
pyxpdf

Release 0.2.3

Ashutosh Varma

Aug 31, 2020

CONTENTS:

1	Getting started	3
1.1	Installation	3
1.2	Quick Start	3
2	Tutorials	5
2.1	Extract text from pdf while maintaining layout	5
2.2	Extract images from pdf file	6
3	API Reference	7
3.1	Document and Page	7
3.2	Output Devices	14
3.3	Global Config	20
3.4	Exceptions	34
4	Speed Comparision	35
4.1	Text Extraction	35
5	Changelog	37
5.1	0.2.3 (2020-08-31)	37
5.2	0.2.2 (2020-07-03)	37
5.3	0.2.1 (2020-06-12)	37
5.4	0.2.0 (2020-06-11)	38
5.5	0.1.1 (2020-05-10)	38
5.6	0.1 (2020-04-20)	38
6	Glossary	39
7	Indices and tables	41
	Index	43

pyxpdf is a fast and memory efficient python module for parsing PDF documents based on xpdf reader sources.
Find the source on [Github](#).

GETTING STARTED

pyxpdf is a wrapper on [xpdf reader](#) sources.

It aims to provide a fast and memory efficient pdf parser with easy to use API.

1.1 Installation

```
pip install pyxpdf
```

For additional encodings support, install optional dependency [pyxpdf_data](#)

```
pip install pyxpdf_data
```

For Image extraction and pdf to image support, install optional dependency [Pillow](#)

```
pip install Pillow
```

1.2 Quick Start

pyxpdf use *Document* to represent and load a PDF file. Similary *Page* for PDF Page.

All the xpdf related settings can be accessed with global *Config* object.

```
from pyxpdf import Document, Page, Config
from pyxpdf.xpdf import TextControl

doc = Document("samples/nonfree/mandarin.pdf")
# or
# load pdf from file like object
with open("samples/nonfree/mandarin.pdf", 'rb') as fp:
    doc = Document(fp)

# get pdf metadata dict
print(doc.info())
# >>> doc.info()
# {'CreationDate': "D:20080721141207-04'00'",
#  'Subject': 'Chinese Version of Universal PCXR8 ...',
#  'Author': 'SKC Inc.',
#  'Creator': 'PScript5.dll
#  ....
```

(continues on next page)

(continued from previous page)

```
# get all text
all_text = doc.text()

# iter first 10 pages
for page in doc[:10]:
    # get page label if any
    print(page.label)

# get page by page label
label_page = doc['1']

# get text in table layout without discarding clipped
# text.
text_control = TextControl("table", discard_clipped=True)
text = label_page.text(control=text_control)

# find case sensitive text within [x_min, y_min, x_max, y_max]
res_box = label_page.find_text(' ', search_box=[0, 0, 400, 400],
                               case_sensitive=True)

# >>> print(res_box)
# (281.88, 269.718, 354.05819999999994, 287.7)

# load xpdfrc
Config.load_file('my_xpdfrc')
# suppress stderr output for xpdf error log.
Config.error_quiet = False
```

Checkout [API Reference](#) for more details.

Todo: Add bechmark and speed comparison with python pdf modules

TUTORIALS

Tutorials can help you get started with pyxpdf.

Before starting tutorial first test the pyxpdf installation by importing it in Python.

```
>>> import pyxpdf
>>> print(pyxpdf.__version__)
<installed_version>
```

2.1 Extract text from pdf while maintaining layout

1. Using `text()` method for *Document* and *Page*:

```
from pyxpdf import Document
from pyxpdf.xpdf import TextOutput, TextControl

# http://www.kurims.kyoto-u.ac.jp/~terui/pssj.pdf
# Install 'pyxpdf_data', needed for additional encodings (japanese)
doc = Document("pssj.pdf")

control = TextControl(mode = "physical")
for page in doc:
    txt = page.text(control=control)
    print(txt)
```

2. Using *TextOutput*:

```
from pyxpdf import Document
from pyxpdf.xpdf import TextOutput, TextControl, page_iterator

# http://www.kurims.kyoto-u.ac.jp/~terui/pssj.pdf
# Install 'pyxpdf_data', needed for additional encodings (japanese)
doc = Document("pssj.pdf")

control = TextControl(mode = "physical")
text_out = TextOutput(doc, control)

for pg_txt in page_iterator(text_out):
    print(pg_txt)
```

2.2 Extract images from pdf file

`PDFImageOutput` can extract *LZW*, *Run Length*, *CCITTFax*, *DCT*, *JBIG2*, *JPX* compressed images and image masks.

```
from pyxpdf import Document
from pyxpdf.xpdf import PDFImageOutput, page_iterator

# https://www.mlcjapanese.co.jp/Download/HiraganaKatakanaWorksheet.pdf
doc = Document("HiraganaKatakanaWorksheet.pdf")
pdfimages_out = PDFImageOutput(doc)

for images in page_iterator(pdfimages_out):
    print(images)
```

2.2.1 Output

```
[<pyxpdf.xpdf.PDFImage type=image compression=flate colorspace=icc bbox=(239.4, 539.
→45, 286.4, 588.45)>, <pyxpdf.xpdf.PDFImage type=image compression=flate_
→colorspace=icc bbox=(147.35000000000002, 612.9459999999999, 163.35000000000002, 650.
→9459999999999)>]
[<pyxpdf.xpdf.PDFImage type=image compression=flate colorspace=icc bbox=(292.75, 27.
→14999999999993, 304.75, 57.14999999999935)>, <pyxpdf.xpdf.PDFImage type=image_
→compression=flate colorspace=icc bbox=(152.25, 179.2499999999994, 252.25, 393.
→2499999999994)>]
[<pyxpdf.xpdf.PDFImage type=image compression=flate colorspace=icc bbox=(292.75, 27.
→14999999999993, 304.75, 57.14999999999935)>, <pyxpdf.xpdf.PDFImage type=image_
→compression=jpeg colorspace=icc bbox=(174.85000000000002, 263.99, 188.
→85000000000002, 286.99)>, <pyxpdf.xpdf.PDFImage type=image compression=jpeg_
→colorspace=icc bbox=(222.3, 264.59, 236.3, 287.59)>]
...
```

API REFERENCE

3.1 Document and Page

3.1.1 Document

class `pyxpdf.xpdf.Document` (*pdf*, *ownerpass=None*, *userpass=None*)
This class represents a PDF Document.

Page objects can be accessed though indexing, slicing.

If *pdf* parameter is a *file-like* object then make sure that it is open in ‘b’ binary mode as *Document* does not check for it.

Examples

```
>>> doc = Document("~/sample.pdf")
```

Total pages in Document

```
>>> len(doc)
17
```

Access page by index

```
>>> page1 = doc[0]
<Page[0]>
```

Access page by label

```
>>> cover_page = doc['Cover1']
<Page[0] (label='Cover1')>
```

Get pages slice (all even number pages)

```
>>> even_pages = doc[0::2]
[<Page[0]>, <Page[2]>, <Page[4]>, ...]
```

Iterate over pages

```
>>> for page in doc:
...     print(page)
<Page[0]>
```

(continues on next page)

(continued from previous page)

```
<Page[1]>
<Page[2]>
...
```

Parameters

- **pdf** (str or *file-like*) – Path of pdf file to load or a *file-like* object.
- **ownerpass** (str, optional) – Owner password of pdf file, if encrypted (default `None`)
- **userpass** (str, optional) – User password of pdf file, if encrypted (default `None`)

Raises

- **PDFPermissionError** – If failed to decrypt encrypted PDF using given passwords
- **PDFIOError** – If failed to load file from given file path

filename

name of the file from which pdf document was loaded.

If pdf was loaded from *file-like* object then it will be a empty str.

Type str

has_page_labels

whether pdf has page labels or not

Type bool

info (self)

Get the PDF's info dictionary.

PDF info dictionary contains keys such as *Author*, *Creator*, *ModDate*, etc.

Returns PDF's information dictionary.

Return type dict

is_encrypted

whether pdf is encrypted or not

Warning: Due to a bug in xpdf sources sometimes even non-encrypted PDF documents return `True`

Type bool

is_linearized

whether pdf is linearised or not

Type bool

num_pages

total pages in pdf

Type int

ok_to_add_notes

PDF add notes permission

Type bool

ok_to_change

PDF change permission

Whether PDF can be modified or not. Modifications include:

- Inserting, Deleting, Rotating pages.
- Commenting, filling in form fields, and signing existing signature fields.

