pdfminer

Release 0.0.1

Levia3

Nov 18, 2017

Contents

CHAPTER 1

PDFMiner

Python PDF parser and analyzer

Homepage *Recent Changes* PDFMiner API

1.1 What's It?

PDFMiner is a tool for extracting information from PDF documents. Unlike other PDF-related tools, it focuses entirely on getting and analyzing text data. PDFMiner allows one to obtain the exact location of text in a page, as well as other information such as fonts or lines. It includes a PDF converter that can transform PDF files into other text formats (such as HTML). It has an extensible PDF parser that can be used for other purposes than text analysis.

1.1.1 Features

- Written entirely in Python. (for version 2.6 or newer)
- Parse, analyze, and convert PDF documents.
- PDF-1.7 specification support. (well, almost)
- CJK languages and vertical writing scripts support.
- Various font types (Type1, TrueType, Type3, and CID) support.
- Basic encryption (RC4) support.
- PDF to HTML conversion (with a sample converter web app).
- Outline (TOC) extraction.
- Tagged contents extraction.
- Reconstruct the original layout by grouping text chunks.

PDFMiner is about 20 times slower than other C/C++-based counterparts such as XPdf.

Online Demo: (pdf -> html conversion webapp)
http://pdf2html.tabesugi.net:8080/

1.1.2 Download

Source distribution:
http://pypi.python.org/pypi/pdfminer/

github:
https://github.com/euske/pdfminer/

1.1.3 Where to Ask

Questions and comments:
http://groups.google.com/group/pdfminer-users/

1.2 How to Install

1. Install Python 2.6 or newer. (**Python 3 is not supported.**)
2. Download the *PDFMiner source*.
3. Unpack it.
4. Run `setup.py` to install:

   ```
   # python setup.py install
   ```

5. Do the following test:

   ```
   $ pdf2txt.py samples/simple1.pdf
   Hello

   World

   Hello

   World

   H e l l o

   W o r l d

   H e l l o

   W o r l d
   ```

6. Done!

1.2.1 For CJK languages

In order to process CJK languages, you need an additional step to take during installation:

```
# make cmap
python tools/conv_cmap.py pdfminer/cmap Adobe-CNS1 cmaprsrc/cid2code_Adobe_
↪CNS1.txt
reading 'cmaprsrc/cid2code_Adobe_CNS1.txt'...
writing 'CNS1_H.py'...
...
(this may take several minutes)

# python setup.py install
```

On Windows machines which don't have `make` command, paste the following commands on a command line prompt:

```
mkdir pdfminer\cmap
python tools\conv_cmap.py -c B5=cp950 -c UniCNS-UTF8=utf-8 pdfminer\cmap␣
↪Adobe-CNS1 cmaprsrc\cid2code_Adobe_CNS1.txt
python tools\conv_cmap.py -c GBK-EUC=cp936 -c UniGB-UTF8=utf-8 pdfminer\cmap␣
↪Adobe-GB1 cmaprsrc\cid2code_Adobe_GB1.txt
python tools\conv_cmap.py -c RKSJ=cp932 -c EUC=euc-jp -c UniJIS-UTF8=utf-8␣
↪pdfminer\cmap Adobe-Japan1 cmaprsrc\cid2code_Adobe_Japan1.txt
python tools\conv_cmap.py -c KSC-EUC=euc-kr -c KSC-Johab=johab -c KSCms-
↪UHC=cp949 -c UniKS-UTF8=utf-8 pdfminer\cmap Adobe-Korea1 cmaprsrc\cid2code_
↪Adobe_Korea1.txt
python setup.py install
```

1.3 Command Line Tools

PDFMiner comes with two handy tools: `pdf2txt.py` and `dumppdf.py`.

1.3.1 pdf2txt.py

`pdf2txt.py` extracts text contents from a PDF file. It extracts all the text that are to be rendered programmatically, i.e. text represented as ASCII or Unicode strings. It cannot recognize text drawn as images that would require optical character recognition. It also extracts the corresponding locations, font names, font sizes, writing direction (horizontal or vertical) for each text portion. You need to provide a password for protected PDF documents when its access is restricted. You cannot extract any text from a PDF document which does not have extraction permission.

