PyMuPDF Documentation

version 1.8

Ruikai Liu

Jorj McKie

January 09, 2016

Contents

The PyMuPDF Documentation

Introduction

PyMuPDF (formerly known as **python-fitz**) is a Python binding for MuPDF - "a lightweight PDF and XPS viewer".

MuPDF can access files in PDF, XPS, OpenXPS, CBZ (comic book) and EPUB (e-book) formats.

These are files with extensions `*.pdf`, `*.xps`, `*.oxps`, `*.cbz` or `*.epub` (so in essence, with this binding you can develop **e-book viewers in Python** ...)

PyMuPDF provides access to all important functions of MuPDF from within a Python environment. Nevertheless, we are continuously expanding this function set.

MuPDF stands out among all similar products for its top rendering capability and unsurpassed processing speed.

You can check this out yourself: Compare the various free PDF-viewers. In terms of speed and rendering quality SumatraPDF ranges at the top (apart from MuPDF's own standalone viewer) - and it is based on MuPDF!

While PyMuPDF has been available since several years for an earlier version of MuPDF (1.2), it was until only mid May 2015, that its creator and a few co-workers decided to elevate it to support the current release of MuPDF (first 1,7a and, since November 2015, 1.8).

And we are determined to keep PyMuPDF current with future MuPDF changes!

This work is now completed.

PyMuPDF has been tested on Linux, Windows 7, Windows 10, Python 2 and Python 3 (x86 versions). Other platforms should work too as long as MuPDF supports them.

The main differences compared to version 1.2 are

- A greatly simplified installation procedure: For Windows and Linux platforms it should come down to running the `python setup.py install` command.
- The API has changed: it is now simpler and a lot less cryptic.
- The supported function set has been significantly increased: apart from rendering, MuPDF's traditional strength, we now also offer a wide range of text extraction options.
- Demo code has been extended, and an additional `examples` directory is there to contain working programs. Among them are an editor for a document's table of contents, a full featured document joiner and a document-to-text conversion utility.

We invite you to join our efforts by contributing to the the wiki pages, by using what is there - and, of course, by submitting issues and bugs to the site!

Note on the Name `fitz`

The Python import statement for this library is `import fitz`. Here is the reason why:

The original rendering library for MuPDF was called `Libart`. "After Artifex Software acquired the MuPDF project, the development focus shifted on writing a new modern graphics library called `Fitz`. Fitz was originally intended as an R&D project to replace the aging Ghostscript graphics library, but has instead become the rendering engine powering MuPDF." (Quoted from Wikipedia).

Installation

This describes how to install PyMuPDF.

Step 1: Download PyMuPDF

Download this repository and unzip / decompress it. This will give you a folder, let us call it `PyFitz`.

Step 2: Download and Generate MuPDF 1.8

If your platform is not Windows (or if it is and you absolutley want to generate MuPDF) download `mupdf-1.8-source.tar.gz` from MuPDF version 1.8 source, now and unzip / decompress it. Let us call the resulting folder `mupdf`.

Put it inside `PyFitz` as a subdirectory, if you want to keep everything in one place.

If your platform is **not Windows**, you must **generate MuPDF now**. The MuPDF download includes generation procedures / makefiles for numerous platforms.

On Windows, you have two options:

- if you **have installed MS Visual Studio**, generate the MuPDF `lib` files now. The respective VS project files are contained in `./PyFitz/mupdf/platform/win32`. If that worked fine, the `lib` files are now in `./PyFitz/mupdf/platform/win32/Release`. Update `setup.py` to reference this directory as `library_dirs=['./mupdf/platform/win32/Release']`.
- if you have not installed Visual Studio or if you do not want to generate MuPDF, you must download PyMuPDF Optional Material now and unzip / decompress its content in directory `./PyFitz/PyMuPDF-optional-material`. This optional material contains the `lib` files needed for PyMuPDF generation, and the MuPDF header files. Update `setup.py`, parameter `include_dirs`, to point to these header files.

Step 3: Build / Setup PyMuPDF

If necessary, adjust the setup.py script now. E.g. make sure that

- the include directory is correctly set in sync with your directory structure
- the object code libraries are correctly defined

Now perform a `python setup.py install`

Note on using UPX

In Windows systems, your PyMuPDF installation will end up with three files: `__init__.py`, `fitz.py` and `_fitz.pyd` in the `site-packages` directory. The PYD file is Python's DLL version on Windows systems. `_fitz.pyd` has a size of 9.5 to 10 MB.

You can reduce this by applying the compression utility UPX to it: `upx -9 _fitz.pyd`. This will reduce the file to about 4.5 MB. This should reduce load times (`import fitz` statement) while keeping it fully functional.

Tutorial

This tutorial will show you the use of MuPDF in Python step by step.

Because MuPDF supports not only PDF, but also XPS, OpenXPS and EPUB formats, so does PyMuPDF. Nevertheless we will only talk about PDF files for the sake of brevity.

As for string handling, MuPDF will pass back any string as UTF-8 encoded - no exceptions. Where this binding has added functionality, we usually decode string to unicode. An example is the **`Document.ToC()`** method.

Import the Bindings

The Python bindings to MuPDF are made available by this import statement:

```
import fitz
```

Open a Document

In order to access a supported document, it must be opened with the following statement:

```
doc = fitz.Document(filename)
```

This will create `doc` as a Document object. `filename` must be a Python string or unicode object that specifies the name of an existing file (with or without a fully or partially qualified path).

It is also possible to construct a document from memory data, i.e. without using a file. See Document for details.

A Document contains several attributes and functions. Among them are meta information (like "author" or "subject"), number of total pages, outline and encryption information.

Some Document methods and attributes

Method / Attribute	Description
`Document.pageCount`	Number of pages of `filename` (integer).
`Document.metadata`	Metadata of the `Document` (dictionary).
`Document.outline`	First outline entry of `Document`
`Document.getToC()`	Table of contents of `Document` (list).
`Document.loadPage()`	Create a `Page` object.

Access Meta Data

`Document.metadata` is a Python dictionary with the following keys. For details of their meanings and formats consult the PDF manuals, e.g. Adobe PDF Reference sixth edition 1.7 November 2006. Further information can also be found in chapter Document. The meta data fields are of type string if not otherwise indicated and may be missing, in which case they contain `None`.

Key	Value
`producer`	Producer (producing software)
`format`	PDF format, e.g. 'PDF-1.4'
`encryption`	Encryption method used
`author`	Author
`modDate`	Date of last modification

keywords	Keywords
title	Title
creationDate	Date of creation
creator	Creating application
subject	Subject

Work with Outlines

The easiest way to get all outlines of a document, is creating a table of contents:

```
toc = doc.getToC(simple = True)        # the simple form, if False, link information is inclu
```

This will return a Python list `[[level, title, page, link], ...]` (or `[]`).

`level` is the hierarchy level of the entry (starting from 1), `title` is the entry's title (unicode), and `page` the page number (1-based). `link` is present if `simple = False` is specified. Its meaning can be look up under **`Page.getLinks()`**.

If you want a more detailed control of what you get, enter the document's outline tree like this:

```
olItem = doc.outline    # the document's first outline item
```

This creates olItem as an Outline object. Look there for further details.

Work with Pages

Tasks that can be performed with a Page are at the core of MuPDF's functionality. Among other things, you can render a Page, optionally zooming, rotating or shearing it. You can write it's image to files (in PNG format), extract text from it or search for text strings. At first, a page object must be created:

```
page = doc.loadPage(n)        # represents page n of the document
```

Some typical uses of Page objects follow:

Inspect the links on a Page

Here is how to get all links and their types:

```
#--------------------------------------------------------------------------------
# Get all links of the current page
#--------------------------------------------------------------------------------
links = page.getLinks()
```

`links` is a Python list containing Python dictionaries as entries. For details see **`Page.getLinks()`**.

Render a Page

This example creates an image out of a page's content:

```
pix = page.getPixmap(matrix = fitz.Identity, colorspace = "RGB")
#--------------------------------------------------------------------------------
# now pix contains an RGB image of the page, ready to be used
#--------------------------------------------------------------------------------
```

Save the page image in a file

We can simply store the image in a PNG file:

```
pix.writePNG("test.png")
```

Display the image in dialog managers

We can also use the image in a dialog. **`Pixmap.samples`** represents the area of bytes of all the pixels as a Python bytearray. This area (or its `str()`-version), is directly usable by presumably most dialog managers. Here are two examples. Please also have a look at the examples directory of this repository.

wxPython:

```
data = pix.samples                          # data = bytearray of raw pixel data (RGBA)
bitmap = wx.BitmapFromBufferRGBA(pix.width, # image width
            pix.height,                     # image height
            str(data))                      # wxPython only accepts strings, no bytearrays
```

Tkinter:

```
data = pix.samples
img = Image.frombytes("RGBA", [pix.width, pix.height], str(data))
photo = ImageTk.PhotoImage(img)
```

Text extraction

We can also extract all text of a page in one chunk of string:

```
text = page.getText(output = "text")
```

For the `output` parameter, the following values can be specified:

- `text`: plain text with line breaks. No format and no position info.
- `html`: line breaks, alignment, grouping. No format and no position info
- `json`: full formatting info (except colors and fonts) down to spans (see appendix)
- `xml`: full formatting info (except colors) down to individual characters

To give you an idea about the output of these alternatives, we did text extracts from this document's PDF version and several other examples. See the appendix for details about implications on processing times and space requirements.