Type `bool`

ok_to_copy

PDF copy permission.

Whether pdf content can be copied or not.

Note: PDF copy permission is required for extraction of text and images from document.

Type `bool`

ok_to_print

PDF print permission.

Whether document can be printed or not.

Type `bool`

pdf_version

version of PDF standard pdf comply with

Type `float`

text (*self*, *start*=0, *end*=- 1, *control*=None)

Parse and extract UTF-8 decoded text from given page range.

Extracted text can be adjusted using *control* parameter.

Parameters

- **start** (*int*) – index of first page to extract
- **end** (*int*) – index of last page to extract
- **control** (*TextControl*, optional) – An *TextControl* object, use to control the format of extracted text. (default is `None` which implies text will be extracted using default values from *TextControl* class)

Returns a ‘UTF-8’ decoded str object containing all the extracted text.

Return type `str`

Note: This method is almost similar to `text_bytes()`, the only difference is that it decodes the extracted bytes in *UTF-8* with ‘ignore’ (`codecs.ignore_errors()`) decoding error handler.

See also:

`Page.text()`

`TextOutput()` PDF to Text output device with caching support.

text_bytes (*self*, *int start=0*, *int end=-1*, *TextControl control=None*)

Parse and extract text from given page range.

Extracted text can be adjusted using *control* parameter. This method should be use when text encoding (*Config.text_encoding*) is different than *UTF-8* or when you to control decoding of bytes by yourself.

Parameters

- **start** (*int*) – index of first page to extract
- **end** (*int*) – index of last page to extract
- **control** (*TextControl*) – An *TextControl* object, use to control the format of extacted text. (default is *None* which implies text will be extracted using default values from *TextOutput* class)

Returns a *Config.text_encoding* encoded bytes object containing all the extracted text.

Return type *bytes*

See also:

Page.text_bytes()

TextOutput() PDF to Text output device with caching support.

xmp_metadata (*self*)

Get the PDF's xmp metadata.

Returns

Return type *str*

3.1.2 Page

class pyxpdf.xpdf.**Page** (*doc*, *index*)

Represents a PDF page

Examples

```
>>> page1 = doc[1]
```

Page index and label (if any)

```
>>> page1.index
1
>>> page1.label
'Cover1'
```

Page BBox(s)

```
>>> page1.mediabox
(0.0, 0.0, 612.0, 792.0)
>>> page1.cropbox
(0.0, 0.0, 612.0, 792.0)
>>> page1.mediabox
(0.0, 0.0, 612.0, 792.0)
```

Find text location in Page

```
>>> page1.find_text("Hello")
(100.0, 74.768, 117.328, 96.968)
```

Parameters

- **doc** (*Document*) – Parent pdf Document
- **index** (*int*) – index of pdf Page

Raises *IndexError* – If *index* parameter is outside page range

doc

Parent pdf document

Type *Document*, readonly

index

Type *int*, readonly

label

Type *str*, readonly

artbox

Page's art box coordinates

Type tuple of float, (x1, y1, x2, y2)

bleedbox

Page's bleed box coordinates

Type tuple of float, (x1, y1, x2, y2)

crop_height

page cropbox width

Type *float*

crop_width

page cropbox width

Type *float*

cropbox

Page's crop box coordinates

Type tuple of float, (x1, y1, x2, y2)

find_all_text (*self, text, search_box=None, case_sensitive=False, wholeword=False, rotation=0*)

Find the *text* and get all the matches

Same as *find_text()*, but return all the matches.

Parameters

- **text** (*str*) – Text to search in page
- **search_box** (*tuple of float, optional*) – tuple of coordinates of *BBox* to set the search area. (default is *None*, means the whole page area)
- **case_sensitive** (*bool, optional*) – If *False*, match the *text* regardless of its case in page (default is *True*)

- **wholeword** (*bool, optional*) – match the *text* as a whole word only. (default is `True`)
- **rotation** (*int, optional*) – rotation of page (default is `0`)

find_text (*self, text, search_box=None, direction='top', case_sensitive=False, wholeword=False, rotation=0*)

Find the text in Page.

Search for the *text* in given *search_box* (*BBox*) of page. If *wholeword* then try to match as a whole word.

if *direction* is 'top' then, start the search from top of page

if *direction* is 'next' then, get the next match from the page

if *direction* is 'previous' then, get the previous match from the page

Parameters

- **text** (*str*) – Text to search in page
- **search_box** (*tuple of float, optional*) – tuple of coordinates of *BBox* to set the search area. (default is `None`, means the whole page area)
- **direction** (*{'top', 'next', 'previous'}*) – style of search
- **case_sensitive** (*bool, optional*) – If `False`, match the *text* regardless of its case in page (default is `True`)
- **wholeword** (*bool, optional*) – match the *text* as a whole word only. (default is `True`)
- **rotation** (*int, optional*) – rotation of page (default is `0`)

Returns If match is found then tuple of coordinates(x1, y1, x2, y2) of *BBox* of *text* in page else `None`

Return type tuple of float, `None`

See also:

`find_all_text()`

is_cropped

whether page is cropped or not

Type `bool`

media_height

page mediabox height

Type `float`

media_width

page mediabox width

Type `float`

mediabox

Page's media box coordinates

Type tuple of float, (x1, y1, x2, y2)

rotation

page rotation in degrees

Type `int`

text (*self*, *page_area=None*, *control=None*)

Parse and extract UTF-8 decoded text from current page.

Extracted text can be adjusted using *control* parameter.

Parameters

- **page_area** (*tuple of float, optional*) – tuple of coordinates of *BBox* to set the extraction area. Only text which is inside provided *page_area* will be extracted. (default is *None*, means the whole page area)
- **control** (*TextControl*, optional) – An *TextControl* object, use to control the format of extracted text. (default is *None* which implies text will be extracted using default values from *TextControl* class)

Returns a ‘UTF-8’ decoded str object containing all the extracted text.

Return type *str*

Note: This method is almost similar to *text_bytes()*, the only difference is that it decodes the extracted bytes in *UTF-8* with ‘*ignore*’ (*codecs.ignore_errors()*) decoding error handler.

See also:

TextOutput() PDF to Text output device with caching support.

text_bytes (*self*, *page_area=None*, *TextControl control=None*)

Parse and extract text bytes from current page.

Extracted text can be adjusted using *control* parameter. This method should be use when text encoding (*Config.text_encoding*) is different than *UTF-8* or when you to control decoding of bytes by yourself.

Parameters

- **page_area** (*tuple of float, optional*) – tuple of coordinates of *BBox* to set the extraction area. Only text which is inside provided *page_area* will be extracted. (default is *None*, means the whole page area)
- **control** (*TextControl*) – An *TextControl* object, use to control the format of extracted text. (default is *None* which implies text will be extracted using default values from *TextOutput* class)

Returns a *Config.text_encoding* encoded bytes object containing all the extracted text.

Return type *bytes*

See also:

TextOutput() PDF to Text output device with caching support.

trimbox

Page’s trim box coordinates

Type tuple of float, (x1, y1, x2, y2)

3.2 Output Devices

Output devices process PDF *Page* and generate/extract resources from them.

All the Output devices inherit from base Output Device:

```
class pyxpdf.xpdf.PDFOutputDevice
```

Generic PDF Output Device

All PDF Output Device inherit from this.

```
get (self, int page_no, **kwargs)
```

Get the output of *page_no* indexed page

Currently there are three Output devices implemented:

3.2.1 TextOutput Device

In *TextOutput* output Device we use *TextControl* to set settings for text extraction/analysis

TextOutput

```
class pyxpdf.xpdf.TextOutput
```

Text extract/analysis PDF Output device

Extract text and do layout analysis on from PDF *Document* while caching results. Page texts are cached for faster access. Page texts are lazy loaded, they are loaded only when you first access them.