Note: Not all characters in a PDF can be safely converted to Unicode.

Examples

```
$ pdf2txt.py -o output.html samples/naacl06-shinyama.pdf
(extract text as an HTML file whose filename is output.html)

$ pdf2txt.py -V -c euc-jp -o output.html samples/jo.pdf
(extract a Japanese HTML file in vertical writing, CMap is required)

$ pdf2txt.py -P mypassword -o output.txt secret.pdf
(extract a text from an encrypted PDF file)
```

Options

`-o filename`

Specifies the output file name. By default, it prints the extracted contents to stdout in text format.

`-p pageno[,pageno,...]`

Specifies the comma-separated list of the page numbers to be extracted. Page numbers start at one. By default, it extracts text from all the pages.

`-c codec`

Specifies the output codec.

`-t type`

Specifies the output format. The following formats are currently supported.

- `text` : TEXT format. (Default)
- `html` : HTML format. Not recommended for extraction purposes because the markup is messy.
- `xml` : XML format. Provides the most information.
- `tag` : "Tagged PDF" format. A tagged PDF has its own contents annotated with HTML-like tags. pdf2txt tries to extract its content streams rather than inferring its text locations. Tags used here are defined in the PDF specification (See §10.7 "*Tagged PDF*").

`-I image_directory`

Specifies the output directory for image extraction. Currently only JPEG images are supported.

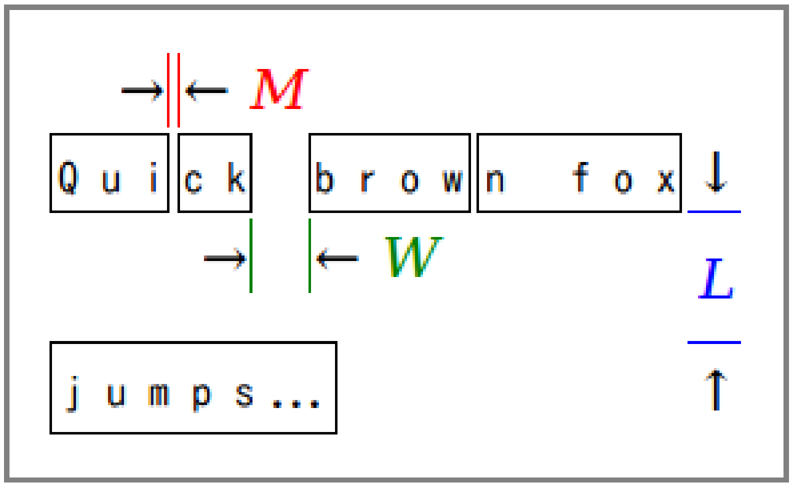
`-M char_margin`

`-L line_margin`

`-W word_margin`

These are the parameters used for layout analysis. In an actual PDF file, text portions might be split into several chunks in the middle of its running, depending on the authoring software. Therefore, text extraction needs to splice text chunks. In the figure below, two text chunks whose distance is closer than the *char_margin* (shown as *M*) is considered continuous and get grouped into one. Also, two lines whose distance is closer than the *line_margin* (*L*) is grouped as a text box, which is a rectangular area that contains a "cluster" of text portions. Furthermore, it may be required to insert blank characters (spaces) as necessary if the distance between two words is greater than the *word_margin* (*W*), as a blank between words might not be represented as a space, but indicated by the positioning of each word.

Each value is specified not as an actual length, but as a proportion of the length to the size of each character in question. The default values are M = 2.0, L = 0.5, and W = 0.1, respectively.



`-F boxes_flow`

Specifies how much a horizontal and vertical position of a text matters when determining a text order. The value should be within the range of -1.0 (only horizontal position matters) to +1.0 (only vertical position matters). The default value is 0.5.

