *Fpdf) Write(h float64, txtStr string)

Write prints text from the current position. When the right margin is reached (or the \n character is met) a line break occurs and text continues from the left margin. Upon method exit, the current position is left just at the end of the text.

It is possible to put a link on the text.

h indicates the line height in the unit of measure specified in New().

func (*Fpdf) WriteAligned Â¶

func (f *Fpdf) WriteAligned(width, lineHeight float64, textStr, alignStr string)

WriteAligned is an implementation of Write that makes it possible to align text.

width indicates the width of the box the text will be drawn in. This is in the unit of measure specified in New(). If it is set to 0, the bounding box of the page will be taken (pageWidth - leftMargin - rightMargin).

lineHeight indicates the line height in the unit of measure specified in New().

alignStr sees to horizontal alignment of the given textStr. The options are "L", "C" and "R" (Left, Center, Right). The default is "L".

Example Â¶

func (*Fpdf) WriteLinkID Â¶

func (f *Fpdf) WriteLinkID(h float64, displayStr string, linkID int)

WriteLinkID writes text that when clicked jumps to another location in the PDF. linkID is an identifier returned by AddLink(). See Write() for argument details.

func (*Fpdf) WriteLinkString Â¶

func (f *Fpdf) WriteLinkString(h float64, displayStr, targetStr string)

WriteLinkString writes text that when clicked launches an external URL. See Write() for argument details.

func (*Fpdf) Writeln Â¶

func (f *Fpdf) Writeln(h float64, fmtStr string, args ...interface{ })

Writeln is like Write but uses printf-style formatting. See the documentation for package fmt for more details on fmtStr and args.

type FpdfTpl Â¶

type FpdfTpl struct {

```
// contains filtered or unexported fields
}
```

FpdfTpl is a concrete implementation of the Template interface.

```
func (*FpdfTpl) Bytes() []byte
```

```
func (t *FpdfTpl) Bytes() []byte
```

Bytes returns the actual template data, not including resources

```
func (*FpdfTpl) FromPage() []byte
```

added in v1.0.1

```
func (t *FpdfTpl) FromPage(page int) (Template, error)
```

FromPage creates a new template from a specific Page

```
func (*FpdfTpl) FromPages() []Template
```

added in v1.0.1

```
func (t *FpdfTpl) FromPages() []Template
```

FromPages creates a template slice with all the pages within a template.

```
func (*FpdfTpl) GobDecode() []byte
```

added in v1.0.1

```
func (t *FpdfTpl) GobDecode(buf []byte) error
```

GobDecode decodes the specified byte buffer into the receiving template.

```
func (*FpdfTpl) GobEncode() []byte
```

added in v1.0.1

```
func (t *FpdfTpl) GobEncode() ([]byte, error)
```

GobEncode encodes the receiving template into a byte buffer. Use GobDecode to decode the byte buffer back to a template.

```
func (*FpdfTpl) ID() string
```

```
func (t *FpdfTpl) ID() string
```

ID returns the global template identifier

```
func (*FpdfTpl) Images() map[string]*ImageInfoType
```

```
func (t *FpdfTpl) Images() map[string]*ImageInfoType
```

Images returns a list of the images used in this template

```
func (*FpdfTpl) NumPages() int
```

added in v1.0.1

```
func (t *FpdfTpl) NumPages() int
```

NumPages returns the number of available pages within the template. Look at FromPage and FromPages on access to that content.

```
func (*FpdfTpl) Serialize() []byte
```

added in v1.0.1

```
func (t *FpdfTpl) Serialize() ([]byte, error)
```

Serialize turns a template into a byte string for later deserialization

```
func (*FpdfTpl) Size() PointType
```

```
func (t *FpdfTpl) Size() (corner PointType, size SizeType)
```