Parameters

- **doc** (*Document*) – PDF Document for this output device
- **control** (*TextControl*, *optional*) – An *TextControl* object for settings to adjust TextControl extraction/analysis. (default is *None*)
- **kwargs** – *TextControl* parameters which will be used if *control* is not provided.

doc

Parent PDF Document

Type *Document*, readonly

control

Layout settings for output device

Type *TextControl*

Raises *XPDFInternalError* – If cannot initialize internal *xpdf* objects with settings provided

```
get (self, int page_no)
```

Get the extracted *UTF-8* decoded *str* from *page_no* indexed page

This method is almost similar to *get_bytes()*, the only difference is that it decodes the extracted bytes in *UTF-8* with 'ignore' (*codecs.ignore_errors()*) decoding error handler.

Parameters *page_no* (*int*) – index of page to extract text bytes from

Returns extracted *UTF-8* decoded text

Return type *str*

`get_all(self) → list`

Get the extracted *UTF-8* decoded text from all pages

Returns list of *UTF-8* decoded text from all the pages

Return type `list` of `str`

`get_bytes(self, int page_no) → bytes`

Get the extracted text bytes from *page_no* indexed page

This method should be use when text encoding (`Config.text_encoding`) is different than *UTF-8* or when you to control decoding of bytes by yourself.

Parameters `page_no (int)` – index of page to extract text bytes from

Returns extracted text bytes

Return type `bytes`

TextControl

`class pyxpdf.xpdf.TextControl`

Parameters for Text extraction and layout analysis

Text layout modes:

- **reading** Keep the text in reading order. It ‘undo’ physical layout (columns, hyphenation, etc.) and output the text in reading order.
- **physical** Maintain (as best as possible) the original physical layout of the text. If the *fixed_pitch* option is given, character spacing within each line will be determined by the specified character pitch.
- **table** It is similar to *physical* layout mode, but optimized for tabular data, with the goal of keeping rows and columns aligned (at the expense of inserting extra whitespace). If the *fixed_pitch* option is given, character spacing within each line will be determined by the specified character pitch.
- **simple** Similar to *physical* layout, but optimized for simple one-column pages. This mode will do a better job of maintaining horizontal spacing, but it will only work properly with a single column of text.
- **lineprinter** Line printer mode uses a strict fixed character pitch and height layout. That is, the page is broken into a grid, and characters are placed into that grid. If the grid spacing is too small for the actual characters, the result is extra whitespace. If the grid spacing is too large, the result is missing whitespace. The grid spacing can be specified using the *fixed_pitch* and *fixed_line_spacing* options. If one or both are not given on the xpdf will attempt to compute appropriate value(s).
- **raw** Keep the text in content stream order. Depending on how the PDF file was generated, this may or may not be useful.

Parameters

- **mode** (`{"reading", "table", "simple", "physical", "lineprinter", "raw"}`) – text analysis/extraction layout mode
- **fixed_pitch** (`float, optional`) – Specify the character pitch (character width), for *physical*, *table*, or *lineprinter* mode. This is ignored in all other modes. (default is 0, means approximate characters’ pitch will be calculated)
- **fixed_line_spacing** (`float, optional`) – Specify the line spacing, in points, for *lineprinter* mode. This is ignored in all other modes. (default is 0, means approximate line spacing will be calculated)

- **enable_html** (*bool, optional*) – enable extra processing for html. (default is `False`)
- **clip_text** (*bool, optional*) – Text which is hidden because of clipping is removed before doing layout, and then added back in. This can be helpful for tables where clipped (invisible) text would overlap the next column. (default is `False`)
- **discard_clipped** (*bool, optional*) – discard all clipped characters (default is `False`)
- **discard_diagonal** (*bool, optional*) – Diagonal text, i.e., text that is not close to one of the 0, 90, 180, or 270 degree axes, is discarded. This is useful to skip watermarks drawn on top of body text, etc. (default is `False`)
- **discard_invisible** (*bool, optional*) – discard all invisible characters (default is `False`)
- **insert_bom** (*bool, optional*) – Insert a Unicode byte order marker (BOM) at the start of the text output.
- **margin_left** (*float, optional*) – Specifies the left margin. Text in the left margin (i.e., within that many points of the left edge of the page) is discarded. (default is 0)
- **margin_right** (*float, optional*) – Specifies the right margin. Text in the right margin (i.e., within that many points of the right edge of the page) is discarded. (default is 0)
- **margin_top** (*float, optional*) – Specifies the top margin. Text in the top margin (i.e., within that many points of the top edge of the page) is discarded. (default is 0)
- **margin_bottom** (*float, optional*) – Specifies the bottom margin. Text in the bottom margin (i.e., within that many points of the bottom edge of the page) is discarded. (default is 0)

Raises `ValueError` – If *mode* invalid

3.2.2 RawImageOutput Device

RawImageOutput

class pyxpdf.xpdf.**RawImageOutput**

Render PDF page as *Image*.

Convert the PDF page to uncompressed raw image.

paper_color depends on the color mode of image, if color mode is *RGB* or *RGBA* than *paper_color* should be a 3 int(0-255) tuple of RGB values, similarly for *CMYK* it should be 4 int(0-255) tuple of CMYK color values.

If you are using image mode with alpha channel and want transparent background then set *no_composite* to *True*

Parameters

- **doc** (*Document*) – PDF Document for this output device
- **mode** (*{ "RGB", "RGBA", "L", "LA", "1", "CMYK" }, optional*) – image modes for output rendered image, equivalent to Pillow's image modes. (default is 'RGB')
- **paper_color** (*tuple of int, optional*) – paper color for rendered pdf page (default is `None`, means 'white' paper color)
- **resolution** (*float, optional*) – X and Y resolution of output image in DPI (default is 150)

- **resolution_x**(*float, optional*) – X resolution in DPI (default is 150)
- **resolution_y**(*float, optional*) – Y resolution in DPI (default is 150)
- **anti_alias**(*bool, optional*) – enable font anti-aliasing for rendering (default is *True*)
- **no_composite**(*bool, optional*) – disables the final composite (with the opaque paper color), resulting in transparent output. (default is *False*)
- **use_cropbox**(*bool, optional*) – use the crop box rather than media box (default is *False*)
- **scale_before_rotation**(*bool, optional*) – resize dimensions before rotation of rotated pdfs (default is *False*)

Note: Additionally you can enable `Config.vector_anti_alias` for better anti-alias effect.

Warning: Avoid ‘l’ image mode, as of now its quite buggy and fonts are not rendered properly in it. Instead use ‘L’ for black and white.

get (*self, int page_no, crop_box=(0, 0, 0, 0), scale_pixel_box=None*)
 Get the rendered `Image` for *page_no* indexed page

Parameters

- **page_no**(*int*) – index of page to render
- **crop_box**(*tuple of float, optional*) – tuple of coordinates of `BBox` to set the rendering area. (default is (0,0,0,0), means the whole page area)
- **scale_pixel_box**(*tuple of int, optional*) – tuple of pair of int which scales the page to fit within x * y pixels

Returns Rendered PDF Page

Return type `Image`

Note: Requires Optional dependency `Pillow` module

resolution_x

‘double’

Type resolution_x

resolution_y

‘double’

Type resolution_y

scale_before_rotation

‘bool’

Type scale_before_rotation

use_cropbox

‘bool’

Type use_cropbox

3.2.3 PDFImageOutput Device

PDFImage

class pyxpdf.xpdf.**PDFImage**

Represents a PDF Image.

Image Colorspace:

- **gray** : DeviceGray, CalGray
- **rgb** : DeviceRGB, CalRGB
- **cmyk** : DeviceCMYK
- **lab** : Lab
- **icc** : ICCBased
- **index** : Indexed
- **sep** : Separation
- **devn** : DeviceN

Image Compression:

- **ccitt** : CCITTFax
- **jpeg** : DCT
- **jpx** : JPX
- **jbig2** : JBIG2
- **flate** : Flate
- **lzw** : LZW
- **rle** : RunLength

bbox

Image's Boundary Box (*BBox*)

Type tuple of float

image

Image data as Pillow Image

Type Image

page_index

Index of Image's PDF page

Type int

interpolate

Whether image is interpolated or not

Type bool

is_inline

Whether image is inline or not

Type bool

hDPI

Image's horizontal DPI

Type `float`

vDPI

Image's vertical DPI

Type `float`

colorspace

Image's color space.

Type `{ 'gray', 'rgb', 'cmyk', 'lab', 'icc', 'index', 'sep', 'devn', 'unknown' }`

components

components in the image's colorspace.