Text Searching

You can find out, exactly where on a page a certain string appears like this:

```
areas = page.searchFor("mupdf", hit_max = 32)
```

The variable `areas` will now contain a list of up to 32 Rect rectangles which surround each occurrence of string "mupdf" (not case sensitive).

Please also do have a look at the demonstration program `demo.py`. Among others it contains details on how the TextPage, TextSheet, Device and DisplayList classes can be used for a more direct control, e.g. when performance considerations require it.

Output

Output capabilities of MuPDF (such as PDF generation) are currently very limited. However, a copy of a currently opened document can be created (PDF only, none of the other file types).

We support this with the method **`Document.save()`**.

If the document had been successfully decrypted before, `save()` will automatically create a decrypted copy.

In addition, this method will also perform some clean-up:

If the document containes invalid or broken xrefs, the saved version will have them automatically corrected, which makes it readable by other Python PDF software, like pdfrw or PyPDF2. In many cases, the saved version will also be smaller than the original.

`Document.save()` now also supports all options of MuPDF's command line utility `mutool clean`, see below (mutool clean option = "MCO").

Option	MCO	Effect
garbage = 1	-g	garbage collect unused objects
garbage = 2	-gg	in addition to 1, compact xref tables
garbage = 3	-ggg	in addition to 2, merge duplicate objects
clean = 1	-s	clean content streams (avoid or use with care!)
deflate = 1	-z	deflate uncompressed streams
ascii = 1	-a	convert data to ASCII format
linear = 1	-l	create a linearized version (do not use yet)
expand = 1	-i	decompress images
expand = 2	-f	decompress fonts
expand = 255	-d	decompress all
incremental = 1	n/a	only save data that have changed (do not use yet)

Please note, that `Document.save()`, according to MuPDF's documentation, is still being further developed, so expect changes in the future here. Also observe our remarks in the above table.

Like with `mutool clean`, not all combinations of the above options may work for all documents - so be ready to experiment a little.

We have found, that the fastest, yet very stable combination is `mutool clean -ggg -z`, giving good compression results. In PyMuPDF this corresponds to `doc.save(filename, garbage=3, deflate=1)`.

In some cases, best compression factors result, if `expand` and `deflate` are used together, though they seem to be contradictory. This works, because MuPDF is forced to expand and then re-compress all objects, which will correct poor compressions during document creation.

Close

In some situations it is desirable to "close" a Document such that it becomes fully available again to the OS while your program is still running.

This can be achieved by the `Document.close()` method. Apart from closing the file, all buffer areas associated with the document will be freed. If the document has been created from memory data, no underlying file is opened by MuPDF, so only the buffer release will take place.

Caution:

As with normal file objects, after close, the document and all objects referencing it will be invalid and **must no longer be used**. This binding protects against most such invalid uses by disabling properties and methods of the Document and any associated `Document.loadPage()` objects.

However, re-opening a previously closed file by a new Document is no problem. Please also do have a look at the following valid example:

```
doc = fitz.Document(f_old)           # open a document
<... some statements ...>            # e.g. decryption
doc.save(fnew, garbage=3, deflate=1) # save a decrypted / compressed version
doc.close()                          # close input file
os.remove(f_old)                     # remove it
os.rename(f_new, f_old)              # rename the decrypted / cleaned version
doc = fitz.Document(f_old)           # use it as input for MuPDF
```

Example: Dynamically cleaning up corrupt PDF documents

This shows a potential use of PyMuPDF with another Python PDF library (`pdfrw`).

If a PDF is broken or needs to be decrypted, one could dynamically invoke PyMuPDF to recover from problems like so:

```
import sys
from pdfrw import PdfReader
import fitz
from cStringIO import StringIO

#---------------------------------------
# 'tolerant' PDF reader
#---------------------------------------
def reader(fname):
    ifile = open(fname, "rb")
    idata = ifile.read()                # put in memory
    ifile.close()
    ibuffer = StringIO(idata)           # convert to stream
    try:
        return PdfReader(ibuffer)       # let us try
    except:                             # problem! see if PyMuPDF can heal it
        doc = fitz.Document("application/pdf",
                            idata,
                            len(idata)) # scan pdf data in memory
        doc.save("test.pdf",            # may want to use a temp file
                 garbage=3,
                 deflate=1)             # save a cleaned version
        ifile = open("test.pdf", "rb")  # open it
        idata = ifile.read()            # put in memory
        ifile.close()
        ibuffer = StringIO(idata)       # convert to stream
        return PdfReader(ibuffer)       # now let pdfrw retry
#---------------------------------------

pdf = reader(sys.argv[1])
print pdf.Info
# do further processing
```

With the command line utility `pdftk` a similar result can be achieved, see here. It even supports buffers for input **and** output. However you must invoke it as a separate process via `subprocess.Popen`, using stdin and stdout as communication vehicles.

Classes and Methods

A list of PyMuPDF classes and methods, accessible via `fitz.<name>` if your import statement was `import fitz`.

Class / Method	Short Description
Colorspace	define the color space of a Pixmap.
Device	target object for rendering or text extraction.
DisplayList	a list containing drawing commands.
Document	basic class for dealing with files.
Identity	the do-nothing Matrix
IRect	a rectangle (pixel coordinates).
Link	a destination
linkDest	the destination of an outline entry
Matrix	a 3x3 matrix used for transformations.
Outline	outline element (a.k.a. bookmark).
Page	a document page.
Pixmap	a pixel map (for rendering).
Point	represents a point in the plane.
Rect	a rectangle (float coordinates).
TextPage	text content of a page.
TextSheet	a list of text styles used in a page.

Colorspace

Represents the color space of a Pixmap.

Class API

class **`Colorspace`**

`__init__` **(**`self`**,** `colorspace`**)**
Constructor

Parameters: **colorspace** (*int*) -- A number identifying the colorspace. Supported colorspacevalues are **`CS_RGB`**, **`CS_GRAY`** and **`CS_CMYK`**.

Device

The different format handlers (pdf, xps, etc.) interpret pages to a "device". These devices are the basis for everything that can be done with a page: rendering, text extraction and searching. The device type is determined by the selected construction method.

Class API

class **`Device`**

`__init__` **(**`self`**,** `object`**)**
Constructor for either a pixel map or a display list device.

`object`
An object representing one of Pixmap, or DisplayList

Type: instance

`__init__` **(**`self`**,** `textsheet`**,** `textpage`**)**
Constructor for a text page device.

`textsheet`
A TextSheet object.

Type: instance

`textpage`
A TextPage object.

Type: instance

DisplayList

DisplayList is a list containing drawing commands (text, images, etc.). The intent is two-fold:

1. as a caching-mechanism to reduce parsing of a page
2. as a data structure in multi-threading setups, where one thread parses the page and another one renders pages.

A `DisplayList` is populated with objects from a page by running **`Page.run()`** on a Device. Replay the list (once or many times) by invoking the display list's **`run()`** function.

Method	Short Description
`run()`	(Re)-run a display list through a device.

Class API

class **`DisplayList`**

`fitz.`**`DisplayList`** **(**`self`**)**
Create a rendering device for a display list.
When the device is rendering a page it will populate the display list with drawing commands (text, images, etc.). The display list can later be reused to render a page many times without having to re-interpret the page from the document file.

Return type: Device

`run` **(**`self`**,** `dev`**,** `ctm`**,** `area`**)**

Parameters:

- **dev** (Device) -- Device obtained from Device
- **ctm** (Matrix) -- Transform matrix to apply to display list contents.
- **area** (Rect) -- Only the part of the contents of the display list visible within this area will be considered when the list is run through the device. This does not imply for tile objects contained in the display list.

Document

This class represents a document. It can be constructed from a file or from memory. See below for details.

Method / Attribute	Short Description
`Document.authenticate()`	Decrypts the document
`Document.loadPage()`	Reads a page
`Document.save()`	Saves a copy of the document
`Document.getToC()`	Create a table of contents
`Document.close()`	Closes the document
`Document.isClosed`	Has document been closed?
`Document.outline`	First *Outline* item
`Document.name`	filename of document
`Document.needsPass`	Require password to access data?
`Document.isEncrypted`	Is document still encrypted?
`Document.pageCount`	The document's number of pages
`Document.metadata`	The document's meta data

Class API

class **`Document`**

`__init__` **(**`self`**,** `filename`**)**
Constructs a `Document` object from a file.