Size gives the bounding dimensions of this template

```
func (*FpdfTpl) Templates() []Template
```

func (t *FpdfTpl) Templates() []Template
Templates returns a list of templates used in this template

type GridType ¶
added in v1.0.1

type GridType struct {

// Labels are inside of graph boundary
XLabelIn, YLabelIn bool
// Labels on X-axis should be rotated
XLabelRotate bool
// Formatters; use nil to eliminate labels
XTickStr, YTickStr TickFormatFncType
// Subdivisions between tickmarks
XDiv, YDiv int

// Line and label colors
ClrText, ClrMain, ClrSub RGBAType
// Line thickness
WdMain, WdSub float64
// Label height in points
TextSize float64
// contains filtered or unexported fields
}

GridType assists with the generation of graphs. It allows the application to work with logical data coordinates rather than page coordinates and assists with the drawing of a background grid.

func NewGrid ¶
added in v1.0.1

func NewGrid(x, y, w, h float64) (grid GridType)

NewGrid returns a variable of type GridType that is initialized to draw on a rectangle of width w and height h with the upper left corner positioned at point (x, y). The coordinates are in page units, that is, the same as those specified in New().

The returned variable is initialized with a very simple default tickmark layout that ranges from 0 to 1 in both axes. Prior to calling Grid(), the application may establish a more suitable tickmark layout by calling the methods TickmarksContainX() and TickmarksContainY(). These methods bound the data range with appropriate boundaries and divisions. Alternatively, if the exact extent and divisions of the tickmark layout are known, the methods TickmarksExtentX() and TickmarksExtentY may be called instead.

Example ¶

func (GridType) Grid ¶
added in v1.0.1

func (g GridType) Grid(pdf *Fpdf)

Grid generates a graph-paperlike set of grid lines on the current page.

func (GridType) Ht ¶
added in v1.0.1

func (g GridType) Ht(dataHt float64) float64

Ht converts dataHt, specified in logical data units, to the unit of measure specified in New().

func (GridType) HtAbs ¶
added in v1.0.1

func (g GridType) HtAbs(dataHt float64) float64

HtAbs returns the absolute value of dataHt, specified in logical data units, that has been converted to the unit of measure specified in New().

func (GridType) Plot \hat{A}

added in v1.0.1

func (g GridType) Plot(pdf *Fpdf, xMin, xMax float64, count int, fnc func(x float64) (y float64))

Plot plots a series of count line segments from xMin to xMax. It repeatedly calls fnc(x) to retrieve the y value associate with x. The currently selected line drawing attributes are used.

func (GridType) Pos \hat{A}

added in v1.0.1

func (g GridType) Pos(xRel, yRel float64) (x, y float64)

Pos returns the point, in page units, indicated by the relative positions xRel and yRel. These are values between 0 and 1. xRel specifies the relative position between the grid's left and right edges. yRel specifies the relative position between the grid's bottom and top edges.

func (*GridType) TickmarksContainX \hat{A}

added in v1.0.1

func (g *GridType) TickmarksContainX(min, max float64)

TickmarksContainX sets the tickmarks to be shown by Grid() in the horizontal dimension. The argument min and max specify the minimum and maximum values to be contained within the grid. The tickmark values that are generated are suitable for general purpose graphs.

See TickmarkExtentX() for an alternative to this method to be used when the exact values of the tickmarks are to be set by the application.

func (*GridType) TickmarksContainY \hat{A}

added in v1.0.1

func (g *GridType) TickmarksContainY(min, max float64)

TickmarksContainY sets the tickmarks to be shown by Grid() in the vertical dimension. The argument min and max specify the minimum and maximum values to be contained within the grid. The tickmark values that are generated are suitable for general purpose graphs.

See TickmarkExtentY() for an alternative to this method to be used when the exact values of the tickmarks are to be set by the application.

func (*GridType) TickmarksExtentX \hat{A}

added in v1.0.1

func (g *GridType) TickmarksExtentX(min, div float64, count int)

TickmarksExtentX sets the tickmarks to be shown by Grid() in the horizontal dimension. count specifies number of major tickmark subdivisions to be graphed. min specifies the leftmost data value. div specifies, in data units, the extent of each major tickmark subdivision.