Type `int`

bpc

bits per component.

Type `int`

compression

Image's compression

Type `{ 'ccitt', 'jpeg', 'jpx', 'jbig2', 'flate', 'lzw', 'rle', 'unknown' }`

PDFImageOutput

class `pyxpdf.xpdf.PDFImageOutput`

Extract the images from PDF Document

Extract and decode images inside a PDF and output them as `Image` object.

Parameters `doc` (`Document`) – PDF Document for this output device

Note: Requires Optional dependency `Pillow` module

get (`self`, `page_no`) → `list`

Get all the images from `page_no` indexed page.

Parameters `page_no` (`int`) – index of page to render

Returns All the images in PDF Page

Return type list of `PDFImage`

3.2.4 Page Iterator

To iterate over a PDF Output Device page wise, we have `page_iterator`:

class `pyxpdf.xpdf.page_iterator` (`output`, `**kwargs`)

Iterate over PDF output devices by page.

Parameters

- **output** – PDF output device to iterate over
- **kwargs** – All the optional arguments to pass to `get()` method of output device

Examples

Iterate pages text from *TextOutput*

```
>>> text_out = TextOutput(doc)
>>> for page_text in page_iterator(text_out)
...     print(page_text)
```

Iterate images from *RawImageOutput* with specific *crop_box*

```
>>> image_out = RawImageOutput(doc)
>>> for image in page_iterator(image_out, crop_box=(0,0,500,500)):
...     image.show()      # pillow image
```

3.3 Global Config

All the settings for xpdf is managed through a global object *Config*.

For most use case settings available from *Config* will be sufficient, but if you want more you can load a *xpdfrc* file.

See :

3.3.1 xpdfrc Configuration File

Sample xpdfrc [here](#)

xpdfrc(5)	File Formats Manual	xpdfrc(5)
NAME		
xpdfrc - configuration file for Xpdf tools (version 4.02)		
DESCRIPTION		
All of the Xpdf tools read a single configuration file. If you have a .xpdfrc file in your home directory, it will be read. Otherwise, a system-wide configuration file will be read from /usr/local/etc/xpdfrc, if it exists. (This is its default location; depending on build options, it may be placed elsewhere.) On Win32 systems, the xpdfrc file should be placed in the same directory as the executables.		
The xpdfrc file consists of a series of configuration options, one per line. Blank lines and lines starting with a '#' (comments) are ignored.		
Arguments may be quoted, using "double-quote" characters, e.g., for file names that contain spaces.		
The following sections list all of the configuration options, sorted into functional groups. There is an examples section at the end.		
INCLUDE FILES		
include config-file		
Includes the specified config file. The effect of this is equivalent to inserting the contents of config-file directly		

(continues on next page)

(continued from previous page)

into the parent config file in place of the include command. Config files can be nested arbitrarily deeply.

GENERAL FONT CONFIGURATION

`fontFile PDF-font-name font-file`

Maps a PDF font, PDF-font-name, to a font for display or PostScript output. The font file, font-file, can be any type allowed in a PDF file. This command can be used for 8-bit or 16-bit (CID) fonts.

`fontDir dir`

Specifies a search directory for font files. There can be multiple fontDir commands; all of the specified directories will be searched in order. The font files can be Type 1 (.pfa or .pfb) or TrueType (.ttf or .ttc); other files in the directory will be ignored. The font file name (not including the extension) must exactly match the PDF font name. This search is performed if the font name doesn't match any of the fonts declared with the fontFile command. There are no default fontDir directories.

`fontFileCC registry-ordering font-file`

Maps the registry-ordering character collection to a font for display or PostScript output. This mapping is used if the font name doesn't match any of the fonts declared with the fontFile, fontDir, psResidentFont16, or psResidentFontCC commands.

POSTSCRIPT FONT CONFIGURATION

`psFontPassthrough yes | no`

If set to "yes", pass 8-bit font names through to the PostScript output without substitution. Fonts which are not embedded in the PDF file are expected to be available on the printer. This defaults to "no".

`psResidentFont PDF-font-name PS-font-name`

When the 8-bit font PDF-font-name is used (without embedding) in a PDF file, it will be translated to the PostScript font PS-font-name, which is assumed to be resident in the printer. Typically, PDF-font-name and PS-font-name are the same. By default, only the Base-14 fonts are assumed to be resident.

`psResidentFont16 PDF-font-name wMode PS-font-name encoding`

When the 16-bit (CID) font PDF-font-name with writing mode wMode is used (without embedding) in a PDF file, it will be translated to the PostScript font PS-font-name, which is assumed to be resident in the printer. The writing mode must be either 'H' for horizontal or 'V' for vertical. The resident font is assumed to use the specified encoding (which must have been defined with the unicodeMap command).

`psResidentFontCC registry-ordering wMode PS-font-name encoding`

When a 16-bit (CID) font using the registry-ordering character collection and wMode writing mode is used (without embedding) in a PDF file, the PostScript font, PS-font-name, is substituted for it. The substituted font is assumed to be resident in the printer. The writing mode must be either 'H' for horizontal or 'V' for vertical. The resident font is assumed to use the specified encoding (which must have been defined with the unicodeMap

(continues on next page)

(continued from previous page)

```

command).

psEmbedType1Fonts yes | no
    If set to "no", prevents embedding of Type 1 fonts in generated
    PostScript. This defaults to "yes".

psEmbedTrueTypeFonts yes | no
    If set to "no", prevents embedding of TrueType fonts in gener-
    ated PostScript. This defaults to "yes".

psEmbedCIDTrueTypeFonts yes | no
    If set to "no", prevents embedding of CID TrueType fonts in gen-
    erated PostScript. For Level 3 PostScript, this generates a CID
    font, for lower levels it generates a non-CID composite font.
    This defaults to "yes".

psEmbedCIDPostScriptFonts yes | no
    If set to "no", prevents embedding of CID PostScript fonts in
    generated PostScript. For Level 3 PostScript, this generates a
    CID font, for lower levels it generates a non-CID composite
    font. This defaults to "yes".

```

POSTSCRIPT CONTROL

```

psPaperSize width(pts) height(pts)
    Sets the paper size for PostScript output. The width and height
    parameters give the paper size in PostScript points (1 point =
    1/72 inch).

psPaperSize letter | legal | A4 | A3 | match
    Sets the paper size for PostScript output to a standard size.
    The default paper size is set when xpdf and pdftops are built,
    typically to "letter" or "A4". This can also be set to "match",
    which will set the paper size to match the size specified in the
    PDF file.

psImageableArea llx lly urx ury
    Sets the imageable area for PostScript output. The four inte-
    gers are the coordinates of the lower-left and upper-right cor-
    ners of the imageable region, specified in points (with the ori-
    gin being the lower-left corner of the paper). This defaults to
    the full paper size; the psPaperSize option will reset the
    imageable area coordinates.

psCrop yes | no
    If set to "yes", PostScript output is cropped to the CropBox
    specified in the PDF file; otherwise no cropping is done. This
    defaults to "yes".

psUseCropBoxAsPage yes | no
    If set to "yes", PostScript output treats the CropBox as the
    page size. By default, this is "no", and the MediaBox is used
    as the page size.

psExpandSmaller yes | no
    If set to "yes", PDF pages smaller than the PostScript imageable
    area are expanded to fill the imageable area. Otherwise, no
    scaling is done on smaller pages. This defaults to "no".

```

(continues on next page)

(continued from previous page)

`psShrinkLarger yes | no`
 If set to `yes`, PDF pages larger than the PostScript imageable area are shrunk to fit the imageable area. Otherwise, no scaling is done on larger pages. This defaults to `"yes"`.

`psCenter yes | no`
 If set to `yes`, PDF pages smaller than the PostScript imageable area (after any scaling) are centered in the imageable area. Otherwise, they are aligned at the lower-left corner of the imageable area. This defaults to `"yes"`.

`psDuplex yes | no`
 If set to `"yes"`, the generated PostScript will set the `"Duplex"` `pagedevice` entry. This tells duplex-capable printers to enable duplexing. This defaults to `"no"`.

`psLevel level1 | level1sep | level2 | level2gray | level2sep | level3 | level3gray | level3sep`
 Sets the PostScript level to generate. This defaults to `"level2"`.

