
pdfminer.six

Release 20200517

Yusuke Shinyama, Philippe Guglielmetti & Pieter Marsman

Oct 17, 2021

CONTENTS

1	Content	3
1.1	Tutorials	3
1.2	How-to guides	6
1.3	Topics	6
1.4	API Reference	8
2	Features	15
3	Installation instructions	17
4	Contributing	19
	Index	21

Pdfminer.six is a python package for extracting information from PDF documents.

Check out the source on [github](#).

CONTENT

This documentation is organized into four sections (according to the [Divio documentation system](#)). The *Tutorials* section helps you setup and use pdfminer.six for the first time. Read this section if this is your first time working with pdfminer.six. The *How-to guides* offers specific recipes for solving common problems. Take a look at the *Topics* if you want more background information on how pdfminer.six works internally. The *API Reference* provides detailed api documentation for all the common classes and functions in pdfminer.six.

1.1 Tutorials

Tutorials help you get started with specific parts of pdfminer.six.

1.1.1 Install pdfminer.six as a Python package

To use pdfminer.six for the first time, you need to install the Python package in your Python environment.

This tutorial requires you to have a system with a working Python and pip installation. If you don't have one and don't know how to install it, take a look at [The Hitchhiker's Guide to Python!](#).

Install using pip

Run the following command on the commandline to install pdfminer.six as a Python package:

```
pip install pdfminer.six
```

Test pdfminer.six installation

You can test the pdfminer.six installation by importing it in Python.

Open an interactive Python session from the commandline import pdfminer .six:

```
>>> import pdfminer
>>> print(pdfminer.__version__)
'<installed version>'
```

Now you can use pdfminer.six as a Python package. But pdfminer.six also comes with a couple of useful commandline tools. To test if these tools are correctly installed, run the following on your commandline:

```
$ pdf2txt.py --version
pdfminer.six <installed version>
```

1.1.2 Extract text from a PDF using the commandline

pdfminer.six has several tools that can be used from the command line. The command-line tools are aimed at users that occasionally want to extract text from a pdf.

Take a look at the high-level or composable interface if you want to use pdfminer.six programmatically.

Examples

pdf2txt.py

```
$ python tools/pdf2txt.py example.pdf
all the text from the pdf appears on the command line
```

The *pdf2txt.py* tool extracts all the text from a PDF. It uses layout analysis with sensible defaults to order and group the text in a sensible way.

dumppdf.py

```
$ python tools/dumppdf.py -a example.pdf
<pdf><object id="1">
...
</object>
...
</pdf>
```

The *dumppdf.py* tool can be used to extract the internal structure from a PDF. This tool is primarily for debugging purposes, but that can be useful to anybody working with PDF's.

1.1.3 Extract text from a PDF using Python

The high-level API can be used to do common tasks.

The most simple way to extract text from a PDF is to use *extract_text*:

```
>>> text = extract_text('samples/simple1.pdf')
>>> print(repr(text))
'Hello \n\nWorld\n\nHello \n\nWorld\n\nH e l l o  \n\nW o r l d\n\nH e l l o  \n\nW o r
↳ l d\n\n\x0c'
>>> print(text)
...
Hello

World

Hello

World

H e l l o
```

(continues on next page)

(continued from previous page)

```

W o r l d

H e l l o

W o r l d

```

To read text from a PDF and print it on the command line:

```

>>> if sys.version_info > (3, 0):
...     from io import StringIO
... else:
...     from io import BytesIO as StringIO
>>> output_string = StringIO()
>>> with open('samples/simple1.pdf', 'rb') as fin:
...     extract_text_to_fp(fin, output_string)
>>> print(output_string.getvalue().strip())
Hello WorldHello WorldHello WorldHello World

```

Or to convert it to html and use layout analysis:

```

>>> if sys.version_info > (3, 0):
...     from io import StringIO
... else:
...     from io import BytesIO as StringIO
>>> from pdfminer.layout import LAParams
>>> output_string = StringIO()
>>> with open('samples/simple1.pdf', 'rb') as fin:
...     extract_text_to_fp(fin, output_string, laparams=LAParams(),
...                        output_type='html', codec=None)

```

1.1.4 Extract text from a PDF using Python - part 2

The command line tools and the high-level API are just shortcuts for often used combinations of pdfminer.six components. You can use these components to modify pdfminer.six to your own needs.