See TickmarkContainX() for an alternative to this method to be used when viewer-friendly tickmarks are to be determined automatically.

func (*GridType) TickmarksExtentY \hat{A}

added in v1.0.1

func (g *GridType) TickmarksExtentY(min, div float64, count int)

TickmarksExtentY sets the tickmarks to be shown by Grid() in the vertical dimension. count specifies number of major tickmark subdivisions to be graphed. min specifies the bottommost data value. div specifies, in data units, the extent of each major tickmark subdivision.

See `TickmarkContainY()` for an alternative to this method to be used when viewer-friendly tickmarks are to be determined automatically.

```
func (GridType) Wd Â¶  
added in v1.0.1
```

```
func (g GridType) Wd(dataWd float64) float64
```

`Wd` converts `dataWd`, specified in logical data units, to the unit of measure specified in `New()`.

```
func (GridType) WdAbs Â¶  
added in v1.0.1
```

```
func (g GridType) WdAbs(dataWd float64) float64
```

`WdAbs` returns the absolute value of `dataWd`, specified in logical data units, that has been converted to the unit of measure specified in `New()`.

```
func (GridType) X Â¶  
added in v1.0.1
```

```
func (g GridType) X(dataX float64) float64
```

`X` converts `dataX`, specified in logical data units, to the `X` position on the current page.

```
func (GridType) XRange Â¶  
added in v1.0.1
```

```
func (g GridType) XRange() (min, max float64)
```

`XRange` returns the minimum and maximum values for the current tickmark sequence. These correspond to the data values of the graph's left and right edges.

```
func (GridType) XY Â¶  
added in v1.0.1
```

```
func (g GridType) XY(dataX, dataY float64) (x, y float64)
```

`XY` converts `dataX` and `dataY`, specified in logical data units, to the `X` and `Y` position on the current page.

```
func (GridType) Y Â¶  
added in v1.0.1
```

```
func (g GridType) Y(dataY float64) float64
```

`Y` converts `dataY`, specified in logical data units, to the `Y` position on the current page.

```
func (GridType) YRange Â¶  
added in v1.0.1
```

```
func (g GridType) YRange() (min, max float64)
```

`YRange` returns the minimum and maximum values for the current tickmark sequence. These correspond to the data values of the graph's bottom and top edges.

```
type HTMLBasicSegmentType Â¶
```

```
type HTMLBasicSegmentType struct {
```

```
    Cat byte          // 'O' open tag, 'C' close tag, 'T' text
```

```
    Str string        // Literal text unchanged, tags are lower case
```

```
    Attr map[string]string // Attribute keys are lower case
```

```
}
```

`HTMLBasicSegmentType` defines a segment of literal text in which the current attributes do not vary, or an open tag or a close tag.

```
func HTMLBasicTokenize Â¶
```

```
func HTMLBasicTokenize(htmlStr string) (list []HTMLBasicSegmentType)
```


HTMLBasicTokenize returns a list of HTML tags and literal elements. This is done with regular expressions, so the result is only marginally better than useless.

```
type HTMLBasicType Â¶
type HTMLBasicType struct {
    Link struct {
        ClrR, ClrG, ClrB      int
        Bold, Italic, Underscore bool
    }
    // contains filtered or unexported fields
}
```

HTMLBasicType is used for rendering a very basic subset of HTML. It supports only hyperlinks and bold, italic and underscore attributes. In the Link structure, the ClrR, ClrG and ClrB fields (0 through 255) define the color of hyperlinks. The Bold, Italic and Underscore values define the hyperlink style.

```
func (*HTMLBasicType) Write Â¶
func (html *HTMLBasicType) Write(lineHt float64, htmlStr string)
Write prints text from the current position using the currently selected font. See HTMLBasicNew() to create a receiver that is associated with the PDF document instance. The text can be encoded with a basic subset of HTML that includes hyperlinks and tags for italic (I), bold (B), underscore (U) and center (CENTER) attributes. When the right margin is reached a line break occurs and text continues from the left margin. Upon method exit, the current position is left at the end of the text.
```

lineHt indicates the line height in the unit of measure specified in New().

```
type ImageInfoType Â¶
type ImageInfoType struct {
    // contains filtered or unexported fields
}
```