`psPreload yes | no`
 If set to `"yes"`, PDF forms are converted to PS procedures, and image data is preloaded. This uses more memory in the PostScript interpreter, but generates significantly smaller PS files in situations where, e.g., the same image is drawn on every page of a long document. This defaults to `"no"`.

`psOPI yes | no`
 If set to `"yes"`, generates PostScript OPI comments for all images and forms which have OPI information. This option is only available if the Xpdf tools were compiled with OPI support. This defaults to `"no"`.

`psASCIIHex yes | no`
 If set to `"yes"`, the `ASCIIHexEncode` filter will be used instead of `ASCII85Encode` for binary data. This defaults to `"no"`.

`psLZW yes | no`
 If set to `"yes"`, the `LZWEncode` filter will be used for lossless compression in PostScript output; if set to `"no"`, the `RunLengthEncode` filter will be used instead. LZW generates better compression (smaller PS files), but may not be supported by some printers. This defaults to `"yes"`.

`psUncompressPreloadedImages yes | no`
 If set to `"yes"`, all preloaded images in PS files will be uncompressed. If set to `"no"`, the original compressed images will be used when possible. The `"yes"` setting is useful to work around certain buggy PostScript interpreters. This defaults to `"no"`.

`psMinLineWidth float`
 Set the minimum line width, in points, for PostScript output. The default value is 0 (no minimum).

`psRasterResolution float`

(continues on next page)

(continued from previous page)

Set the resolution (in dpi) for rasterized pages in PostScript output. (Pdftops will rasterize pages which use transparency.) This defaults to 300.

psRasterMono yes | no

If set to "yes", rasterized pages in PS files will be monochrome (8-bit gray) instead of color. This defaults to "no".

psRasterSliceSize pixels

When rasterizing pages, pdftops splits the page into horizontal "slices", to limit memory usage. This option sets the maximum slice size, in pixels. This defaults to 20000000 (20 million).

psAlwaysRasterize yes | no

If set to "yes", all PostScript output will be rasterized. This defaults to "no".

psNeverRasterize yes | no

Pdftops rasterizes an pages that use transparency (because PostScript doesn't support transparency). If psNeverRasterize is set to "yes", rasterization is disabled: pages will never be rasterized, even if they contain transparency. This will likely result in incorrect output for PDF files that use transparency, and a warning message to that effect will be printed. This defaults to "no".

fontDir dir

See the description above, in the DISPLAY FONTS section.

TEXT CONTROL AND CHARACTER MAPPING

textEncoding encoding-name

Sets the encoding to use for text output. (This can be overridden with the "-enc" switch on the command line.) The encoding-name must be defined with the unicodeMap command (see above). This defaults to "Latin1".

textEOL unix | dos | mac

Sets the end-of-line convention to use for text output. The options are:

```
unix = LF
dos  = CR+LF
mac  = CR
```

(This can be overridden with the "-eol" switch on the command line.) The default value is based on the OS where xpdf and pdftotext were built.

textPageBreaks yes | no

If set to "yes", text extraction will insert page breaks (form feed characters) between pages. This defaults to "yes".

textKeepTinyChars yes | no

If set to "yes", text extraction will keep all characters. If set to "no", text extraction will discard tiny (smaller than 3 point) characters after the first 50000 per page, avoiding extremely slow run times for PDF files that use special fonts to

(continues on next page)

(continued from previous page)

do shading or cross-hatching. This defaults to "yes".

nameToUnicode map-file

Specifies a file with the mapping from character names to Unicode. This is used to handle PDF fonts that have valid encodings but no ToUnicode entry. Each line of a nameToUnicode file looks like this:

```
hex-string name
```

The hex-string is the Unicode (UCS-2) character index, and name is the corresponding character name. Multiple nameToUnicode files can be used; if a character name is given more than once, the code in the last specified file is used. There is a built-in default nameToUnicode table with all of Adobe's standard character names.

cidToUnicode registry-ordering map-file

Specifies the file with the mapping from character collection to Unicode. Each line of a cidToUnicode file represents one character:

```
hex-string
```

The hex-string is the Unicode (UCS-2) index for that character. The first line maps CID 0, the second line CID 1, etc. File size is determined by size of the character collection. Only one file is allowed per character collection; the last specified file is used. There are no built-in cidToUnicode mappings.

unicodeToUnicode font-name-substring map-file

This is used to work around PDF fonts which have incorrect Unicode information. It specifies a file which maps from the given (incorrect) Unicode indexes to the correct ones. The mapping will be used for any font whose name contains font-name-substring. Each line of a unicodeToUnicode file represents one Unicode character:

```
in-hex out-hex1 out-hex2 ...
```

The in-hex field is an input (incorrect) Unicode index, and the rest of the fields are one or more output (correct) Unicode indexes. Each occurrence of in-hex will be converted to the specified output sequence.

unicodeRemapping remap-file

Remap Unicode characters when doing text extraction. This specifies a file that maps from a particular Unicode index to zero or more replacement Unicode indexes. Each line of the remap file represents one Unicode character:

```
in-hex out-hex1 out-hex2 ...
```

Any Unicode characters not listed will be left unchanged. This function is typically used to remap things like non-breaking spaces, soft hyphens, ligatures, etc.

(continues on next page)

(continued from previous page)

`unicodeMap encoding-name map-file`

Specifies the file with mapping from Unicode to encoding-name. These encodings are used for text output (see below). Each line of a unicodeMap file represents a range of one or more Unicode characters which maps linearly to a range in the output encoding:

```
in-start-hex in-end-hex out-start-hex
```

Entries for single characters can be abbreviated to:

```
in-hex out-hex
```

The in-start-hex and in-end-hex fields (or the single in-hex field) specify the Unicode range. The out-start-hex field (or the out-hex field) specifies the start of the output encoding range. The length of the out-start-hex (or out-hex) string determines the length of the output characters (e.g., UTF-8 uses different numbers of bytes to represent characters in different ranges). Entries must be given in increasing Unicode order. Only one file is allowed per encoding; the last specified file is used. The Latin1, ASCII7, Symbol, ZapfDingbats, UTF-8, and UCS-2 encodings are predefined.

`cMapDir registry-ordering dir`

Specifies a search directory, dir, for CMaps for the registry-ordering character collection. There can be multiple directories for a particular collection. There are no default CMap directories.

`toUnicodeDir dir`

Specifies a search directory, dir, for ToUnicode CMaps. There can be multiple ToUnicode directories. There are no default ToUnicode directories.

`mapNumericCharNames yes | no`

If set to "yes", the Xpdf tools will attempt to map various numeric character names sometimes used in font subsets. In some cases this leads to usable text, and in other cases it leads to gibberish -- there is no way for Xpdf to tell. This defaults to "yes".

`mapUnknownCharNames yes | no`

If set to "yes", and mapNumericCharNames is set to "no", the Xpdf tools will apply a simple pass-through mapping (Unicode index = character code) for all unrecognized glyph names. (For CID fonts, setting mapNumericCharNames to "no" is unnecessary.) In some cases, this leads to usable text, and in other cases it leads to gibberish -- there is no way for Xpdf to tell. This defaults to "no".

`mapExtTrueTypeFontsViaUnicode yes | no`

When rasterizing text using an external TrueType font, there are two options for handling character codes. If mapExtTrueTypeFontsViaUnicode is set to "yes", Xpdf will use the font encoding/ToUnicode info to map character codes to Unicode, and then use the font's Unicode cmap to map Unicode to GIDs. If mapExt-

(continues on next page)

(continued from previous page)

TrueTypeFontsViaUnicode is set to "no", Xpdf will assume the character codes are GIDs (i.e., use an identity mapping). This defaults to "yes".

dropFont font-name

Drop all text drawn in the specified font. To drop text drawn in unnamed fonts, use:

```
dropFont ""
```

There can be any number of dropFont commands.