Parameters: **filename** (*string*) -- A string (UTF-8 or unicode) containing the path / name of the document file to be used. The file will be opened and remain open until either explicitely closed (see below) or until end of program.

Return type: Document

Returns: A `Document` object.

`__init__` **(**`self`**,** `filetype`**,** `stream=data`**,** `streamlen=len(data)`**)**
Constructs a `Document` object from memory data.

Parameters:

- **filetype** (*string*) -- A string specifying the type of document contained in `stream`. This may be either something that looks like a filename (e.g. `x.pdf`), in which case MuPDF uses the extension to determine the type, or a mime type like `application/pdf`. Recommended is using the filename scheme, or even the name of the original file for documentation purposes.
- **stream** (*string*) -- A string of data representing the content of a supported document type.
- **streamlen** (*int*) -- An integer specifying the length of the stream.

Return type: Document

Returns: A `Document` object.

`authenticate` **(**`password`**)**
Decrypts the document with the string `password`. If successfull, all of the document's data can be accessed (e.g. for rendering).

Parameters: **password** (*string*) -- The password to be used.

Return type: int

Returns: `True (1)` if decryption with `password` was successfull, `False (0)` otherwise.

loadPage (number)
Loads a Page for further processing like rendering, text searching, etc. See the Page object.

Parameters: **number** (*int*) -- page number, zero-based (0 is the first page of the document).

Return type: Page

getToC (simple = True)
Creates a table of contents out of the document's outline chain.

Parameters: **simple** (*boolean*) -- Indicates whether a detailed ToC is required. If `simple = False` is specified, each entry of the list also contains a dictionary with Link details for each outline entry.

Return type: list

save (outfile, garbage=0, clean=0, deflate=0, incremental=0, ascii=0, expand=0, linear=0)
Saves a copy of the document under the name `outfile`. Include path specifications as necessary. Only PDF documents are supported by this function. Internally the document may have changed. E.g. after a successfull `authenticate`, a decrypted copy will be saved, and, in addition (even without any of the optional parameters), some basic cleaning of the document data could also have occurred, e.g. broken xref tables have been corrected and incremental changes have been resolved.

Parameters:

- **outfile** (*string*) -- The file name to save to. Must be different from the original `filename` / `filetype` value or else a ValueError will be raised.
- **garbage** (*int*) -- Do garbage collection: 0 = none, 1 = remove unused objects, 2 = in addition compact xref tables, 3 = in addition merge duplicate objects.
- **clean** (*int*) -- Clean content streams: 0 = False, 1 = True.
- **deflate** (*int*) -- Deflate uncompressed streams: 0 = False, 1 = True.
- **incremental** (*int*) -- Only save changed objects: 0 = False, 1 = True.
- **ascii** (*int*) -- Where possible make the output ASCII: 0 = False, 1 = True.
- **expand** (*int*) -- One byte bitfield to decompress contents: 0 = none, 1 = images, 2 = fonts, 255 = all. This convenience option generates a decompressed file version that can be better read by some other programs.
- **linear** (*int*) -- Save a linearised version of the document: 0 = False, 1 = True. This option creates a file format for improved performance when read via internet connections.

Return type: int

Returns: Count of errors that occurred during save. Note: PyMuPDF will recover from many errors encountered in a PDF and continue processing.

close ()
Releases space allocations associated with the document, and, if created from a file, closes `filename` thus releasing control of it to the OS.

outline
Contains either `None` or the first Outline entry of the document. Can be used as a starting point to walk through all outline items. If an encrypted document has not yet been authenticated, an `AttributeError` exception will be raised, when this attribute is being accessed.

Return type: Outline

isClosed
`False (0)` if document is still open, `True (1)` otherwise. If closed, most other attributes and all methods will have been deleted / disabled. In addition, Page objects referring to this document (i.e. created with **`Document.loadPage()`**) will no longer be usable. For reference purposes, **`Document.name`** still exists and will contain the filename of the original document.

Return type: int

needsPass
Contains an indicator showing whether the document is encrypted (`True = 1`) or not (`False = 0`). This indicator remains unchanged - even after the document has been authenticated.

Return type: bool

isEncrypted
This indicator initially equals the value of `needsPass`. After a successfull authentification, it is set to `False = 0` to reflect the situation.

Return type: bool

metadata
Contains the document's meta data as a Python dictionary or `None` if the document is encrypted. Its keys are `format`, `encryption`, `title`, `author`, `subject`, `keywords`, `creator`, `producer`, `creationDate`, `modDate`. All item values are strings or `None`.
Except `format` and `encryption`, the key names correspond in an obvious way to a PDF's "official" meta data fields `/Creator`, `/Producer`, `/CreationDate`, `/ModDate`, `/Title`, `/Author`, `/Subject`, `/Keywords` respectively.
The value of `format` contains the version of the PDF format (e.g. 'PDF-1.6').
The value of `encryption` either contains `None` (not encrypted), or a string naming the used encryption method (e.g. `'Standard V4 R4 128-bit RC4'`).
If the date fields contain meaningfull data (which need not be the case), they are strings in the PDF-internal timestamp format "D:<TS><TZ>", where
<TS> is the 12 character ISO timestamp `YYYMMDDhhmmss` (`YYYY` - year, `MM` - month, `DD` - day, `hh` - hour, `mm` - minute, `ss` - second), and
<TZ> is a time zone value (time intervall relative to GMT) containing a sign ('+' or '-'), the hour (hh), and the minute (`'mm'`, attention: enclose in apostrophies!).
For example, a Venezuelan value might look like `D:20150415131602-04'30'`, which corresponds to the timestamp April 15, 2015, at 1:16:02 pm local time Venezuela.

Return type: dict

name
Contains the `filename` or `filetype` value with which `Document` was created.

Return type: string

pageCount
Contains the number of pages of the document. May return 0 for documents with no pages.

Return type: int

Identity

Identity is just a Matrix that performs no action, to be used whenever the syntax requires a Matrix, but no actual transformation should take place.

Caution: `Identity` is a constant in the C code and therefore **readonly, do not try to modify** its properties in any way, i.e. you must not manipulate its `[a,b,c,d,e,f]`, neither apply any method.

`Matrix(1, 1)` creates a matrix that acts like `Identity`, but it may be changed. Use this when you need a starting point for further modification, e.g. by one of the Matrix methods.

In other words:

```
# the following will not work - the interpreter will crash!
m = fitz.Identity.preRotate(90)

# do this instead:
m = fitz.Matrix(1, 1).preRotate(90)
```

IRect

IRect is a rectangular bounding box similar to Rect, except that all corner coordinates are integers. IRect is used to specify an area of pixels, e.g. to receive image data during rendering.

Attribute	Short Description
`IRect.width`	Width of the bounding box
`IRect.height`	Height of the bounding box
`IRect.x0`	X-coordinate of the top left corner
`IRect.y0`	Y-coordinate of the top left corner
`IRect.x1`	X-coordinate of the bottom right corner
`IRect.y1`	Y-coordinate of the bottom right corner

Class API

class **`IRect`**

`__init__` **(**`self`, `x0`, `y0`, `x1`, `y1`**)**
Constructor. Without parameters defaulting to `IRect(0, 0, 0, 0)`, an empty rectangle. Also see the example below. Function **`Rect.round()`** creates the smallest `IRect` containing `Rect`.

`width`
Contains the width of the bounding box. Equals `x1 - x0`.

Type: int

`height`
Contains the height of the bounding box. Equals `y1 - y0`.

Type: int

`x0`
X-coordinate of the top left corner.

Type: int

`y0`
Y-coordinate of the top left corner.

Type: int

`x1`
X-coordinate of the bottom right corner.

Type: int

`y1`
Y-coordinate of the bottom right corner.