`-C`

Suppress object caching. This will reduce the memory consumption but also slows down the process.

`-n`

Suppress layout analysis.

`-A`

Forces to perform layout analysis for all the text strings, including text contained in figures.

`-V`

Allows vertical writing detection.

`-Y layout_mode`

Specifies how the page layout should be preserved. (Currently only applies to HTML format.)

- `exact` : preserve the exact location of each individual character (a large and messy HTML).
- `normal` : preserve the location and line breaks in each text block. (Default)
- `loose` : preserve the overall location of each text block.

`-E extractdir`

Specifies the extraction directory of embedded files.

`-s scale`

Specifies the output scale. Can be used in HTML format only.

`-m maxpages`

Specifies the maximum number of pages to extract. By default, it extracts all the pages in a document.

`-P password`

Provides the user password to access PDF contents.

`-d`

Increases the debug level.

1.3.2 dumppdf.py

`dumppdf.py` dumps the internal contents of a PDF file in pseudo-XML format. This program is primarily for debugging purposes, but it's also possible to extract some meaningful contents (such as images).

Examples

```
$ dumppdf.py -a foo.pdf
(dump all the headers and contents, except stream objects)

$ dumppdf.py -T foo.pdf
(dump the table of contents)

$ dumppdf.py -r -i6 foo.pdf > pic.jpeg
(extract a JPEG image)
```

Options

`-a`

Instructs to dump all the objects. By default, it only prints the document trailer (like a header).

`-i objno,objno, ...`

Specifies PDF object IDs to display. Comma-separated IDs, or multiple `-i` options are accepted.

`-p pageno,pageno, ...`

Specifies the page number to be extracted. Comma-separated page numbers, or multiple `-p` options are accepted. Note that page numbers start at one, not zero.

`-r` (raw)

`-b` (binary)

`-t` (text)

Specifies the output format of stream contents. Because the contents of stream objects can be very large, they are omitted when none of the options above is specified.

With `-r` option, the "raw" stream contents are dumped without decompression. With `-b` option, the decompressed contents are dumped as a binary blob. With `-t` option, the decompressed contents are dumped in a text format, similar to `repr()` manner. When `-r` or `-b` option is given, no stream header is displayed for the ease of saving it to a file.

`-T`

Shows the table of contents.

`-E directory`

Extracts embedded files from the pdf into the given directory.

`-P password`

Provides the user password to access PDF contents.

`-d`

Increases the debug level.

1.4 Changes

- 2014/03/28: Further bugfixes.
- 2014/03/24: Bugfixes and improvements for fauly PDFs. API changes:
  - `PDFDocument.initialize()` method is removed and no longer needed. A password is given as an argument of a PDFDocument constructor.
- 2013/11/13: Bugfixes and minor improvements. As of November 2013, there were a few changes made to the PDFMiner API prior to October 2013. This is the result of code restructuring. Here is a list of the changes:
  - `PDFDocument` class is moved to `pdfdocument.py`.
  - `PDFDocument` class now takes a `PDFParser` object as an argument.
  - `PDFDocument.set_parser()` and `PDFParser.set_document()` is removed.
  - `PDFPage` class is moved to `pdfpage.py`.
  - `process_pdf` function is implemented as `PDFPage.get_pages`.
- 2013/10/22: Sudden resurge of interests. API changes. Incorporated a lot of patches and robust handling of broken PDFs.
- 2011/05/15: Speed improvements for layout analysis.
- 2011/05/15: API changes. `LTText.get_text()` is added.
- 2011/04/20: API changes. LTPolygon class was renamed as LTCurve.
- 2011/04/20: LTLine now represents horizontal/vertical lines only. Thanks to Koji Nakagawa.
- 2011/03/07: Documentation improvements by Jakub Wilk. Memory usage patch by Jonathan Hunt.
- 2011/02/27: Bugfixes and layout analysis improvements. Thanks to fujimoto.report.
- 2010/12/26: A couple of bugfixes and minor improvements. Thanks to Kevin Brubeck Unhammer and Daniel Gerber.
- 2010/10/17: A couple of bugfixes and minor improvements. Thanks to standardabweichung and Alastair Irving.
- 2010/09/07: A minor bugfix. Thanks to Alexander Garden.
- 2010/08/29: A couple of bugfixes. Thanks to Sahan Malagi, pk, and Humberto Pereira.
- 2010/07/06: Minor bugfixes. Thanks to Federico Brega.
- 2010/06/13: Bugfixes and improvements on CMap data compression. Thanks to Jakub Wilk.
- 2010/04/24: Bugfixes and improvements on TOC extraction. Thanks to Jose Maria.
- 2010/03/26: Bugfixes. Thanks to Brian Berry and Lubos Pintes.
- 2010/03/22: Improved layout analysis. Added regression tests.
- 2010/03/12: A couple of bugfixes. Thanks to Sean Manefield.
- 2010/02/27: Changed the way of internal layout handling. (LTTextItem -> LTChar)
- 2010/02/15: Several bugfixes. Thanks to Sean.