RASTERIZER SETTINGS

enableFreeType yes | no

Enables or disables use of FreeType (a TrueType / Type 1 font rasterizer). This is only relevant if the Xpdf tools were built with FreeType support. ("enableFreeType" replaces the old "freetypeControl" option.) This option defaults to "yes".

disableFreeTypeHinting yes | no

If this is set to "yes", FreeType hinting will be forced off. This option defaults to "no".

antialias yes | no

Enables or disables font anti-aliasing in the PDF rasterizer. This option affects all font rasterizers. ("antialias" replaces the anti-aliasing control provided by the old "tllibControl" and "freetypeControl" options.) This default to "yes".

vectorAntialias yes | no

Enables or disables anti-aliasing of vector graphics in the PDF rasterizer. This defaults to "yes".

antialiasPrinting yes | no

If this is "yes", bitmaps sent to the printer will be antialiased (according to the "antialias" and "vectorAntialias" settings). If this is "no", printed bitmaps will not be antialiased. This defaults to "no".

strokeAdjust yes | no | cad

Sets the stroke adjustment mode. If set to "no", no stroke adjustment will be done. If set to "yes", normal stroke adjustment will be done: horizontal and vertical lines will be moved by up to half a pixel to make them look cleaner when vector anti-aliasing is enabled. If set to "cad", a slightly different stroke adjustment algorithm will be used to ensure that lines of the same original width will always have the same adjusted width (at the expense of allowing gaps and overlaps between adjacent lines). This defaults to "yes".

forceAccurateTiling yes | no

If this is set to "yes", the TilingType is forced to 2 (no distortion) for all tiling patterns, regardless of the setting in the pattern dictionary. This defaults to "no".

screenType dispersed | clustered | stochasticClustered

Sets the halftone screen type, which will be used when generat-

(continues on next page)

(continued from previous page)

ing a monochrome (1-bit) bitmap. The three options are dispersed-dot dithering, clustered-dot dithering (with a round dot and 45-degree screen angle), and stochastic clustered-dot dithering. By default, "stochasticClustered" is used for resolutions of 300 dpi and higher, and "dispersed" is used for resolutions lower than 300 dpi.

screenSize integer

Sets the size of the (square) halftone screen threshold matrix. By default, this is 4 for dispersed-dot dithering, 10 for clustered-dot dithering, and 100 for stochastic clustered-dot dithering.

screenDotRadius integer

Sets the halftone screen dot radius. This is only used when screenType is set to stochasticClustered, and it defaults to 2. In clustered-dot mode, the dot radius is half of the screen size. Dispersed-dot dithering doesn't have a dot radius.

screenGamma float

Sets the halftone screen gamma correction parameter. Gamma values greater than 1 make the output brighter; gamma values less than 1 make it darker. The default value is 1.

screenBlackThreshold float

When halftoning, all values below this threshold are forced to solid black. This parameter is a floating point value between 0 (black) and 1 (white). The default value is 0.

screenWhiteThreshold float

When halftoning, all values above this threshold are forced to solid white. This parameter is a floating point value between 0 (black) and 1 (white). The default value is 1.

minLineWidth float

Set the minimum line width, in device pixels. This affects the rasterizer only, not the PostScript converter (except when it uses rasterization to handle transparency). The default value is 0 (no minimum).

enablePathSimplification yes | no

If set to "yes", simplify paths by removing points where it won't make a significant difference to the shape. The default value is "no".

overprintPreview yes | no

If set to "yes", generate overprint preview output, honoring the OP/op/OPM settings in the PDF file. Ignored for non-CMYK output. The default value is "no".

VIEWER SETTINGS

These settings only apply to the Xpdf GUI PDF viewer.

initialZoom percentage | page | width

Sets the initial zoom factor. A number specifies a zoom percentage, where 100 means 72 dpi. You may also specify 'page', to fit the page to the window size, or 'width', to fit the page

(continues on next page)

(continued from previous page)

width to the window width.

`defaultFitZoom` percentage
 If xpdf is started with `fit-page` or `fit-width` zoom and no window geometry, it will calculate a desired window size based on the PDF page size and this `defaultFitZoom` value. I.e., the window size will be chosen such that exactly one page will fit in the window at this zoom factor (which must be a percentage). The default value is based on the screen resolution.

`initialDisplayMode` `single` | `continuous` | `sideBySideSingle` | `sideBySideContinuous` | `horizontalContinuous`
 Sets the initial display mode. The default setting is `"continuous"`.

`initialToolbarState` `yes` | `no`
 If set to `"yes"`, xpdf opens with the toolbar visible. If set to `"no"`, xpdf opens with the toolbar hidden. The default is `"yes"`.

`initialSidebarState` `yes` | `no`
 If set to `"yes"`, xpdf opens with the sidebar (tabs, outline, etc.) visible. If set to `"no"`, xpdf opens with the sidebar collapsed. The default is `"yes"`.

`initialSelectMode` `block` | `linear`
 Sets the initial selection mode. The default setting is `"linear"`.

`paperColor` color
 Set the `"paper color"`, i.e., the background of the page display. The color can be `#RRGGBB` (hexadecimal) or a named color. This option will not work well with PDF files that do things like filling in white behind the text.

`matteColor` color
 Set the matte color, i.e., the color used for background outside the actual page area. The color can be `#RRGGBB` (hexadecimal) or a named color.

`fullScreenMatteColor` color
 Set the matte color for full-screen mode. The color can be `#RRGGBB` (hexadecimal) or a named color.

`reverseVideoInvertImages` `yes` | `no`
 If set to `"no"`, xpdf's reverse-video mode inverts text and vector graphic content, but not images. If set to `"yes"`, xpdf inverts images as well. The default is `"no"`.

`popupMenuCmd` title command ...
 Add a command to the popup menu. Title is the text to be displayed in the menu. Command is an Xpdf command (see the `COMMANDS` section of the `xpdf(1)` man page for details). Multiple commands are separated by whitespace.

`maxTileWidth` pixels
 Set the maximum width of tiles to be used by xpdf when rasterizing pages. This defaults to 1500.

(continues on next page)

(continued from previous page)

```

maxTileHeight pixels
    Set the maximum height of tiles to be used by xpdf when raster-
    izing pages. This defaults to 1500.

tileCacheSize tiles
    Set the maximum number of tiles to be cached by xpdf when ras-
    terizing pages. This defaults to 10.

workerThreads numThreads
    Set the number of worker threads to be used by xpdf when raster-
    izing pages. This defaults to 1.

launchCommand command
    Sets the command executed when you click on a "launch"-type
    link. The intent is for the command to be a program/script
    which determines the file type and runs the appropriate viewer.
    The command line will consist of the file to be launched, fol-
    lowed by any parameters specified with the link. Do not use
    "%" in "command". By default, this is unset, and Xpdf will
    simply try to execute the file (after prompting the user).

movieCommand command
    Sets the command executed when you click on a movie annotation.
    The string "%" will be replaced with the movie file name. This
    has no default value.

defaultPrinter printer
    Sets the default printer used in the viewer's print dialog.

bind modifiers-key context command ...
    Add a key or mouse button binding. Modifiers can be zero or
    more of:

        shift-
        ctrl-
        alt-

    Key can be a regular ASCII character, or any one of:

        space
        tab
        return
        enter
        backspace
        esc
        insert
        delete
        home
        end
        pgup
        pgdn
        left / right / up / down      (arrow keys)
        f1 .. f35                     (function keys)
        mousePress1 .. mousePress7    (mouse buttons)
        mouseRelease1 .. mouseRelease7 (mouse buttons)
        mouseClicked1 .. mouseClicked (mouse buttons)

```

(continues on next page)

(continued from previous page)

Context is either "any" or a comma-separated combination of:

fullScreen / window	(full screen mode on/off)
continuous / singlePage	(continuous mode on/off)
overLink / offLink	(mouse over link or not)
scrLockOn / scrLockOff	(scroll lock on/off)

The context string can include only one of each pair in the above list.

Command is an Xpdf command (see the COMMANDS section of the xpdf(1) man page for details). Multiple commands are separated by whitespace.

The bind command replaces any existing binding, but only if it was defined for the exact same modifiers, key, and context. All tokens (modifiers, key, context, commands) are case-sensitive.

Example key bindings:

```
# bind ctrl-a in any context to the nextPage
# command
bind ctrl-a any nextPage

# bind uppercase B, when in continuous mode
# with scroll lock on, to the reload command
# followed by the prevPage command
bind B continuous,scrLockOn reload prevPage
```

See the xpdf(1) man page for more examples.

unbind modifiers-key context

Removes a key binding established with the bind command. This is most useful to remove default key bindings before establishing new ones (e.g., if the default key binding is given for "any" context, and you want to create new key bindings for multiple contexts).

tabStateFile path

Sets the file used by the loadTabState and saveTabState commands (see the xpdf(1) man page for more information).