Type: int

Example:

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import fitz
>>> irect = fitz.IRect(10, 10, 410, 610)
>>> irect
fitz.IRect(10, 10, 410, 610)
>>> irect.height
600
```

```
>>> irect.width
400
>>>
```

Link

Represents a pointer to somewhere (this document, other documents, the internet). Links exist per document page, and they are forward-chained to each other, starting from an initial link which is accessible by the **`Page.loadLinks()`** method.

Attribute	Short Description
`Link.rect`	Clickable area in untransformed coordinates.
`Link.dest`	Kind of link destination.
`Link.next`	Link to next link

Class API

class **`Link`**

`rect`
The area that can be clicked in untransformed coordinates.

Return type: Rect

`dest`
The link destination kind. An integer to be interpreted as one of the `FZ_LINK_*` values.

Return type: int

`next`
The next `Link` or `None`

Return type: Link

linkDest

Class representing the *dest* property of an outline entry.

Attribute	Short Description
`linkDest.dest`	Destination
`linkDest.fileSpec`	File specification (path, filename)
`linkDest.flags`	Descriptive flags
`linkDest.isMap`	Is this a MAP?
`linkDest.isUri`	Is this an URI?
`linkDest.kind`	Kind of destination
`linkDest.lt`	Top left coordinates
`linkDest.named`	Name if named destination
`linkDest.newWindow`	Name of new window
`linkDest.page`	Page number
`linkDest.rb`	Bottom right coordinates
`linkDest.uri`	URI

Class API

class **`linkDest`**

`dest`
Destination of `linkDest`.

Return type: Link

`fileSpec`
Contains the filename (including any path specifications) this link points to, if applicable.

Return type: string

`flags`
A one-byte bitfield consisting of indicators describing the validity and meaning of the different aspects of the destination. As far as possible, link destinations are constructed such that e.g. **`linkDest.lt`** and **`linkDest.rb`** can be treated as defining a bounding box, though the validity flags (see `LINK_FLAG_*` values) indicate which of the values were actually specified. Note that the numerical values for each of the LINK_FLAGs are powers of 2 and thus indicate the position of the bit to be tested. More than one bit can be `True`, so do not test for the value of the integer.

Return type: int

`isMap`
This flag specifies whether to track the mouse position when the URI is resolved. Default value: False.

Return type: bool

`isUri`
Specifies whether this destination is an internet resource.

Return type: bool

`kind`
Indicates the type of this destination, like a place in this document, a URI, a file launch, an action or a place in another file. Look at index entries `FZ_LINK_*` to see the names and numerical values.

Return type: int

`lt`
The top left Point of the destination.

Return type: Point

`named`
This destination refers to some named resource of the document (see Adobe PDF documentation).

Return type: int

`newWindow`
This destination refers to an action that will open a new window.

Return type: bool

`page`
The page number (in this or the target document) this destination points to.

Return type: int

`rb`
The bottom right Point of this destination.

Return type: Point

`uri`
The name of the URI this destination points to.

Return type: string

Matrix

Matrix is a row-major 3x3 matrix used by image transformations in MuPDF. With matrices you can manipulate the rendered image of a page in a variety of ways: (parts of) the page can be rotated, zoomed, flipped, sheared and shifted by setting some or all of just six numerical values.

Since all points or pixels live in a two-dimensional space, one column vector of that matrix is a constant unit vector, and only the remaining six elements are used for manipulations. These six elements are usually represented by `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

It should be noted, that

- the below methods are just convenience functions - everything they do, can also be achieved by directly manipulating `[a,b,c,d,e,f]`
- all manipulations can be combined - you can construct a matrix that does a rotate **and** a shear **and** a scale **and** a shift etc. in one go

Method / Attribute	Description
`Matrix.__init__()`	Constructor.
`Matrix.preRotate()`	Perform a rotation
`Matrix.preScale()`	Perform a scaling
`Matrix.preShear()`	Perform a shearing
`Matrix.a`	Zoom factor X direction
`Matrix.b`	Shearing effect Y direction
`Matrix.c`	Shearing effect X direction
`Matrix.d`	Zoom factor Y direction
`Matrix.e`	Horizontal shift
`Matrix.f`	Vertical shift

Class API

class **`Matrix`**

`__init__` **(**`self`, `a=1`, `b=0`, `c=0`, `d=1`, `e=0`, `f=0`**)**
Constructor. `Matrix(1, 1)` will construct a modifyable version of the Identity matrix.

`preRotate` **(**`deg`**)**
Performs a clockwise rotation for positive `deg` degrees, else counterclockwise. This will change the matrix elements in the following way: `a = cos(deg)`, `b = sin(deg)`, `c = -sin(deg)`, `d = cos(deg)`. `e` and `f` will remain unchanged.

Parameters: **deg** (*float*) -- The rotation angle in degrees (use conventional notation based on Pi = 180 degrees).

Return type: Matrix

`preScale` **(**`sx`, `sy`**)**
Scales by the zoom factors sx and sy. Has effects on attributes `a` and `d` only.

Parameters:
- **sx** (*float*) -- Zoom factor in X direction. For the effect see description of attribute a.
- **sy** (*float*) -- Zoom factor in Y direction. For the effect see description of attribute d.

Return type: Matrix

preShear (sx**,** sy**)**

Performs shearing, i.e. transformation of rectangles into parallelograms (rhomboids). Has effects on attributes b and c only.

Parameters:
- **sx** (*float*) -- Shearing effect in X direction. See attribute c.
- **sy** (*float*) -- Shearing effect in Y direction. See attribute b.

Return type: Matrix

a

Scaling in X-direction **(width)**. For example, a value of 0.5 performs a shrink of the **width** by a factor of 2. If a < 0, a (additional) vertical flip will occur, i.e. the rectangle's picture will be mirrored along the Y axis.

Type: float

b

Causes a shearing effect: each `Point(x, y)` will become `Point(x, y - b*x)`. Therefore, looking from left to right, e.g. horizontal lines will be "tilt" - downwards if b > 0, upwards otherwise (b is the tangens of the tilting angle).

Type: float

c

Causes a shearing effect: each `Point(x, y)` will become `Point(x - c*y, y)`. Therefore, looking upwards, vertical lines will be "tilt" - to the left if c > 0, to the right otherwise (c ist the tangens of the tilting angle).

Type: float

d

Scaling in Y-direction **(height)**. For example, a value of 1.5 performs a stretch of the **height** by 50%. If d < 0, a (additional) horizontal flip will occur, i.e. the rectangle's picture will be mirrored along the X axis.

Type: float

e

Causes a horizontal shift effect: Each `Point(x, y)` will be shifted right to become `Point(x + e, y)`. Note that negative values of `e` will shift left.

Type: float

f

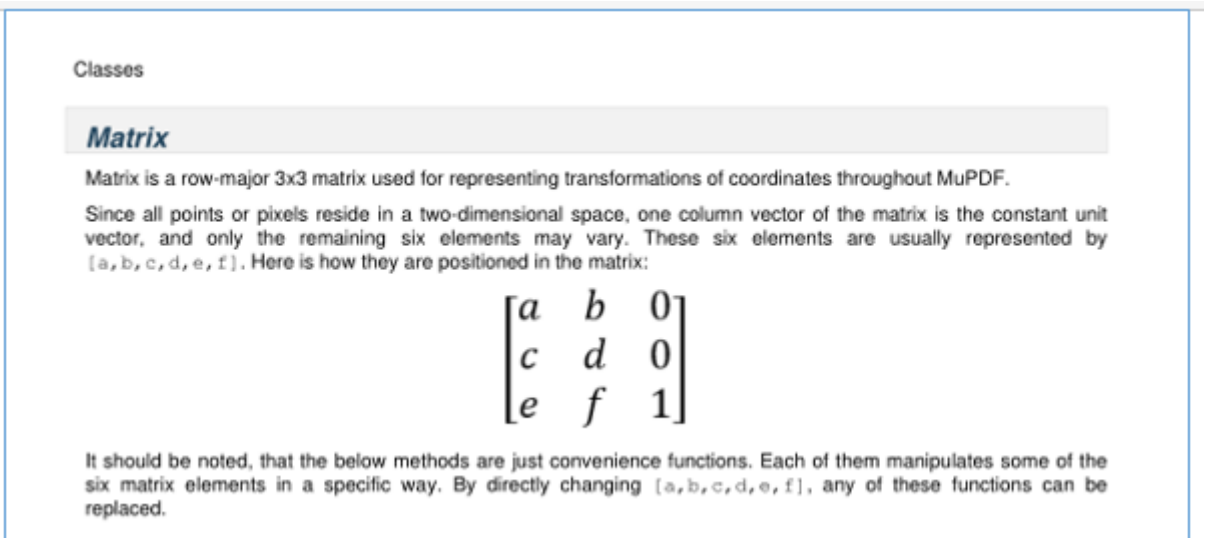
Causes a vertical shift effect: Each `Point(x, y)` will be shifted down to become `Point(x, y - f)`. Note that negative values of `f` will shift up.