ImageInfoType contains size, color and other information about an image. Changes to this structure should be reflected in its GobEncode and GobDecode methods.

```
func (*ImageInfoType) Extent Â¶
func (info *ImageInfoType) Extent() (wd, ht float64)
Extent returns the width and height of the image in the units of the Fpdf object.
```

```
func (*ImageInfoType) GobDecode Â¶
added in v1.0.1
func (info *ImageInfoType) GobDecode(buf []byte) (err error)
GobDecode decodes the specified byte buffer (generated by GobEncode) into the receiving image.
```

```
func (*ImageInfoType) GobEncode Â¶
added in v1.0.1
func (info *ImageInfoType) GobEncode() (buf []byte, err error)
GobEncode encodes the receiving image to a byte slice.
```

```
func (*ImageInfoType) Height Â¶
func (info *ImageInfoType) Height() float64
Height returns the height of the image in the units of the Fpdf object.
```

```
func (*ImageInfoType) SetDpi Â¶
func (info *ImageInfoType) SetDpi(dpi float64)
SetDpi sets the dots per inch for an image. PNG images MAY have their dpi set automatically, if the image
```

specifies it. DPI information is not currently available automatically for JPG and GIF images, so if it's important to you, you can set it here. It defaults to 72 dpi.

```
func (*ImageInfoType) Width() float64
```

```
func (info *ImageInfoType) Width() float64
```

Width returns the width of the image in the units of the Fpdf object.

```
type ImageOptions struct {
```

```
    ImageType      string
```

```
    ReadDpi        bool
```

```
    AllowNegativePosition bool
```

```
}
```

ImageOptions provides a place to hang any options we want to use while parsing an image.

ImageType's possible values are (case insensitive): "JPG", "JPEG", "PNG" and "GIF". If empty, the type is inferred from the file extension.

ReadDpi defines whether to attempt to automatically read the image dpi information from the image file. Normally, this should be set to true (understanding that not all images will have this info available). However, for backwards compatibility with previous versions of the API, it defaults to false.

AllowNegativePosition can be set to true in order to prevent the default coercion of negative x values to the current x position.

```
type InitType struct {
```

```
    OrientationStr string
```

```
    UnitStr      string
```

```
    SizeStr      string
```

```
    Size         SizeType
```

```
    FontDirStr   string
```

```
}
```

InitType is used with NewCustom() to customize an Fpdf instance. OrientationStr, UnitStr, SizeStr and FontDirStr correspond to the arguments accepted by New(). If the Wd and Ht fields of Size are each greater than zero, Size will be used to set the default page size rather than SizeStr. Wd and Ht are specified in the units of measure indicated by UnitStr.

```
type PageBox struct {
```

```
    SizeType
```

```
    PointType
```

```
}
```

```
}
```

PageBox defines the coordinates and extent of the various page box types

```
type Pdf interface {
```

```
    AddFont(familyStr, styleStr, fileStr string)
```

```
    AddFontFromBytes(familyStr, styleStr string, jsonFileBytes, zFileBytes []byte)
```

```
    AddFontFromReader(familyStr, styleStr string, r io.Reader)
```

```
    AddLayer(name string, visible bool) (layerID int)
```