MISCELLANEOUS SETTINGS

drawAnnotations yes | no

If set to "no", annotations will not be drawn or printed. The default value is "yes".

drawFormFields yes | no

If set to "no", form fields will not be drawn or printed. The default value is "yes".

enableXFA yes | no

If set to "yes", an XFA form (if present) will be rendered in place of an AcroForm. If "no", an XFA form will never be rendered. This defaults to "yes".

(continues on next page)

(continued from previous page)

```

printCommands yes | no
    If set to "yes", drawing commands are printed as they're exe-
    cuted (useful for debugging). This defaults to "no".

errQuiet yes | no
    If set to "yes", this suppresses all error and warning messages
    from all of the Xpdf tools. This defaults to "no".

```

EXAMPLES

The following is a sample xpdfrc file.

```

# from the Thai support package
nameToUnicode /usr/local/share/xpdf/Thai.nameToUnicode

# from the Japanese support package
cidToUnicode Adobe-Japan1 /usr/local/share/xpdf/Adobe-Japan1.cidToUnicode
unicodeMap JISX0208 /usr/local/share/xpdf/JISX0208.unicodeMap
cMapDir Adobe-Japan1 /usr/local/share/xpdf/cmap/Adobe-Japan1

# use the Base-14 Type 1 fonts from ghostscript
fontFile Times-Roman /usr/local/share/ghostscript/fonts/n021003l.pfb
fontFile Times-Italic /usr/local/share/ghostscript/fonts/n021023l.pfb
fontFile Times-Bold /usr/local/share/ghostscript/fonts/n021004l.pfb
fontFile Times-BoldItalic /usr/local/share/ghostscript/fonts/n021024l.pfb
fontFile Helvetica /usr/local/share/ghostscript/fonts/n019003l.pfb
fontFile Helvetica-Oblique /usr/local/share/ghostscript/fonts/n019023l.pfb
fontFile Helvetica-Bold /usr/local/share/ghostscript/fonts/n019004l.pfb
fontFile Helvetica-BoldOblique /usr/local/share/ghostscript/fonts/n019024l.pfb
fontFile Courier /usr/local/share/ghostscript/fonts/n022003l.pfb
fontFile Courier-Oblique /usr/local/share/ghostscript/fonts/n022023l.pfb
fontFile Courier-Bold /usr/local/share/ghostscript/fonts/n022004l.pfb
fontFile Courier-BoldOblique /usr/local/share/ghostscript/fonts/n022024l.pfb
fontFile Symbol /usr/local/share/ghostscript/fonts/s050000l.pfb
fontFile ZapfDingbats /usr/local/share/ghostscript/fonts/d050000l.pfb

# use the Bakoma Type 1 fonts
# (this assumes they happen to be installed in /usr/local/fonts/bakoma)
fontDir /usr/local/fonts/bakoma

# set some PostScript options
psPaperSize letter
psDuplex no
psLevel level2
psEmbedType1Fonts yes
psEmbedTrueTypeFonts yes

# assume that the PostScript printer has the Univers and
# Univers-Bold fonts
psResidentFont Univers Univers
psResidentFont Univers-Bold Univers-Bold

# set the text output options
textEncoding UTF-8
textEOL unix

# misc options
enableFreeType yes

```

(continues on next page)

(continued from previous page)

```

launchCommand    viewer-script

FILES
    /usr/local/etc/xdpfrc
        This is the default location for the system-wide configuration
        file. Depending on build options, it may be placed elsewhere.

    $HOME/.xdpfrc
        This is the user's configuration file. If it exists, it will be
        read in place of the system-wide file.

AUTHOR
    The Xpdf software and documentation are copyright 1996-2019 Glyph &
    Cog, LLC.

SEE ALSO
    xpdf(1), pdftops(1), pdftotext(1), pdftohtml(1), pdfinfo(1), pdf-
    fonts(1), pdfdetach(1), pdftoppm(1), pdftopng(1), pdfimages(1)
    http://www.xpdfreader.com/

                                25 Sep 2019                                xdpfrc(5)

```

xpdfrc manual taken from [xpdfrc](#). Copyright 2002-2003 Glyph & Cog, LLC

3.3.2 Config

pyxpdf.xpdf.Config

Global XPDF config object

`Config.reset()`

Reset the global configuration to default.

`Config.load_file(cfg_path)`

load the settings from given *cfg_path* *xpdfrc*.

`Config.add_font_file(font_name, file)`

Maps a PDF Font *font_name* to font from path *file*. The font files can be Type 1 (.pfa or .pfb) or TrueType (.ttf or .ttc)

`Config.text_encoding`

Sets the encoding to use for text output. 'UTF-8', 'Latin1', 'ASCII7', 'Symbol', 'ZapfDingbats', 'UCS-2' is pre defined. For more encodings support install `pyxpdf_data` package (see [Installation](#)). (default is *UTF-8*)

Type *str*,

`Config.text_eol`

Sets the end-of-line convention to use for text output. The options are

unix = LF

dos = CR+LF

mac = CR

(default, platform dependent)

Type {'unix', 'dos', 'mac'}

Config.text_page_breaks

If set to *True*, text extraction will insert page breaks (form feed characters) between pages. (default is *True*)

Type bool

Config.text_keep_tiny

If set to *True*, text extraction will keep all characters. If set to "no", text extraction will discard tiny (smaller than 3 point) characters after the first 50000 per page, avoiding extremely slow run times for PDF files that use special fonts to do shading or cross-hatching. (default is *True*)

Type bool

Config.enable_freetype

Enables or disables use of FreeType (a TrueType/Type 1 font rasterizer). (default is *True*)

Type bool

Config.anti_alias

Enables or disables font anti-aliasing in the PDF Output Devices. This option affects all font rasterizers. (default is *True*)

Type bool

Config.vector_anti_alias

Enables or disables anti-aliasing of vector graphics in the PDF rasterizer. (default is 'True')

Type bool

3.4 Exceptions

exception pyxpdf.xpdf.PDFError (*message*)

Main exception base class for pyxpdf. All other exceptions inherit from this one.

exception pyxpdf.xpdf.XPDFError (*message=None*)

Base exception class for all xpdf errors.

exception pyxpdf.xpdf.PDFSyntaxError

Problem in parsing PDF file.

exception pyxpdf.xpdf.XPDFConfigError

Wrong or unsupported xpdf configuration setting.

exception pyxpdf.xpdf.PDFIOError

Error r/w file

exception pyxpdf.xpdf.PDFPermissionError

PDF does not have required permissions or is encrypted.

exception pyxpdf.xpdf.XPDFInternalError

xpdf internal errors.

exception pyxpdf.xpdf.XPDFNotImplementedError

NotImplemented in xpdf sources

SPEED COMPARSION

Thanks to the brilliant [xpdf reader](#) sources and the fact that pyxpdf is written in [cython](#) as Python C-API module makes it much faster than pure python based pdf parsers.