Type: float

Examples

Here are examples to illustrate some of the effects achievable with matrices. The following pictures start with a page of the PDF version of this help file. We show what will happen when a matrix is being applied (though always full pages are created, only parts are displayed here to save space).

This is the original page image



Classes

Matrix

Matrix is a row-major 3x3 matrix used for representing transformations of coordinates throughout MuPDF.

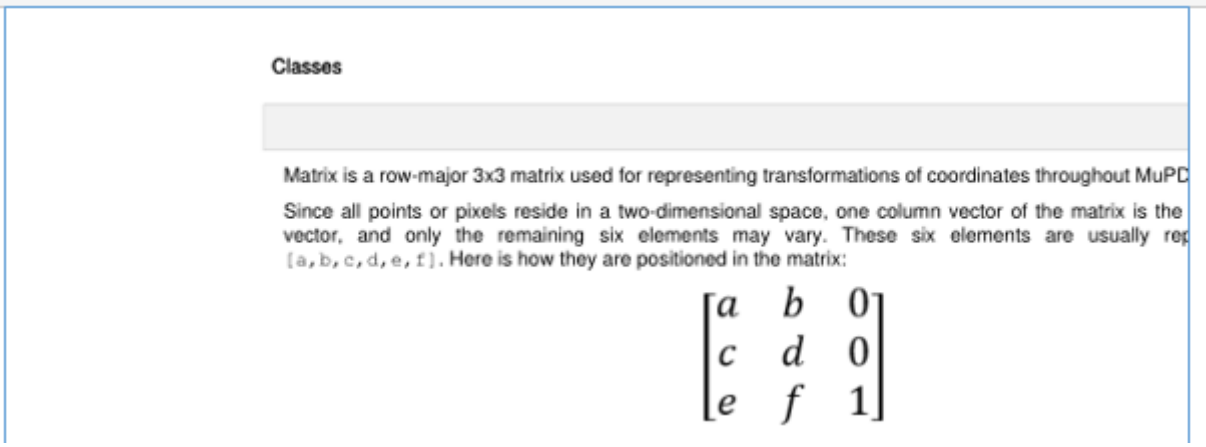
Since all points or pixels reside in a two-dimensional space, one column vector of the matrix is the constant unit vector, and only the remaining six elements may vary. These six elements are usually represented by `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

It should be noted, that the below methods are just convenience functions. Each of them manipulates some of the six matrix elements in a specific way. By directly changing `[a,b,c,d,e,f]`, any of these functions can be replaced.

Shifting

We transform it with a matrix where `e = 100` (right shift by 100 pixels)



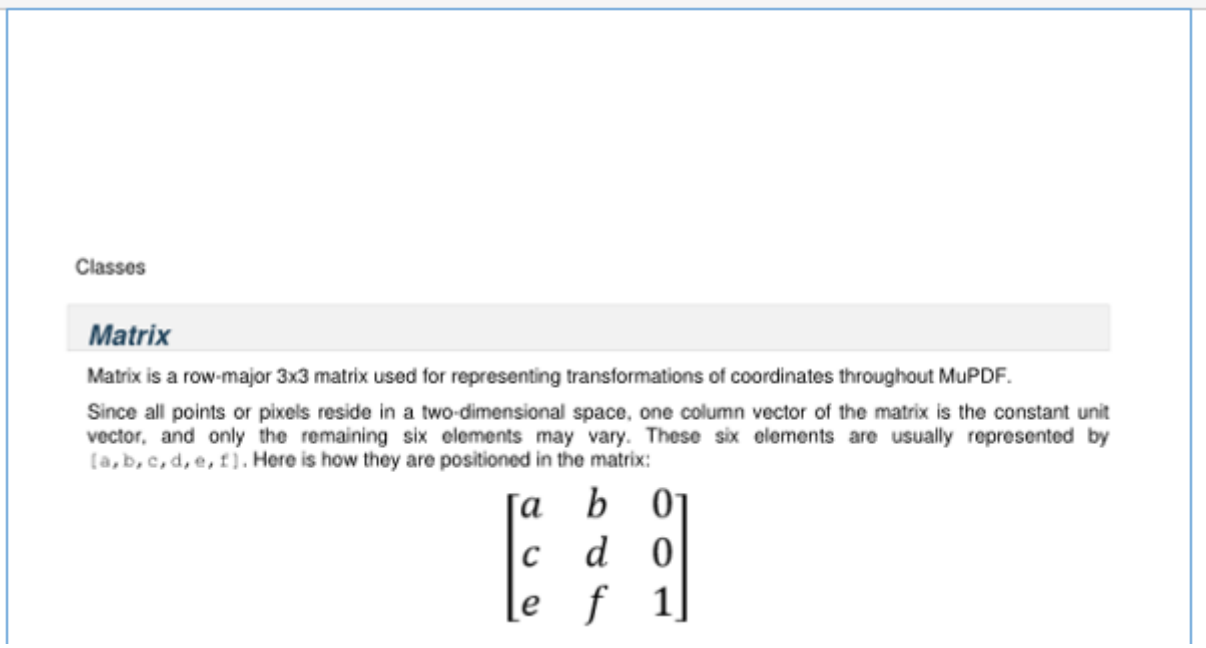
Classes

Matrix is a row-major 3x3 matrix used for representing transformations of coordinates throughout MuPD

Since all points or pixels reside in a two-dimensional space, one column vector of the matrix is the vector, and only the remaining six elements may vary. These six elements are usually rep `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Next we do a down shift by 100 pixels: `f = 100`



Classes

Matrix

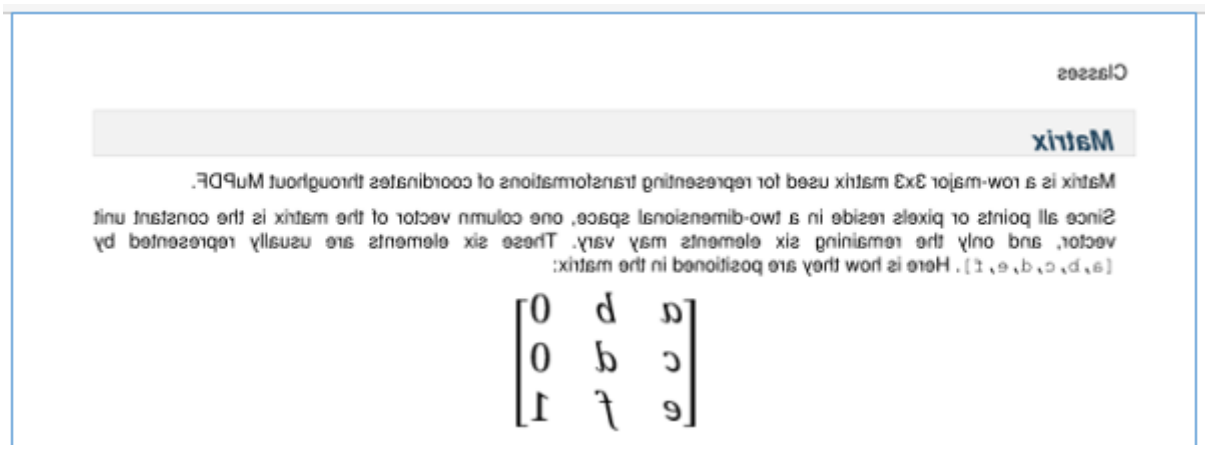
Matrix is a row-major 3x3 matrix used for representing transformations of coordinates throughout MuPDF.

Since all points or pixels reside in a two-dimensional space, one column vector of the matrix is the constant unit vector, and only the remaining six elements may vary. These six elements are usually represented by `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

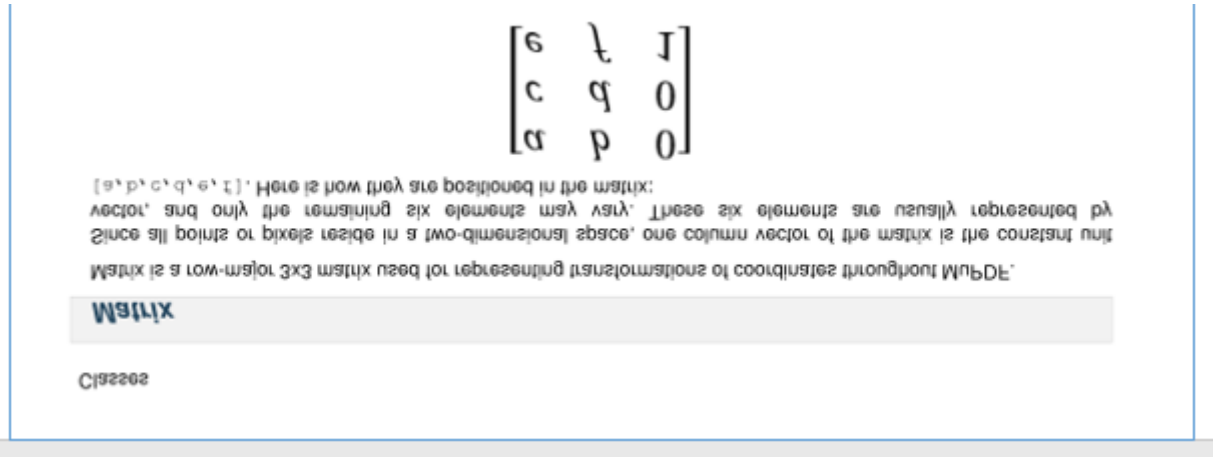
$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Flipping

Flip the page vertically (`a = -1`)

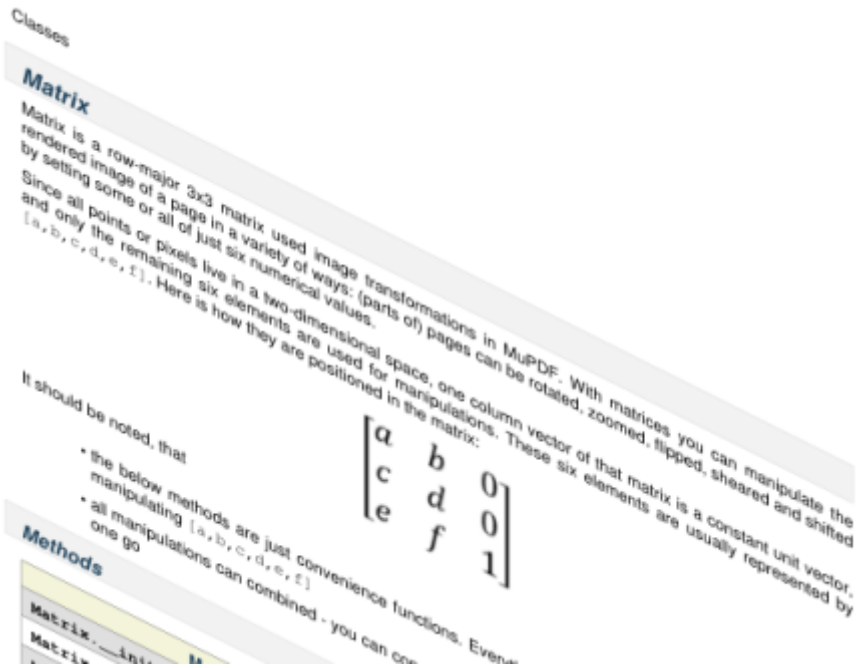


Flip horizontally (`d = -1`)



Shearing

First a shear in Y direction (`b = 0.5`)



Second a shear in X direction (`c = 0.5`)

Classes

Matrix

Matrix is a row-major 3x3 matrix used image transformations in MuPDF. With matrices you can manipulate the rendered image of a page in a variety of ways: (parts of) pages can be rotated, zoomed, flipped, sheared and shifted by setting some or all of just six numerical values.

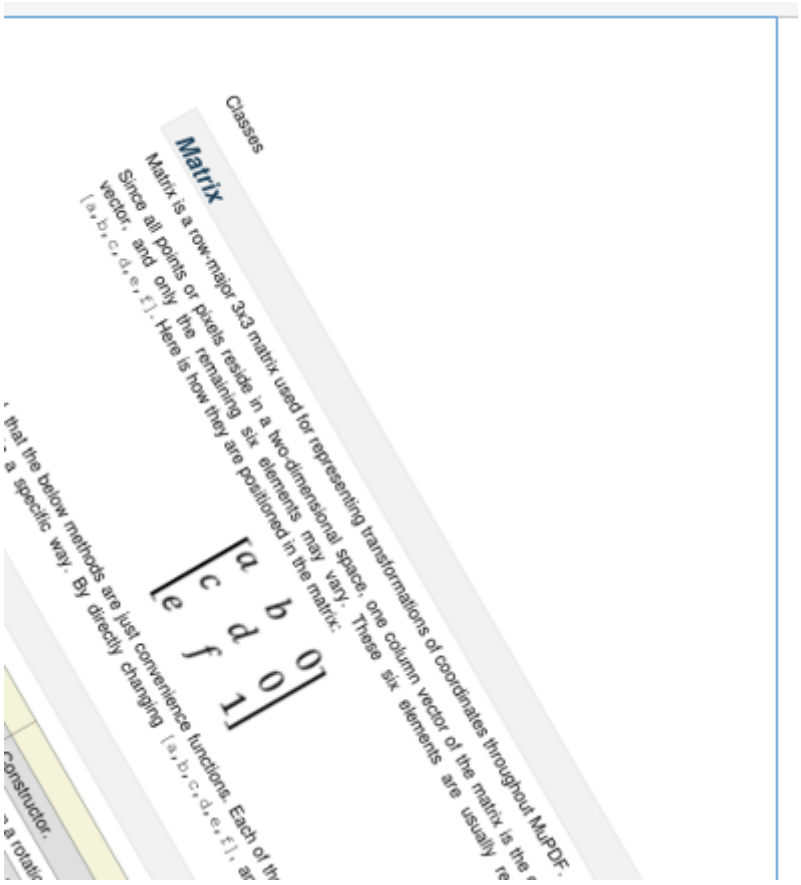
Since all points or pixels live in a two-dimensional space, one column vector of that matrix is a constant unit vector, and only the remaining six elements are used for manipulations. These six elements are usually represented by `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

It should be noted, that

Rotating

Finally a rotation by 60 degrees



Outline

`outline` is a property of `Document`. If not `None`, it stands for the first outline item of the document. Its properties in turn define the characteristics of this item and also point to other outline items in "horizontal" direction by property `Outline.next` to the next item of same level, and "downwards" by property `Outline.down` to the next item one level lower. The full tree of all outline items for e.g. a conventional table of contents can be recovered by following these "pointers".

Method / Attribute	Short Description
`Outline.down`	Next item downwards
`Outline.next`	Next item same level
`Outline.dest`	Link destination
`Outline.title`	Title
`Outline.saveText()`	Prints a conventional table of contents to a file
`Outline.saveXML()`	Prints an XML-like table of contents to a file

Class API

class **`Outline`**

`down`
The next outline item on the next level down. Is `None` if the item has no children.

Return type: Outline

`next`
The next outline item at the same level as this item. Is `None` if the item is the last one in its level.

Return type: Outline

`dest`
The destination this entry points to. Can be a place in this or another document, or an internet resource. It can include actions to perform like opening a new window, invoking a javascript or opening another document.

Return type: linkDest

`title`
The item's title as a string or `None`.

Return type: string

`saveText ()`
The chain of outline items is being processed and printed to the file `filename` as a conventional table of contents. Each line of this file has the format `<tab>...<tab><title><tab><page#>`, where the number of leading tabs is (n-1), with n equal to the outline level of the entry. Page numbers are 1-based in this case, while `page# = 0` if and only if the outline entry points to a place outside this document. If no title was specified for this outline entry, it appears as a tab character in this file.

Parameters: **filename** (*string*) -- Name of the file to write to.

`saveXML ()`
The chain of outline items is being processed and printed to a file `filename` as an XML-like table of contents. Each line of this file has the format `<outline title="..." page="n"/>`, if the entry has no children. Otherwise the format is `<outline title="..." page="n">`, and child entries will follow. The parent entry will be finished by a line containing `</outline>`.

Parameters: **filename** (*string*) -- Name of the file to write to.

Page

Page interface, created by `Document.loadPage()`.

Method / Attribute	Short Description
`Page.bound()`	the page's rectangle
`Page.loadLinks()`	get the first link of a page
`Page.getLinks()`	get all links of a page
`Page.getText()`	extract the text of a page
`Page.getPixmap()`	create a `Pixmap` for the page
`Page.searchFor()`	search for a string on a page
`Page.run()`	run a page through a device
`Page.number`	page number

Class API

class **Page**

bound ()
Determine the a page's rectangle (before transformation).

Return type: Rect

loadLinks ()
Get the first link of a page.

Return type: Link

Returns: A Link or `None` if the page has no links.

getLinks ()
Retrieves **all** links of a page.

Return type: list

Returns: A list of dictionaries or `[]`. Please note that the links are in the order as specified during PDF generation, not necessarily in any "natural" reading order.

getText (`output = 'text'`**)**
Retrieves the text of a page. Depending on the output parameter, the results of the TextPage extract methods are returned. The required objects TextSheet, DisplayList, Device and TextPage are automatically created or reused if present.
If `output = 'text'` is specified, the text is returned in the order as specified during PDF creation (which is not necessarily the normal reading order). As this may not always look like expected, consider using the example program `PDF2TextJS.py`. It is based on `extractJSON()` output and re-arranges text according to the Western reading layout convention "from top-left to bottom-right".