AddLink() int
AddPage()
AddPageFormat(orientationStr string, size SizeType)
AddSpotColor(nameStr string, c, m, y, k byte)
AliasNbPages(aliasStr string)
ArcTo(x, y, rx, ry, degRotate, degStart, degEnd float64)
Arc(x, y, rx, ry, degRotate, degStart, degEnd float64, styleStr string)
BeginLayer(id int)
Beziergon(points []PointType, styleStr string)
Bookmark(txtStr string, level int, y float64)
CellFormat(w, h float64, txtStr, borderStr string, ln int, alignStr string, fill bool, link int, linkStr string)
Cellf(w, h float64, fmtStr string, args ...interface{ })
Cell(w, h float64, txtStr string)
Circle(x, y, r float64, styleStr string)
ClearError()
ClipCircle(x, y, r float64, outline bool)
ClipEllipse(x, y, rx, ry float64, outline bool)
ClipEnd()
ClipPolygon(points []PointType, outline bool)
ClipRect(x, y, w, h float64, outline bool)
ClipRoundedRect(x, y, w, h, r float64, outline bool)
ClipText(x, y float64, txtStr string, outline bool)
Close()
ClosePath()
CreateTemplateCustom(corner PointType, size SizeType, fn func(*Tpl)) Template
CreateTemplate(fn func(*Tpl)) Template
CurveBezierCubicTo(cx0, cy0, cx1, cy1, x, y float64)
CurveBezierCubic(x0, y0, cx0, cy0, cx1, cy1, x1, y1 float64, styleStr string)
CurveCubic(x0, y0, cx0, cy0, x1, y1, cx1, cy1 float64, styleStr string)
CurveTo(cx, cy, x, y float64)
Curve(x0, y0, cx, cy, x1, y1 float64, styleStr string)
DrawPath(styleStr string)
Ellipse(x, y, rx, ry, degRotate float64, styleStr string)
EndLayer()
Err() bool
Error() error
GetAlpha() (alpha float64, blendModeStr string)
GetAutoPageBreak() (auto bool, margin float64)
GetCellMargin() float64
GetConversionRatio() float64
GetDrawColor() (int, int, int)
GetDrawSpotColor() (name string, c, m, y, k byte)
GetFillColor() (int, int, int)
GetFillSpotColor() (name string, c, m, y, k byte)
GetFontDesc(familyStr, styleStr string) FontDescType
GetFontSize() (ptSize, unitSize float64)
GetImageInfo(imageStr string) (info *ImageInfoType)
GetLineWidth() float64
GetMargins() (left, top, right, bottom float64)
GetPageSizeStr(sizeStr string) (size SizeType)
GetPageSize() (width, height float64)
GetStringWidth(s string) float64
GetTextColor() (int, int, int)

GetTextSpotColor() (name string, c, m, y, k byte)
GetX() float64
GetXY() (float64, float64)
GetY() float64
HTMLBasicNew() (html HTMLBasicType)
Image(imageNameStr string, x, y, w, h float64, flow bool, tp string, link int, linkStr string)
ImageOptions(imageNameStr string, x, y, w, h float64, flow bool, options ImageOptions, link int, linkStr string)
ImageTypeFromMime(mimeStr string) (tp string)
LinearGradient(x, y, w, h float64, r1, g1, b1, r2, g2, b2 int, x1, y1, x2, y2 float64)
LineTo(x, y float64)
Line(x1, y1, x2, y2 float64)
LinkString(x, y, w, h float64, linkStr string)
Link(x, y, w, h float64, link int)
Ln(h float64)
MoveTo(x, y float64)
MultiCell(w, h float64, txtStr, borderStr, alignStr string, fill bool)
Ok() bool
OpenLayerPane()
OutputAndClose(w io.WriteCloser) error
OutputFileAndClose(fileStr string) error
Output(w io.Writer) error
PageCount() int
PageNo() int
PageSize(pageNum int) (wd, ht float64, unitStr string)
PointConvert(pt float64) (u float64)
PointToUnitConvert(pt float64) (u float64)
Polygon(points []PointType, styleStr string)
RadialGradient(x, y, w, h float64, r1, g1, b1, r2, g2, b2 int, x1, y1, x2, y2, r float64)
RawWriteBuf(r io.Reader)
RawWriteStr(str string)
Rect(x, y, w, h float64, styleStr string)
RegisterAlias(alias, replacement string)
RegisterImage(fileStr, tp string) (info *ImageInfoType)
RegisterImageOptions(fileStr string, options ImageOptions) (info *ImageInfoType)
RegisterImageOptionsReader(imgName string, options ImageOptions, r io.Reader) (info *ImageInfoType)
RegisterImageReader(imgName, tp string, r io.Reader) (info *ImageInfoType)
SetAcceptPageBreakFunc(fnc func() bool)
SetAlpha(alpha float64, blendModeStr string)
SetAuthor(authorStr string, isUTF8 bool)
SetAutoPageBreak(auto bool, margin float64)
SetCatalogSort(flag bool)
SetCellMargin(margin float64)
SetCompression(compress bool)
SetCreationDate(tm time.Time)
SetCreator(creatorStr string, isUTF8 bool)
SetDashPattern(dashArray []float64, dashPhase float64)
SetDisplayMode(zoomStr, layoutStr string)
SetDrawColor(r, g, b int)
SetDrawSpotColor(nameStr string, tint byte)
SetError(err error)
SetErrorf(fmtStr string, args ...interface{ })
SetFillColor(r, g, b int)