4.1 Text Extraction

Comparing text extraction (while maintaining layout) speed with popular [pdfminer.six](#) module. (python script used - [compare.py](#))

Running Python 3.6.9, gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0, Ubuntu 18.04, on Azure Standard B2ms (2 vcpus, 8 GiB memory) [Intel(R) Xeon(R) Platinum 8171M CPU @ 2.60GHz]

```
'pdfminer_text' took: 0.9271 sec
'pyxpdf_text' took: 0.0424 sec

'pdfminer_text_100mb' took: 7.2833 sec
'pyxpdf_text_100mb' took: 0.3301 sec

'pdfminer_text_500mb' took: 36.5288 sec
'pyxpdf_text_500mb' took: 0.9786 sec
```

Size	pdfminer.six	pyxpdf	times faster
1 MB	0.9271 sec	0.0424 sec	x21
100 MB	7.2833 sec	0.3301 sec	x22
500 MB	36.5288 sec	0.9786 sec	x37

pyxpdf is atleast x20 times faster

CHANGELOG

5.1 0.2.3 (2020-08-31)

- Config: make `Config.cfg_path` public attribute
- Document: add support for `Path` for loading pdf
- pyxpdf_data: add 35 base Postscript fonts from ghostscript

Bugs Fixed

- Fix #9: segfault using `text()`
- Fix #8: add checks for file in `Config.add_font_file()`

5.2 0.2.2 (2020-07-03)

- Config: add function to add missing fonts `Config.add_font_file()`
- Introduce `PDFImage` to represent a PDF Image.
- PDFImageOutput: `get()` returns `PDFImage` instead of Pillow Image

5.3 0.2.1 (2020-06-12)

Bugs Fixed

- fix all direct memory leaks
- Config: fix `Config.text_encoding` setter, encodings with lowercase characters were not able to set.
- fix weird bytes encoding problem in python debug builds

5.4 0.2.0 (2020-06-11)

- Python 2.7 support dropped
- 2 optional dependencies ([Pillow](#), [pyxpdf_data](#)) introduced

New Features

- Introduce (optional) package [pyxpdf_data](#) which add more encoding support.
- API: add specialised classes for pdf outputs, [PDFOutputDevice](#).
 - **TextOutput** - For Text extraction
 - **RawImageOutput** - Render PDF Page as Image
 - **PDFImageOutput** - Extract images from PDF
- Config: add new global settings:
 - `Config.anti_alias`
 - `Config.enable_freetype`
 - `Config.vector_anti_alias`

Bugs Fixed

- pdftotext: extracted text contains clipped text even when explicitly discarding it.
- Config: fix loading of external xdfrc with `Config.load_file()`

5.5 0.1.1 (2020-05-10)

- FIX: default `Config.text_encoding` value i.e UTF-8 does not persist `Config.reset()` and changes to Latin1.
- pdftotext: remove all parameters that change global `Config` properties.

5.6 0.1 (2020-04-20)

Initial stable release.

GLOSSARY

BBox Boundary Box, is the rectangle which enclose content in a PDF page. In `pyxpdf` a BBox is represented as a 4-int tuple of two diagonal vertices of rectangle - (x1, y1, x2, y2)

file-like An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource.

xpdfrc See *xpdfrc Configuration File*

INDICES AND TABLES

- `genindex`
- `search`

A

`add_font_file()` (*pyxpdf.xpdf.Config* method), 33
`anti_alias` (*pyxpdf.xpdf.Config* attribute), 34
`artbox` (*pyxpdf.xpdf.Page* attribute), 11

B

`BBox`, 39
`bbox` (*pyxpdf.xpdf.PDFImage* attribute), 18
`bleedbox` (*pyxpdf.xpdf.Page* attribute), 11
`bpc` (*pyxpdf.xpdf.PDFImage* attribute), 19

C

`colorspace` (*pyxpdf.xpdf.PDFImage* attribute), 19
`components` (*pyxpdf.xpdf.PDFImage* attribute), 19
`compression` (*pyxpdf.xpdf.PDFImage* attribute), 19
`Config` (in module *pyxpdf.xpdf*), 33
`control` (*pyxpdf.xpdf.TextOutput* attribute), 14
`crop_height` (*pyxpdf.xpdf.Page* attribute), 11
`crop_width` (*pyxpdf.xpdf.Page* attribute), 11
`cropbox` (*pyxpdf.xpdf.Page* attribute), 11

D

`doc` (*pyxpdf.xpdf.Page* attribute), 11
`doc` (*pyxpdf.xpdf.TextOutput* attribute), 14
`Document` (class in *pyxpdf.xpdf*), 7

E

`enable_freetype` (*pyxpdf.xpdf.Config* attribute), 34

F

`file-like`, 39
`filename` (*pyxpdf.xpdf.Document* attribute), 8
`find_all_text()` (*pyxpdf.xpdf.Page* method), 11
`find_text()` (*pyxpdf.xpdf.Page* method), 12

G

`get()` (*pyxpdf.xpdf.PDFImageOutput* method), 19
`get()` (*pyxpdf.xpdf.PDFOutputDevice* method), 14
`get()` (*pyxpdf.xpdf.RawImageOutput* method), 17
`get()` (*pyxpdf.xpdf.TextOutput* method), 14
`get_all()` (*pyxpdf.xpdf.TextOutput* method), 14

`get_bytes()` (*pyxpdf.xpdf.TextOutput* method), 15

H

`has_page_labels` (*pyxpdf.xpdf.Document* attribute), 8
`hDPI` (*pyxpdf.xpdf.PDFImage* attribute), 18

I

`image` (*pyxpdf.xpdf.PDFImage* attribute), 18
`index` (*pyxpdf.xpdf.Page* attribute), 11
`info()` (*pyxpdf.xpdf.Document* method), 8
`interpolate` (*pyxpdf.xpdf.PDFImage* attribute), 18
`is_cropped` (*pyxpdf.xpdf.Page* attribute), 12
`is_encrypted` (*pyxpdf.xpdf.Document* attribute), 8
`is_inline` (*pyxpdf.xpdf.PDFImage* attribute), 18
`is_linearized` (*pyxpdf.xpdf.Document* attribute), 8

L

`label` (*pyxpdf.xpdf.Page* attribute), 11
`load_file()` (*pyxpdf.xpdf.Config* method), 33

M

`media_height` (*pyxpdf.xpdf.Page* attribute), 12
`media_width` (*pyxpdf.xpdf.Page* attribute), 12
`mediabox` (*pyxpdf.xpdf.Page* attribute), 12

N

`num_pages` (*pyxpdf.xpdf.Document* attribute), 8

O

`ok_to_add_notes` (*pyxpdf.xpdf.Document* attribute), 8
`ok_to_change` (*pyxpdf.xpdf.Document* attribute), 8
`ok_to_copy` (*pyxpdf.xpdf.Document* attribute), 9
`ok_to_print` (*pyxpdf.xpdf.Document* attribute), 9

P

`Page` (class in *pyxpdf.xpdf*), 10
`page_index` (*pyxpdf.xpdf.PDFImage* attribute), 18
`page_iterator` (class in *pyxpdf.xpdf*), 19
`pdf_version` (*pyxpdf.xpdf.Document* attribute), 9

PDFError, [34](#)
PDFImage (*class in pyxpdf.xpdf*), [18](#)
PDFImageOutput (*class in pyxpdf.xpdf*), [19](#)
PDFIOError, [34](#)
PDFOutputDevice (*class in pyxpdf.xpdf*), [14](#)
PDFPermissionError, [34](#)
PDFSyntaxError, [34](#)

R

RawImageOutput (*class in pyxpdf.xpdf*), [16](#)
reset() (*pyxpdf.xpdf.Config method*), [33](#)
resolution_x (*pyxpdf.xpdf.RawImageOutput attribute*), [17](#)
resolution_y (*pyxpdf.xpdf.RawImageOutput attribute*), [17](#)
rotation (*pyxpdf.xpdf.Page attribute*), [12](#)

S

scale_before_rotation (*pyxpdf.xpdf.RawImageOutput attribute*), [17](#)

T

text() (*pyxpdf.xpdf.Document method*), [9](#)
text() (*pyxpdf.xpdf.Page method*), [12](#)
text_bytes() (*pyxpdf.xpdf.Document method*), [9](#)
text_bytes() (*pyxpdf.xpdf.Page method*), [13](#)
text_encoding (*pyxpdf.xpdf.Config attribute*), [33](#)
text_eol (*pyxpdf.xpdf.Config attribute*), [33](#)
text_keep_tiny (*pyxpdf.xpdf.Config attribute*), [34](#)
text_page_breaks (*pyxpdf.xpdf.Config attribute*), [34](#)
TextControl (*class in pyxpdf.xpdf*), [15](#)
TextOutput (*class in pyxpdf.xpdf*), [14](#)
trimbox (*pyxpdf.xpdf.Page attribute*), [13](#)

U

use_cropbox (*pyxpdf.xpdf.RawImageOutput attribute*), [17](#)

V

vDPI (*pyxpdf.xpdf.PDFImage attribute*), [19](#)
vector_anti_alias (*pyxpdf.xpdf.Config attribute*), [34](#)

X

xmp_metadata() (*pyxpdf.xpdf.Document method*), [10](#)
XPDFConfigError, [34](#)
XPDFError, [34](#)
XPDFInternalError, [34](#)
XPDFNotImplementedError, [34](#)
xpdfrc, [39](#)