In an earlier version, we provided a different taste of `TextPage.extractText()` (parameter `basic=False`), which did a similar job. However, the example program mentioned above is more than 50 times faster, so we deleted this option.

Parameters: **output** (*string*) -- A string indicating the requested text format, one of `text` (default), `html`, `json`, or `xml`.

Return type: string

Returns: The page's text as one string.

getPixmap (`matrix = fitz.Identity, colorspace = "RGB"`**)**
Creates a Pixmap from a given page. The method will automatically create or reuse a DisplayList for the page and also create the required drawing Device.

Parameters:
- **matrix** (Matrix) -- A Matrix object. Default is the Identity matrix.
- **colorspace** (*string*) -- Defines the required colorspace, one of `GRAY`, `CMYK` or `RGB` (default).

Return type: Pixmap

Returns: Pixmap of the page.

`searchFor (``text`**`, hit_max = 16)`**

Searches for `text` on a page. Identical to **`TextPage.search()`**.

Return type: list

Returns: A list of Rect rectangles each of which surrounds one occurrence of `text`.

`run (``dev`**`, transform)`**

Run a page through a device.

Parameters:
- **dev** (Device) -- Device, obtained from one of the Device constructors.
- **transform** (Matrix) -- Transformation to apply to the page. May include for example scaling and rotation, see **`Matrix.preScale()`** and **`Matrix.preRotate()`**. Set it to Identity if no transformation is desired.

`number`

The page number

Return type: int

Pixmap

Pixmaps represent a rectangular set of pixels in a 2 dimensional region. Each pixel is described by a number of bytes ("components") plus an alpha (transparency) byte. The data is in premultiplied alpha when rendering, but non-premultiplied for colorspace conversions and rescaling.

There are two ways to construct pixmaps: either an empty one based on Colorspace and IRect information, or one containing an image from some source (e.g. a file), based on the image data area and its length.

Please have a look at the **example** to see some pixmap usage "at work".

Method / Attribute	Short Description
`Pixmap.clearWith()`	clears a pixmap
`Pixmap.clearIRectWith()`	clears parts of a pixmap
`Pixmap.writePNG()`	saves a pixmap as a PNG file
`Pixmap.copyPixmap()`	copy parts of another pixmap
`Pixmap.getSize()`	returns the pixmap's total length
`Pixmap.getColorspace()`	returns the Colorspace used
`Pixmap.getIRect()`	returns the IRect used
`Pixmap.invertIRect()`	invert the pixels of a given bounding box
`Pixmap.samples`	the components data for all pixels
`Pixmap.height`	height of the region in pixels
`Pixmap.width`	width of the region in pixels
`Pixmap.x`	X-coordinate of top-left corner of pixmap
`Pixmap.y`	Y-coordinate of top-left corner of pixmap
`Pixmap.n`	number of bytes per pixel
`Pixmap.xres`	resolution in X-direction
`Pixmap.yres`	resolution in Y-direction
`Pixmap.interpolate`	interpolation method indicator

Class API

class **`Pixmap`**

`__init__` **(`self`, `colorspace`, `irect`)**
This constructor creates an empty pixmap of a size and an origin specified by the irect object. So for a `fitz.IRect(x0, y0, x1, y1)`, `fitz.Point(x0, y0)` will designate the top left corner of the pixmap.

Parameters:
- **colorspace** (Colorspace) -- The required colorspace of the pixmap.
- **irect** (IRect) -- Specifies the pixmap's area and its location.

`__init__` **(`self`, `data`, `len`)**
This constructor creates a (non-empty) pixmap from `data`, which is assumed to contain a PNG image and which is `len` bytes long.

Parameters:
- **data** (*string*) -- Data containing a complete, valid image in PNG format. E.g. this may have been obtained from a statement like `data = open('somepic.png', 'rb').read()`.
- **len** (*int*) -- An integer specifying the length of `data`.

`clearWith` **(`value`)**
Clears a pixmap.

Parameters: **value** (*int*) -- Values from 0 to 255 are valid. Each color byte of each pixel will be set to this value, while alpha will always be set to 255 (non-transparent). Default is 0 (black).

clearIRectWith (`value`**,** `irect`**)**
Clears an area within a pixmap.

Parameters:
- **value** (*int*) -- Values from 0 to 255 are valid. Each color byte of each pixel of the area will be set to this value, while alpha will always be set to 255 (non-transparent). Default is 0 (black).
- **irect** (IRect) -- An IRect object specifying the area to be cleared.

copyPixmap (`source`**,** `irect`**)**
Copies the IRect part of the `source` pixmap into the corresponding area of this one. The two pixmaps may have different dimensions and different colorspaces. The copy mechanism automatically adjusts to any discrepancies between source and target pixmap.

Parameters:
- **source** (*A pixmap*) -- The pixmap from where to copy.
- **irect** (IRect) -- An IRect object specifying the area to be copied.

getSize ()
Returns the total length of the pixmap. This will generally equal `len(pix.samples) + 52`. The following will evaluate to `True`: `len(pixmap) == pixmap.getSize()`.

Return type: int

getColorspace ()
Returns the colorspace used for constructing the pixmap. Usefull when the pixmap has been created from an image area.

Return type: Colorspace

getIRect ()
Returns the IRect used for constructing the pixmap. Usefull when the pixmap has been created from an image area.

Return type: IRect

samples
The color and transparency values for all pixels. `samples` is a memory area of size `width * height * n` bytes. Each n bytes define one pixel. Each successive n bytes yield another pixel in scanline order. Subsequent scanlines follow each other with no padding. E.g. for an RGBA colorspace (i.e. `n = 4`) this means, `samples` is a bytearray like `..., R, G, B, A, ...`, and the four byte values R, G, B, A describe one pixel.

Return type: bytearray

width
The width of the region in pixels.

Return type: int

height
The height of the region in pixels.

Return type: int

x
X-coordinate of top-left corner

Return type: int

y

Y-coordinate of top-left corner

Return type: int

n
Number of components per pixel. This number depends on and identifies the chosen colorspace: **CS_GRAY** = 2, **CS_RGB** = 4, **CS_CMYK** = 5.

Return type: int

xres
Horizontal resolution in dpi (dots per inch).

Return type: int

yres
Vertical resolution in dpi.

Return type: int

invertIRect (irect)
Invert the color of all pixels in IRect. All components except alpha are inverted.

Parameters: **irect** -- Invert all the pixels in the irect. If omitted, the whole pixmap will be inverted.

writePNG (filename, savealpha=False)
Save a pixmap as a png file.

Parameters:
- **filename** (*string*) -- The filename to save as (the extension `png` must be specified).
- **savealpha** (*bool*) -- Save alpha or not.

interpolate
An information-only boolean flag set to `True` if the image will be drawn using linear interpolation, or set to `False` if image is created using nearest neighbour sampling.

Return type: bool

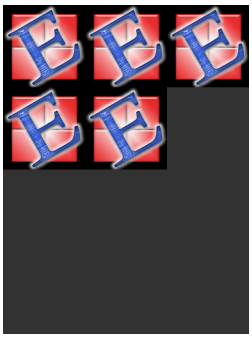
Example

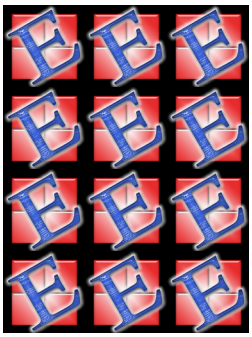
This shows how pixmaps can be used for purely graphical, non-PDF purposes:

```
# read in picture image and create a pixmap of it
pic = open("editra.png", "rb").read()
pix0 = fitz.Pixmap(pic, len(pic))

# calculate target pixmap dimensions and create it
tar_width  = pix0.width * 3
tar_height = pix0.height * 4
tar_irect  = fitz.IRect(0, 0, tar_width, tar_height)
tar_pix    = fitz.Pixmap(fitz.Colorspace(fitz.CS_RGB), tar_irect)
tar_pix.clearWith(50)        # clear target with a lively gray :-)

# now fill target with 3 * 4 tiles of input picture
for i in list(range(4)):
    pix0.y = i * pix0.height                          # modify input's y coord
    for j in list(range(3)):
        pix0.x = j * pix0.width                       # modify input's x coord
        tar_pix.copyPixmap(pix0, pix0.getIRect())     # copy input to new loc
        # save intermediate image to show what is happening
        fn = "target-" + str(i) + str(j) + ".png"
        tar_pix.writePNG(fn)
```

This is the input picture `editra.png` (taken from the wxPython directory `/tools/Editra/pixmaps`):

Some intermediate picture and the result:





Point

`Point` represents a point in the plane, defined by its x and y coordinates.

Attribute	Short Description
`Point.x`	The X-coordinate
`Point.y`	The Y-coordinate

Class API

class **`Point`**

`__init__` **(**`self`**,** `x`**,** `y`**)**

Constructor, without parameters defaulting to `Point(0.0, 0.0)` ("top left"). Also see the example below.

Parameters:

- **x** (*float*) -- X coordinate of the point
- **y** (*float*) -- Y coordinate of the point

Example:

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import fitz
>>> point = fitz.Point(25, 30)
>>> point
fitz.Point(25.0, 30.0)
>>>
```

Rect

`Rect` represents a rectangle defined by its top left and its bottom right Point objects, in coordinates: ((x0, y0), (x1, y1)). Respectively, a rectangle can be defined in one of the two ways: either as a pair of Point objects, or as a tuple of four coordinates.

Rectangle borders are always in parallel with the respective X- and Y-axes. A rectangle is called "finite" if x0 <= x1 and y0 <= y1 is true, else "infinite".

Methods / Attributes	Short Description
`Rect.round()`	creates the smallest IRect containing `Rect`
`Rect.transform()`	transform `Rect` with a Matrix
`Rect.height`	`Rect` height
`Rect.width`	`Rect` width
`Rect.x0`	Top left corner's X-coordinate
`Rect.y0`	Top left corner's Y-coordinate
`Rect.x1`	Bottom right corner's X-coordinate
`Rect.y1`	Bottom right corner's Y-coordinate

Class API

class **`Rect`**

`__init__` **(**`self`, `x0`, `y0`, `x1`, `y1`**)**
Constructor. Without parameters will create the empty rectangle `Rect(0.0, 0.0, 0.0, 0.0)`.