SetFillSpotColor(nameStr string, tint byte)
SetFont(familyStr, styleStr string, size float64)
SetFontLoader(loader FontLoader)
SetFontLocation(fontDirStr string)
SetFontSize(size float64)
SetFontStyle(styleStr string)
SetFontUnitSize(size float64)
SetFooterFunc(fnc func())
SetFooterFuncLpi(fnc func(lastPage bool))
SetHeaderFunc(fnc func())
SetHeaderFuncMode(fnc func(), homeMode bool)
SetHomeXY()
SetJavascript(script string)
SetKeywords(keywordsStr string, isUTF8 bool)
SetLeftMargin(margin float64)
SetLineCapStyle(styleStr string)
SetLineJoinStyle(styleStr string)
SetLineWidth(width float64)
SetLink(link int, y float64, page int)
SetMargins(left, top, right float64)
SetPageBoxRec(t string, pb PageBox)
SetPageBox(t string, x, y, wd, ht float64)
SetPage(pageNum int)
SetProtection(actionFlag byte, userPassStr, ownerPassStr string)
SetRightMargin(margin float64)
SetSubject(subjectStr string, isUTF8 bool)
SetTextColor(r, g, b int)
SetTextSpotColor(nameStr string, tint byte)
SetTitle(titleStr string, isUTF8 bool)
SetTopMargin(margin float64)
SetUnderlineThickness(thickness float64)
SetXmpMetadata(xmpStream []byte)
SetX(x float64)
SetXY(x, y float64)
SetY(y float64)
SplitLines(txt []byte, w float64) [][]byte
String() string
SVGBasicWrite(sb *SVGBasicType, scale float64)
Text(x, y float64, txtStr string)
TransformBegin()
TransformEnd()
TransformMirrorHorizontal(x float64)
TransformMirrorLine(angle, x, y float64)
TransformMirrorPoint(x, y float64)
TransformMirrorVertical(y float64)
TransformRotate(angle, x, y float64)
TransformScale(scaleWd, scaleHt, x, y float64)
TransformScaleX(scaleWd, x, y float64)
TransformScaleXY(s, x, y float64)
TransformScaleY(scaleHt, x, y float64)
TransformSkew(angleX, angleY, x, y float64)
TransformSkewX(angleX, x, y float64)
TransformSkewY(angleY, x, y float64)

```

Transform(tm TransformMatrix)
TransformTranslate(tx, ty float64)
TransformTranslateX(tx float64)
TransformTranslateY(ty float64)
UnicodeTranslatorFromDescriptor(cpStr string) (rep func(string) string)
UnitToPointConvert(u float64) (pt float64)
UseTemplateScaled(t Template, corner PointType, size SizeType)
UseTemplate(t Template)
WriteAligned(width, lineHeight float64, textStr, alignStr string)
Writeln(h float64, fmtStr string, args ...interface{ })
Write(h float64, txtStr string)
WriteLinkID(h float64, displayStr string, linkID int)
WriteLinkString(h float64, displayStr, targetStr string)
}