For example, to extract the text from a PDF file and save it in a python variable:

```

from io import StringIO

from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfparser import PDFParser

output_string = StringIO()
with open('samples/simple1.pdf', 'rb') as in_file:
    parser = PDFParser(in_file)
    doc = PDFDocument(parser)
    rsrcmgr = PDFResourceManager()
    device = TextConverter(rsrcmgr, output_string, laparams=LAParams())

```

(continues on next page)

(continued from previous page)

```
interpreter = PDFPageInterpreter(rsrcmgr, device)
for page in PDFPage.create_pages(doc):
    interpreter.process_page(page)

print(output_string.getvalue())
```

1.2 How-to guides

How-to guides help you to solve specific problems with pdfminer.six.

1.2.1 How to extract images from a PDF

Before you start, make sure you have *installed pdfminer.six*. The second thing you need is a PDF with images. If you don't have one, you can download [this research paper](https://www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/parkhi12a.pdf) with images of cats and dogs and save it as *example.pdf*:

```
$ curl https://www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/parkhi12a.pdf --
  ↪output example.pdf
```

Then run the *pdf2txt* command:

```
$ pdf2txt.py example.pdf --output-dir cats-and-dogs
```

This command extracts all the images from the PDF and saves them into the *cats-and-dogs* directory.

1.3 Topics

1.3.1 Converting a PDF file to text

Most PDF files look like they contain well structured text. But the reality is that a PDF file does not contain anything that resembles a paragraphs, sentences or even words. When it comes to text, a PDF file is only aware of the characters and their placement.

This makes extracting meaningful pieces of text from PDF's files difficult. The characters that compose a paragraph are no different from those that compose the table, the page footer or the description of a figure. Unlike other documents formats, like a *.txt* file or a word document, the PDF format does not contain a stream of text.

A PDF document does consists of a collection of objects that together describe the appearance of one or more pages, possibly accompanied by additional interactive elements and higher-level application data. A PDF file contains the objects making up a PDF document along with associated structural information, all represented as a single self-contained sequence of bytes.¹

¹ Adobe System Inc. (2007). *Pdf reference: Adobe portable document format, version 1.7*.

Layout analysis algorithm

PDFMiner attempts to reconstruct some of those structures by using heuristics on the positioning of characters. This works well for sentences and paragraphs because meaningful groups of nearby characters can be made.

The layout analysis consist of three different stages: it groups characters into words and lines, then it groups lines into boxes and finally it groups textboxes hierarchically. These stages are discussed in the following sections. The resulting output of the layout analysis is an ordered hierarchy of layout objects on a PDF page.

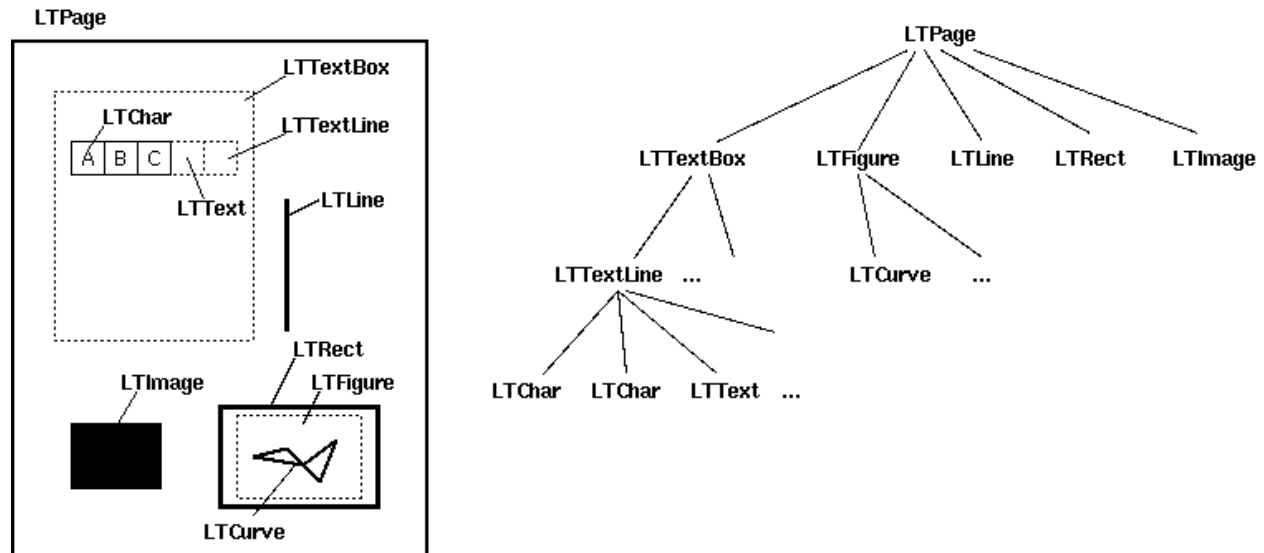


Fig. 1: The output of the layout analysis is a hierarchy of layout objects.