- 2010/02/13: Bugfix and enhancement. Thanks to André Auzi.
- 2010/02/07: Several bugfixes. Thanks to Hiroshi Manabe.
- 2010/01/31: JPEG image extraction supported. Page rotation bug fixed.
- 2010/01/04: Python 2.6 warning removal. More doctest conversion.
- 2010/01/01: CMap bug fix. Thanks to Winfried Plappert.
- 2009/12/24: RunLengthDecode filter added. Thanks to Troy Bollinger.
- 2009/12/20: Experimental polygon shape extraction added. Thanks to Yusuf Dewaswala for reporting.
- 2009/12/19: CMap resources are now the part of the package. Thanks to Adobe for open-sourcing them.
- 2009/11/29: Password encryption bug fixed. Thanks to Yannick Gingras.
- 2009/10/31: SGML output format is changed and renamed as XML.
- 2009/10/24: Charspace bug fixed. Adjusted for 4-space indentation.
- 2009/10/04: Another matrix operation bug fixed. Thanks to Vitaly Sedelnik.
- 2009/09/12: Fixed rectangle handling. Able to extract image boundaries.
- 2009/08/30: Fixed page rotation handling.
- 2009/08/26: Fixed zlib decoding bug. Thanks to Shon Urbas.
- 2009/08/24: Fixed a bug in character placing. Thanks to Pawan Jain.
- 2009/07/21: Improvement in layout analysis.
- 2009/07/11: Improvement in layout analysis. Thanks to Lubos Pintes.
- 2009/05/17: Bugfixes, massive code restructuring, and simple graphic element support added. setup.py is supported.
- 2009/03/30: Text output mode added.
- 2009/03/25: Encoding problems fixed. Word splitting option added.
- 2009/02/28: Robust handling of corrupted PDFs. Thanks to Troy Bollinger.
- 2009/02/01: Various bugfixes. Thanks to Hiroshi Manabe.
- 2009/01/17: Handling a trailer correctly that contains both /XrefStm and /Prev entries.
- 2009/01/10: Handling Type3 font metrics correctly.
- 2008/12/28: Better handling of word spacing. Thanks to Christian Nentwich.
- 2008/09/06: A sample pdf2html webapp added.
- 2008/08/30: ASCII85 encoding filter support.
- 2008/07/27: Tagged contents extraction support.
- 2008/07/10: Outline (TOC) extraction support.
- 2008/06/29: HTML output added. Reorganized the directory structure.
- 2008/04/29: Bugfix for Win32. Thanks to Chris Clark.
- 2008/04/27: Basic encryption and LZW decoding support added.
- 2008/01/07: Several bugfixes. Thanks to Nick Fabry for his vast contribution.
- 2007/12/31: Initial release.

- 2004/12/24: Start writing the code out of boredom...