```

Pdf defines the interface used for various methods. It is implemented by the main FPDF instance as well as templates.

```

type PointType ¶
type PointType struct {
    X, Y float64
}

```

PointType fields X and Y specify the horizontal and vertical coordinates of a point, typically used in drawing.

```

func (*PointType) Transform ¶
func (p *PointType) Transform(x, y float64) PointType
Transform moves a point by given X, Y offset

```

```

func (PointType) XY ¶
func (p PointType) XY() (float64, float64)
XY returns the X and Y components of the receiver point.

```

```

type RGBAType ¶
added in v1.0.1
type RGBAType struct {
    R, G, B int
    Alpha float64
}

```

RGBAType holds fields for red, green and blue color components (0..255) and an alpha transparency value (0..1)

```

type RGBType ¶
added in v1.0.1
type RGBType struct {
    R, G, B int
}

```

RGBType holds fields for red, green and blue color components (0..255)

```

type SVGBasicSegmentType ¶
type SVGBasicSegmentType struct {
    Cmd byte // See http://www.w3.org/TR/SVG/paths.html for path command structure
    Arg [6]float64
}

```

SVGBasicSegmentType describes a single curve or position segment

```
type SVGBasicType Â¶
type SVGBasicType struct {
```

```
    Wd, Ht float64
    Segments [][]SVGBasicSegmentType
}
```

SVGBasicType aggregates the information needed to describe a multi-segment basic vector image

```
func SVGBasicFileParse Â¶
```

```
func SVGBasicFileParse(svgFileStr string) (sig SVGBasicType, err error)
```

SVGBasicFileParse parses a simple scalable vector graphics (SVG) file into a basic descriptor. The SVGBasicWrite() example demonstrates this method.

```
func SVGBasicParse Â¶
```

```
func SVGBasicParse(buf []byte) (sig SVGBasicType, err error)
```

SVGBasicParse parses a simple scalable vector graphics (SVG) buffer into a descriptor. Only a small subset of the SVG standard, in particular the path information generated by jSignature, is supported. The returned path data includes only the commands 'M' (absolute moveto: x, y), 'L' (absolute lineto: x, y), 'C' (absolute cubic BÃ©zier curve: cx0, cy0, cx1, cy1, x1,y1), 'Q' (absolute quadratic BÃ©zier curve: x0, y0, x1, y1) and 'Z' (closepath).

```
type SizeType Â¶
```

```
type SizeType struct {
    Wd, Ht float64
}
```

SizeType fields Wd and Ht specify the horizontal and vertical extents of a document element such as a page.

```
func (*SizeType) Orientation Â¶
```

```
func (s *SizeType) Orientation() string
```

Orientation returns the orientation of a given size: "P" for portrait, "L" for landscape

```
func (*SizeType) ScaleBy Â¶
```

```
func (s *SizeType) ScaleBy(factor float64) SizeType
```

ScaleBy expands a size by a certain factor

```
func (*SizeType) ScaleToHeight Â¶
```

```
func (s *SizeType) ScaleToHeight(height float64) SizeType
```

ScaleToHeight adjusts the width of a size to match the given height

```
func (*SizeType) ScaleToWidth Â¶
```

```
func (s *SizeType) ScaleToWidth(width float64) SizeType
```