The output of the layout analysis heavily depends on a couple of parameters. All these parameters are part of the *LAParams* class.

Grouping characters into words and lines

The first step in going from characters to text is to group characters in a meaningful way. Each character has an x-coordinate and a y-coordinate for its bottom-left corner and upper-right corner, i.e. its bounding box. Pdfminer .six uses these bounding boxes to decide which characters belong together.

Characters that are both horizontally and vertically close are grouped onto one line. How close they should be is determined by the *char_margin* (M in figure) and the *line_overlap* (not in figure) parameter. The horizontal *distance* between the bounding boxes of two characters should be smaller than the *char_margin* and the vertical *overlap* between the bounding boxes should be smaller than the *line_overlap*.

The values of *char_margin* and *line_overlap* are relative to the size of the bounding boxes of the characters. The *char_margin* is relative to the maximum width of either one of the bounding boxes, and the *line_overlap* is relative to the minimum height of either one of the bounding boxes.

Spaces need to be inserted between characters because the PDF format has no notion of the space character. A space is inserted if the characters are further apart than the *word_margin* (W in the figure). The *word_margin* is relative to the maximum width or height of the new character. Having a smaller *word_margin* creates smaller words. Note that the *word_margin* should at least be smaller than the *char_margin* otherwise none of the characters will be separated by a space.

The result of this stage is a list of lines. Each line consists a list of characters. These characters either original *LTChar* characters that originate from the PDF file, or inserted *LAnno* characters that represent spaces between words or

newlines at the end of each line.

Grouping lines into boxes

The second step is grouping lines in a meaningful way. Each line has a bounding box that is determined by the bounding boxes of the characters that it contains. Like grouping characters, pdfminer.six uses the bounding boxes to group the lines.

Lines that are both horizontally overlapping and vertically close are grouped. How vertically close the lines should be is determined by the *line_margin*. This margin is specified relative to the height of the bounding box. Lines are close if the gap between the tops (see L_1 in the figure) and bottoms (see L_2 in the figure) of the bounding boxes are closer together than the absolute line margin, i.e. the *line_margin* multiplied by the height of the bounding box.

The result of this stage is a list of text boxes. Each box consist of a list of lines.

Grouping textboxes hierarchically

the last step is to group the text boxes in a meaningful way. This step repeatedly merges the two text boxes that are closest to each other.

The closeness of bounding boxes is computed as the area that is between the two text boxes (the blue area in the figure). In other words, it is the area of the bounding box that surrounds both lines, minus the area of the bounding boxes of the individual lines.

Working with rotated characters

The algorithm described above assumes that all characters have the same orientation. However, any writing direction is possible in a PDF. To accommodate for this, pdfminer.six allows to detect vertical writing with the *detect_vertical* parameter. This will apply all the grouping steps as if the pdf was rotated 90 (or 270) degrees

References

1.4 API Reference

1.4.1 Command-line API

pdf2txt.py

A command line tool for extracting text and images from PDF and output it to plain text, html, xml or tags.

```
usage: python tools/pdf2txt.py [-h] [--version] [--debug] [--disable-caching]
                             [--page-numbers PAGE_NUMBERS [PAGE_NUMBERS ...]]
                             [--pagenos PAGENOS] [--maxpages MAXPAGES]
                             [--password PASSWORD] [--rotation ROTATION]
                             [--no-laparams] [--detect-vertical]
                             [--char-margin CHAR_MARGIN]
                             [--word-margin WORD_MARGIN]
                             [--line-margin LINE_MARGIN]
                             [--boxes-flow BOXES_FLOW] [--all-texts]
```

(continues on next page)

(continued from previous page)

```

[--outfile OUTFILE] [--output_type OUTPUT_TYPE]
[--codec CODEC] [--output-dir OUTPUT_DIR]
[--layoutmode LAYOUTMODE] [--scale SCALE]
[--strip-control]
files [files ...]

```

Positional Arguments

files One or more paths to PDF files.

options

--version, -v show program's version number and exit

--debug, -d Use debug logging level.
Default: False

--disable-caching, -C If caching or resources, such as fonts, should be disabled.
Default: False

Parser

Used during PDF parsing

--page-numbers A space-separated list of page numbers to parse.

--pagenos, -p A comma-separated list of page numbers to parse. Included for legacy applications, use `--page-numbers` for more idiomatic argument entry.