1.5 TODO

- PEP-8 and PEP-257 conformance.
- Better documentation.
- Better text extraction / layout analysis. (writing mode detection, Type1 font file analysis, etc.)
- Crypt stream filter support. (More sample documents are needed!)

1.6 Related Projects

- pyPdf
- xpdf
- pdfbox
- mupdf

1.7 Terms and Conditions

(This is so-called MIT/X License)

Copyright (c) 2004-2013 Yusuke Shinyama <yusuke at cs dot nyu dot edu>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Yusuke Shinyama (yusuke at cs dot nyu dot edu)

[Back to PDFMiner]

CHAPTER 2

Programming with PDFMiner

This page explains how to use PDFMiner as a library from other applications.

2.1 Overview

PDF is evil. Although it is called a PDF "document", it's nothing like Word or HTML document. PDF is more like a graphic representation. PDF contents are just a bunch of instructions that tell how to place the stuff at each exact position on a display or paper. In most cases, it has no logical structure such as sentences or paragraphs and it cannot adapt itself when the paper size changes. PDFMiner attempts to reconstruct some of those structures by guessing from its positioning, but there's nothing guaranteed to work. Ugly, I know. Again, PDF is evil.

[More technical details about the internal structure of PDF: "How to Extract Text Contents from PDF Manually" (part1) (part2) (part3)]

Because a PDF file has such a big and complex structure, parsing a PDF file as a whole is time and memory consuming. However, not every part is needed for most PDF processing tasks. Therefore PDFMiner takes a strategy of lazy parsing, which is to parse the stuff only when it's necessary. To parse PDF files, you need to use at least two classes: `PDFParser` and `PDFDocument`. These two objects are associated with each other. `PDFParser` fetches data from a file, and `PDFDocument` stores it. You'll also need `PDFPageInterpreter` to process the page contents and `PDFDevice` to translate it to whatever you need. `PDFResourceManager` is used to store shared resources such as fonts or images.

Figure 1 shows the relationship between the classes in PDFMiner.