ScaleToWidth adjusts the height of a size to match the given width

```
type StateType Â¶
```

added in v1.0.1

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

StateType holds various commonly used drawing values for convenient retrieval (StateGet()) and restore (Put) methods.

```
func StateGet Â¶
```

added in v1.0.1

```
func StateGet(pdf *Fpdf) (st StateType)
```

StateGet returns a variable that contains common state values.

```
func (StateType) Put Â¶
```

added in v1.0.1

```
func (st StateType) Put(pdf *Fpdf)
```

Put sets the common state values contained in the state structure specified by st.

```
type Template Â¶
```

```
type Template interface {
```

```
    ID() string
```

```
    Size() (PointType, SizeType)
```

```
    Bytes() []byte
```

```
    Images() map[string]*ImageInfoType
```

```
    Templates() []Template
```

```
    NumPages() int
```

```
    FromPage(int) (Template, error)
```

```
    FromPages() []Template
```

```
    Serialize() ([]byte, error)
```

```
    gob.GobDecoder
```

```
    gob.GobEncoder
```

```
}
```

Template is an object that can be written to, then used and re-used any number of times within a document.

```
func CreateTemplate Â¶
```

```
func CreateTemplate(corner PointType, size SizeType, unitStr, fontDirStr string, fn func(*Tpl)) Template
```

CreateTemplate creates a template that is not attached to any document.

This function is deprecated; it incorrectly assumes that a page with a width smaller than its height is oriented in portrait mode, otherwise it assumes landscape mode. This causes problems when placing the template in a master document where this condition does not apply. CreateTpl() is a similar function that lets you specify the orientation to avoid this problem.

```
func CreateTpl Â¶
```

added in v1.0.1

```
func CreateTpl(corner PointType, size SizeType, orientationStr, unitStr, fontDirStr string, fn func(*Tpl))  
Template
```

CreateTpl creates a template not attached to any document

```
func DeserializeTemplate Â¶
```

added in v1.0.1

```
func DeserializeTemplate(b []byte) (Template, error)
```

DeserializeTemplate creates a template from a previously serialized template

```
type TickFormatFncType Â¶
```

added in v1.0.1

```
type TickFormatFncType func(val float64, precision int) string
```

TickFormatFncType defines a callback for label drawing.

```
type Tpl Â¶
```

```
type Tpl struct {
```

```
    Fpdf
```

```
}
```


Tpl is an Fpdf used for writing a template. It has most of the facilities of an Fpdf, but cannot add more pages. Tpl is used directly only during the limited time a template is writable.

```
type TransformMatrix Â¶
type TransformMatrix struct {
    A, B, C, D, E, F float64
}
```

TransformMatrix is used for generalized transformations of text, drawings and images.

```
type TtfType Â¶
type TtfType struct {
    Embeddable      bool
    UnitsPerEm      uint16
    PostScriptName   string
    Bold            bool
    ItalicAngle      int16
    IsFixedPitch     bool
    TypoAscender     int16
    TypoDescender    int16
    UnderlinePosition int16
    UnderlineThickness int16
    Xmin, Ymin, Xmax, Ymax int16
    CapHeight        int16
    Widths            []uint16
    Chars            map[uint16]uint16
}
```

TtfType contains metrics of a TrueType font.

```
func TtfParse Â¶
func TtfParse(fileStr string) (TtfRec TtfType, err error)
TtfParse extracts various metrics from a TrueType font file.
```

Example Â¶

Source Files Â¶

View all Source files

attachments.go

compare.go

def.go

doc.go

embedded.go

font.go

fpdf.go

fpdftrans.go

grid.go

htmlbasic.go

label.go

layer.go

png.go

protect.go

splittext.go

spotcolor.go

subwrite.go

svgbasic.go

svgwrite.go
template.go
template_impl.go
ttfparser.go
utf8fontfile.go
util.go
Directories Â¶
Show internal
Expand all
contrib
list
makefont
Command makefont generates a font definition file.
Why Go
Use Cases
Case Studies
Get Started
Playground
Tour
Stack Overflow
Help
Packages
Standard Library
Sub-repositories
About Go Packages
About
Download
Blog
Issue Tracker
Release Notes
Brand Guidelines
Code of Conduct
Connect
Twitter
GitHub
Slack
r/golang
Meetup
Golang Weekly
Gopher in flight goggles
Copyright
Terms of Service
Privacy Policy
Report an Issue
System theme
Theme Toggle

Shortcuts Modal

Google logo