--maxpages, -m The maximum number of pages to parse.
Default: 0

--password, -P The password to use for decrypting PDF file.
Default: ""

--rotation, -R The number of degrees to rotate the PDF before other types of processing.
Default: 0

Layout analysis

Used during layout analysis.

--no-laparams, -n If layout analysis parameters should be ignored.
Default: False

--detect-vertical, -V If vertical text should be considered during layout analysis
Default: False

- char-margin, -M** If two characters are closer together than this margin they are considered to be part of the same line. The margin is specified relative to the width of the character.
Default: 2.0
- word-margin, -W** If two characters on the same line are further apart than this margin then they are considered to be two separate words, and an intermediate space will be added for readability. The margin is specified relative to the width of the character.
Default: 0.1
- line-margin, -L** If two lines are close together they are considered to be part of the same paragraph. The margin is specified relative to the height of a line.
Default: 0.5
- boxes-flow, -F** Specifies how much a horizontal and vertical position of a text matters when determining the order of lines. The value should be within the range of -1.0 (only horizontal position matters) to +1.0 (only vertical position matters). You can also pass *None* to disable advanced layout analysis, and instead return text based on the position of the bottom left corner of the text box.
Default: 0.5
- all-texts, -A** If layout analysis should be performed on text in figures.
Default: False

Output

Used during output generation.

- outfile, -o** Path to file where output is written. Or “-” (default) to write to stdout.
Default: “-”
- output_type, -t** Type of output to generate {text,html,xml,tag}.
Default: “text”
- codec, -c** Text encoding to use in output file.
Default: “utf-8”
- output-dir, -O** The output directory to put extracted images in. If not given, images are not extracted.
- layoutmode, -Y** Type of layout to use when generating html {normal,exact,loose}. If normal, each line is positioned separately in the html. If exact, each character is positioned separately in the html. If loose, same result as normal but with an additional newline after each text line. Only used when output_type is html.
Default: “normal”
- scale, -s** The amount of zoom to use when generating html file. Only used when output_type is html.
Default: 1.0
- strip-control, -S** Remove control statement from text. Only used when output_type is xml.
Default: False

dumppdf.py

Extract pdf structure in XML format

```
usage: python tools/dumppdf.py [-h] [--version] [--debug]
                             [--extract-toc | --extract-embedded EXTRACT_EMBEDDED]
                             [--page-numbers PAGE_NUMBERS [PAGE_NUMBERS ...]]
                             [--pagenos PAGENOS] [--objects OBJECTS] [--all]
                             [--password PASSWORD] [--outfile OUTFILE]
                             [--raw-stream | --binary-stream | --text-stream]
                             files [files ...]
```

Positional Arguments

files	One or more paths to PDF files.
--------------	---------------------------------

options

--version, -v	show program's version number and exit
--debug, -d	Use debug logging level. Default: False
--extract-toc, -T	Extract structure of outline Default: False
--extract-embedded, -E	Extract embedded files

Parser

Used during PDF parsing

--page-numbers	A space-separated list of page numbers to parse.
--pagenos, -p	A comma-separated list of page numbers to parse. Included for legacy applications, use --page-numbers for more idiomatic argument entry.
--objects, -i	Comma separated list of object numbers to extract
--all, -a	If the structure of all objects should be extracted Default: False
--password, -P	The password to use for decrypting PDF file. Default: ""

Output

Used during output generation.

- | | |
|----------------------------|--|
| --outfile, -o | Path to file where output is written. Or “-” (default) to write to stdout.
Default: “-” |
| --raw-stream, -r | Write stream objects without encoding
Default: False |
| --binary-stream, -b | Write stream objects with binary encoding
Default: False |
| --text-stream, -t | Write stream objects as plain text
Default: False |

1.4.2 High-level functions API

extract_text

```
pdfminer.high_level.extract_text(pdf_file, password="", page_numbers=None, maxpages=0, caching=True,
                                codec='utf-8', laparams=None)
```

Parse and return the text contained in a PDF file.

Parameters

- **pdf_file** – Either a file path or a file-like object for the PDF file to be worked on.
- **password** – For encrypted PDFs, the password to decrypt.
- **page_numbers** – List of zero-indexed page numbers to extract.
- **maxpages** – The maximum number of pages to parse
- **caching** – If resources should be cached
- **codec** – Text decoding codec
- **laparams** – An LAParams object from pdfminer.layout. If None, uses some default settings that often work well.