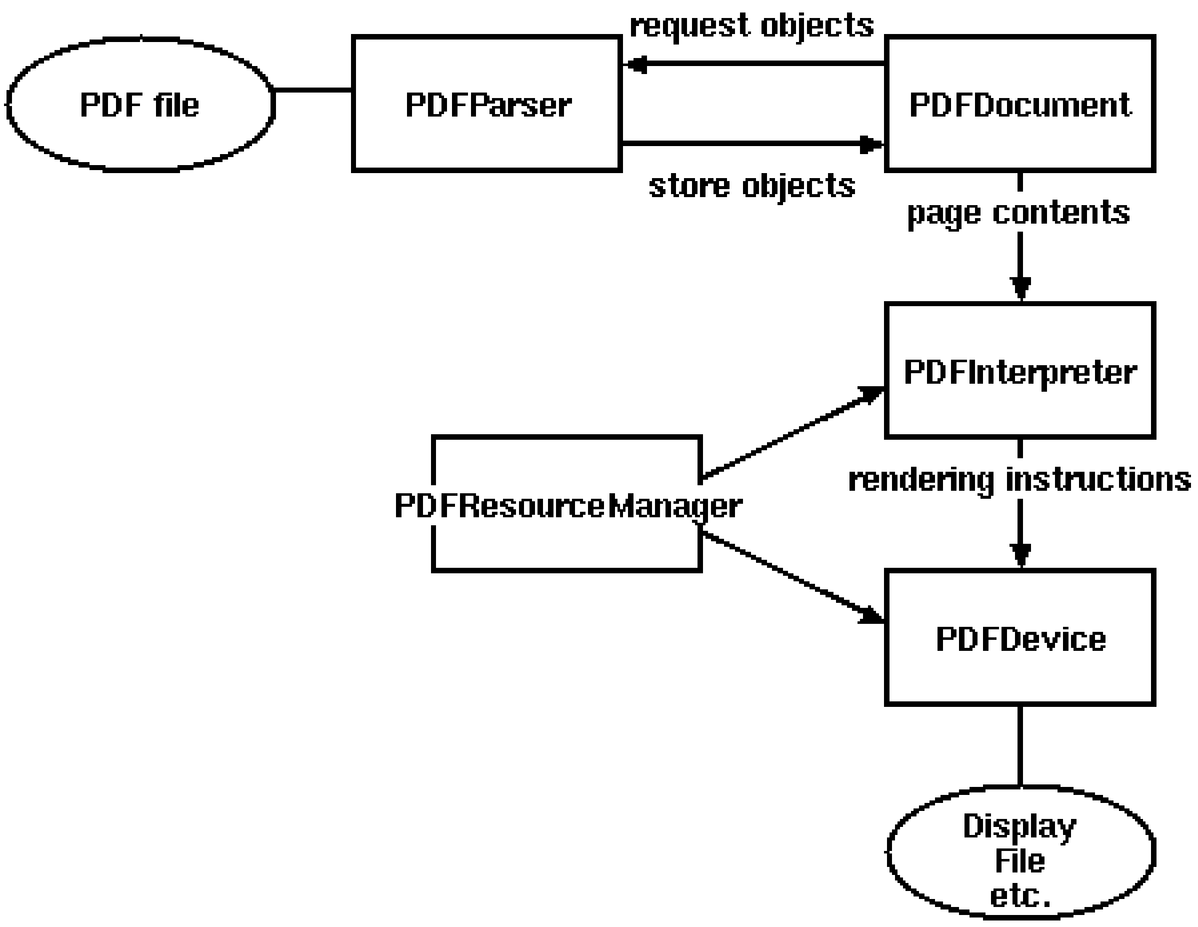


Figure 1. Relationships between PDFMiner classes

2.2 Basic Usage

A typical way to parse a PDF file is the following:

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfpage import PDFTextExtractionNotAllowed
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfdevice import PDFDevice

# Open a PDF file.
fp = open('mypdf.pdf', 'rb')
# Create a PDF parser object associated with the file object.
parser = PDFParser(fp)
# Create a PDF document object that stores the document structure.
# Supply the password for initialization.
document = PDFDocument(parser, password)
# Check if the document allows text extraction. If not, abort.
if not document.is_extractable:
    raise PDFTextExtractionNotAllowed
```

```
# Create a PDF resource manager object that stores shared resources.
rsrcmgr = PDFResourceManager()
# Create a PDF device object.
device = PDFDevice(rsrcmgr)
# Create a PDF interpreter object.
interpreter = PDFPageInterpreter(rsrcmgr, device)
# Process each page contained in the document.
for page in PDFPage.create_pages(document):
    interpreter.process_page(page)
```

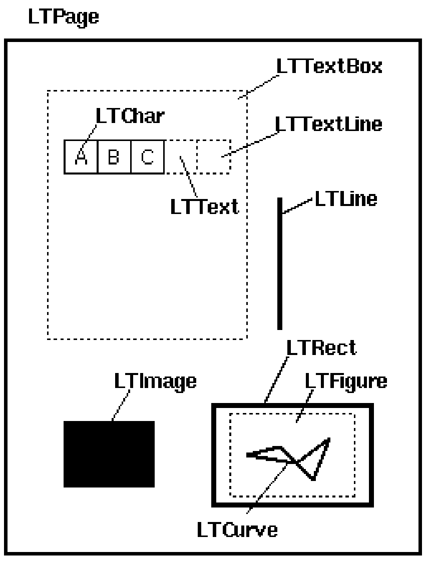
2.3 Performing Layout Analysis

Here is a typical way to use the layout analysis function:

```
from pdfminer.layout import LAParams
from pdfminer.converter import PDFResourceManager, PDFPageAggregator
from pdfminer.pdfpage import PDFPage
from pdfminer.layout import LTTextBoxHorizontal

document = open('myfile.pdf, 'rb')
#Create resource manager
rsrcmgr = PDFResourceManager()
# Set parameters for analysis.
laparams = LAParams()
# Create a PDF page aggregator object.
device = PDFPageAggregator(rsrcmgr, laparams=laparams)
interpreter = PDFPageInterpreter(rsrcmgr, device)
for page in PDFPage.get_pages(document):
    interpreter.process_page(page)
    # receive the LTPage object for the page.
    layout = device.get_result()
    for element in layout:
        if instanceof(element, LTTextBoxHorizontal)
            print(element.get_text())
```

A layout analyzer returns a `LTPage` object for each page in the PDF document. This object contains child objects within the page, forming a tree structure. Figure 2 shows the relationship between these objects.



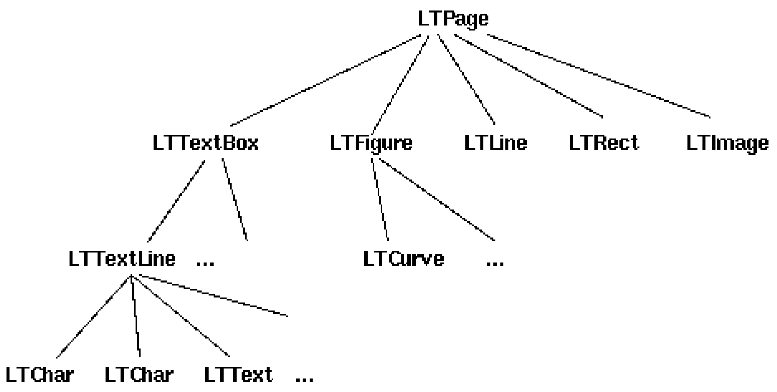


Figure 2. Layout objects and its tree structure

`LTPage`

Represents an entire page. May contain child objects like `LTTextBox`, `LTFigure`, `LTImage`, `LTRect`, `LTCurve` and `LTLine`.

`LTTextBox`

Represents a group of text chunks that can be contained in a rectangular area. Note that this box is created by geometric analysis and does not necessarily represents a logical boundary of the text. It contains a list of `LTTextLine` objects. `get_text()` method returns the text content.

`LTTextLine`

Contains a list of `LTChar` objects that represent a single text line. The characters are aligned either horizontaly or vertically, depending on the text's writing mode. `get_text()` method returns the text content.

`LTChar`

`LTAnno`

Represent an actual letter in the text as a Unicode string. Note that, while a `LTChar` object has actual boundaries, `LTAnno` objects does not, as these are "virtual" characters, inserted by a layout analyzer according to the relationship between two characters (e.g. a space).

`LTFigure`

Represents an area used by PDF Form objects. PDF Forms can be used to present figures or pictures by embedding yet another PDF document within

a page. Note that `LTFigure` objects can appear recursively.

`LTImage`

Represents an image object. Embedded images can be in JPEG or other formats, but currently PDFMiner does not pay much attention to graphical objects.

`LTLine`

Represents a single straight line. Could be used for separating text or figures.

`LTRect`

Represents a rectangle. Could be used for framing another pictures or figures.

`LTCurve`

Represents a generic Bezier curve.

Also, check out a more complete example by Denis Papathanasiou.

2.4 Obtaining Table of Contents

PDFMiner provides functions to access the document's table of contents ("Outlines").

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument

# Open a PDF document.
fp = open('mypdf.pdf', 'rb')
parser = PDFParser(fp)
document = PDFDocument(parser, password)

# Get the outlines of the document.
outlines = document.get_outlines()
for (level,title,dest,a,se) in outlines:
    print (level, title)
```

Some PDF documents use page numbers as destinations, while others use page numbers and the physical location within the page. Since PDF does not have a logical structure, and it does not provide a way to refer to any in-page object from the outside, there's no way to tell exactly which part of text these destinations are referring to.

2.5 Extending Functionality

You can extend `PDFPageInterpreter` and `PDFDevice` class in order to process them differently / obtain other information.

Yusuke Shinyama