Returns a string containing all of the text extracted.

extract_text_to_fp

```
pdfminer.high_level.extract_text_to_fp(inf, outfp, output_type='text', codec='utf-8', laparams=None,
                                       maxpages=0, page_numbers=None, password="", scale=1.0,
                                       rotation=0, layoutmode='normal', output_dir=None,
                                       strip_control=False, debug=False, disable_caching=False,
                                       **kwargs)
```

Parses text from inf-file and writes to outfp file-like object.

Takes loads of optional arguments but the defaults are somewhat sane. Beware laparams: Including an empty LAParams is not the same as passing None!

Parameters

- **inf** – a file-like object to read PDF structure from, such as a file handler (using the builtin *open()* function) or a *BytesIO*.
- **outfp** – a file-like object to write the text to.
- **output_type** – May be ‘text’, ‘xml’, ‘html’, ‘tag’. Only ‘text’ works properly.
- **codec** – Text decoding codec
- **laparams** – An LParams object from pdfminer.layout. Default is None but may not layout correctly.
- **maxpages** – How many pages to stop parsing after
- **page_numbers** – zero-indexed page numbers to operate on.
- **password** – For encrypted PDFs, the password to decrypt.
- **scale** – Scale factor
- **rotation** – Rotation factor
- **layoutmode** – Default is ‘normal’, see pdfminer.converter.HTMLConverter
- **output_dir** – If given, creates an ImageWriter for extracted images.
- **strip_control** – Does what it says on the tin
- **debug** – Output more logging data
- **disable_caching** – Does what it says on the tin
- **other** –

Returns nothing, acting as it does on two streams. Use StringIO to get strings.

extract_pages

```
pdfminer.high_level.extract_pages(pdf_file, password="", page_numbers=None, maxpages=0,
                                caching=True, laparams=None)
```

Extract and yield LTPage objects

Parameters

- **pdf_file** – Either a file path or a file-like object for the PDF file to be worked on.
- **password** – For encrypted PDFs, the password to decrypt.
- **page_numbers** – List of zero-indexed page numbers to extract.
- **maxpages** – The maximum number of pages to parse
- **caching** – If resources should be cached
- **laparams** – An LParams object from pdfminer.layout. If None, uses some default settings that often work well.

Returns

1.4.3 Composable API

LAParams

```
class pdfminer.layout.LAParams(line_overlap=0.5, char_margin=2.0, line_margin=0.5, word_margin=0.1,  
                               boxes_flow=0.5, detect_vertical=False, all_texts=False)
```

Parameters for layout analysis

Parameters

- **line_overlap** – If two characters have more overlap than this they are considered to be on the same line. The overlap is specified relative to the minimum height of both characters.
- **char_margin** – If two characters are closer together than this margin they are considered part of the same line. The margin is specified relative to the width of the character.
- **word_margin** – If two characters on the same line are further apart than this margin then they are considered to be two separate words, and an intermediate space will be added for readability. The margin is specified relative to the width of the character.
- **line_margin** – If two lines are close together they are considered to be part of the same paragraph. The margin is specified relative to the height of a line.
- **boxes_flow** – Specifies how much a horizontal and vertical position of a text matters when determining the order of text boxes. The value should be within the range of -1.0 (only horizontal position matters) to +1.0 (only vertical position matters). You can also pass *None* to disable advanced layout analysis, and instead return text based on the position of the bottom left corner of the text box.
- **detect_vertical** – If vertical text should be considered during layout analysis
- **all_texts** – If layout analysis should be performed on text in figures.

Todo:

- *PDFDevice*
 - *TextConverter*
 - *PDFPageAggregator*
- *PDFPageInterpreter*

FEATURES

- Parse all objects from a PDF document into Python objects.
- Analyze and group text in a human-readable way.
- Extract text, images (JPG, JBIG2 and Bitmaps), table-of-contents, tagged contents and more.
- Support for (almost all) features from the PDF-1.7 specification
- Support for Chinese, Japanese and Korean CJK languages as well as vertical writing.
- Support for various font types (Type1, TrueType, Type3, and CID).
- Support for RC4 and AES encryption.

INSTALLATION INSTRUCTIONS

Before using it, you must install it using Python 3.4 or newer.

```
$ pip install pdminer.six
```


CONTRIBUTING

We welcome any contributors to pdfminer.six! But, before doing anything, take a look at the [contribution guide](#).

INDEX

E

`extract_pages()` (*in module `pdfminer.high_level`*), [13](#)
`extract_text()` (*in module `pdfminer.high_level`*), [12](#)
`extract_text_to_fp()` (*in module `pdfminer.high_level`*), [12](#)

L

`LAParams` (*class in `pdfminer.layout`*), [14](#)