`__init__` **(**`self`, `p1`, `p2`**)**
Constructor. This will create a rectangle from two Point objects: `p1` will be taken as the top-left and `p2` as the bottom-right point of the rectangle. See the example below.

`round` **()**
Creates the smallest IRect that contains `Rect`. This is **not** simply the same as rounding each of the rectangle's coordinates!. Look at the example below.

Return type: IRect

`transform` (`m`)
Transforms `Rect` with a Matrix.

Parameters: **m** -- A Matrix to be used for the transformation.
Return type: Rect

`width`
Contains the width of the rectangle. Equals `x1 - x0`.

Return type: float

`height`
Contains the height of the rectangle. Equals `y1 - y0`.

Return type: float

`x0`
X-coordinate of the top left corner.

Type: float

`y0`
Y-coordinate of the top left corner.

Type: float

x1
X-coordinate of the bottom right corner.

Type: float

y1
Y-coordinate of the bottom right corner.

Type: float

Example 1:

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import fitz
>>> p1 = fitz.Point(10, 10)
>>> p2 = fitz.Point(300, 450)
>>> rect1 = fitz.Rect(p1, p2)
>>> rect1
fitz.Rect(10.0, 10.0, 300.0, 450.0)
>>> rect2 = fitz.Rect(0, 0, 250, 380)
>>> rect2
fitz.Rect(0.0, 0.0, 250.0, 380.0)
>>>
```

Example 2:

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import fitz
>>> r = fitz.Rect(0.5, -0.01, 123.88, 455.123456)
>>> r
fitz.Rect(0.5, -0.009999999776482582, 123.87999725341797, 455.1234436035156)
>>> r.round()
fitz.IRect(0, -1, 124, 456)
>>>
```

As can be seen, `r.round().x0 = int(math.floor(r.x0))`, `r.round().y0 = int(math.floor(r.y0))`, `r.round().x1 = int(math.ceil(r.x1))` and `r.round().y1 = int(math.ceil(r.y1))`.

TextPage

`TextPage` represents the text of a page.

Method	Short Description
`TextPage.extractText()`	Extract the page's plain text
`TextPage.extractHTML()`	Extract the page's text in HTML format
`TextPage.extractJSON()`	Extract the page's text in JSON format
`TextPage.extractXML()`	Extract the page's text in XML format
`TextPage.search()`	Search for a string in the page

Class API

class **`TextPage`**

`extractText ()`
Extract the text from a `TextPage` object. Returns a string of the page's complete text. No attempt is being made to adhere to a natural reading sequence: the text is returned UTF-8 encoded and in the same sequence as the PDF creator specified it. If this looks awkward for your PDF file, consider using program that re-arranges the text according to a more familiar layout, e.g. `PDF2TextJS.py` in the examples directory.

Return type: string

`extractHTML ()`
Extract the text from a `TextPage` object in HTML format. This version contains some more formatting information about how the text is being dislayed on the page. See the tutorial chapter for an example.

Return type: string

`extractJSON ()`
Extract the text from a `TextPage` object in JSON format. This version contains significantly more formatting information about how the text is being dislayed on the page. It is almost as complete as the `extractXML` version, except that positioning information is detailed down to the span level, not to a single character. See the tutorial chapter for an example. To process the returned JSON text use one of the json modules like `json`, `simplejson`, `ujson`, `cjson`, etc. See example program `PDF2TextJS.py` for how to do that.

Return type: string

`extractXML ()`
Extract the text from a `TextPage` object in XML format. This contains complete formatting information about every single text character on the page: font, size, line, paragraph, location, etc. This may easily reach several hundred kilobytes of uncompressed data for a text oriented page. See the tutorial chapter for an example.

Return type: string

`search (``string`**`, hit_max = 16)`**
Search for the string `string`.

Parameters:
- **string** (*string*) -- The string to search for.
- **hit_max** (*int*) -- Maximum number of expected hits (default 16).

Return type: list

Returns: A python list. If not empty, each element of the list is a Rect (without transformation) surrounding a found `string` occurrence.

TextSheet

`TextSheet` contains a list of distinct text styles used on a page (or a series of pages).

Constants and Enumerations

Constants and enumerations of MuPDF as implemented by PyMuPDF. If your import statement was `import fitz` then each of the following variables `var` is accessible as `fitz.var`.

Constants

Constant	Description
CS_RGB	1 - Type of Colorspace is RGBA
CS_GRAY	2 - Type of Colorspace is GRAY
CS_CMYK	3 - Type of Colorspace is CMYK
VersionBind	'1.8.0' - Version of PyMuPDF (this binding)
VersionFitz	'1.8' - Version of MuPDF

Enumerations

Possible values of **`linkDest.kind`** (link destination type). For details consult Adobe PDF Reference sixth edition 1.7 November 2006, chapter 8.2 on page 581 ff.

Value	Description
LINK_NONE	0 - No destination
LINK_GOTO	1 - Points to a place in this document
LINK_URI	2 - Points to an URI
LINK_LAUNCH	3 - Launch (open) another document
LINK_NAMED	4 - Perform some action
LINK_GOTOR	5 - Points to another document

Possible values of **`linkDest.flags`** (link destination flags). **Attention**: The rightmost byte of this integer is a bit field. The values represent boolean indicators showing whether the associated statement is `True`.

Value	Description
LINK_FLAG_L_VALID	1 (bit 0) Top left x value is valid
LINK_FLAG_T_VALID	2 (bit 1) Top left y value is valid
LINK_FLAG_R_VALID	4 (bit 2) Bottom right x value is valid
LINK_FLAG_B_VALID	8 (bit 3) Bottom right y value is valid
LINK_FLAG_FIT_H	16 (bit 4) Horizontal fit

LINK_FLAG_FIT_V	32 (bit 5) Vertical fit
LINK_FLAG_R_IS_ZOOM	64 (bit 6) Bottom right x is a zoom figure

Appendix

This chapter contains additional comments and examples.

A Glimpse on PyMuPDF's Performance Part 1: Parsing

We have tried to get an impression on PyMuPDF's performance. While we know this is very hard and a fair comparison is almost impossible, we feel that at least some quantitative information should be provided to justify our bold comments on MuPDF's "top performance".

We therefore address the question: How fast are the most basic functions (opening and parsing a document, then saving it back to disk as a new PDF file)?

We chose a variety of tools and a set of different files to test against.

These were the tools

All tools are either platform independant, or at least can run on Windows and Unix / Linux (pdftk).

Poppler is missing here, because it specifically is a Linux tool set, although we know there exist Windows ports (created with considerable effort apparently). Technically, it is a C/C++ library, for which a Python binding exists - in so far it is somewhat comparable to PyMuPDF. But Poppler in contrast is tightly coupled to **Qt** and **Cairo**. We may still include it in future, when a more handy Windows installation is available. We have seen however some analysis, that hints at a much lower performance than MuPDF. Our comparison of text extraction speeds also show a much lower performance of Poppler's PDF code base **Xpdf**.

Image rendering of MuPDF also is about three times faster than the one of Xpdf when comparing the command line tools `mudraw` of MuPDF and `pdftopng` of Xpdf - see part 3 of this chapter.

The sheer parsing performance cannot directly be compared, because some are batch utilities that can execute a requested task only in one go, front to end. `pdfrw` too, has a `lazy` strategy for parsing, meaning it only parses those parts of a document that are required in any moment.

Tool	Description
PyMuPDF	tool of this manual, appearing as "fitz" in reports
pdfrw	a pure Python tool, can be used as frontend to ReportLab and rst2pdf
PyPDF2	a pure Python tool with a very complete function set
pdftk	a command line utility with numerous functions

This is how each of the tools is being used with the test:

PyMuPDF:

```
doc = fitz.Document("input.pdf")
doc.save("output.pdf")
```

pdfrw:

```
doc = PdfReader("input.pdf")
writer = PdfWriter()
writer.trailer = doc
writer.write("output.pdf")
```

PyPDF2:

```
pdfmerge = PyPDF2.PdfFileMerger()
pdfmerge.append("input.pdf")
pdfmerge.write("output.pdf")
pdfmerge.close()
```

pdftk:

```
pdftk input.pdf output output.pdf
```

These were the PDF files

We have chosen a set of mostly "challenging" PDF files. Among them are a complete year of Spektrum, the German version of Scientific American, Adobe's PDF manual, and also the PDF version of this manual.

The scientific magazines are of sizes between 10 and 25 MB, each with mixture of text and lots of images. Each of it has 100 to 110 pages.

Adobe manual is the largest with more than 30 MB, mostly text and it is a linearized (!) PDF. This special property will yield interesting implications in the data following.

Our own manual with just about half a MB is here to stand for "the most simple case".

Here is an overview of the output sizes under this simple copy operation:

File Sizes (KB)	Tool				
File	0 original	1 fitz	2 pdfrw	3 PyPDF2	4 pdftk
PyMuPDF.pdf	468	459	469	472	469
sdw_2015_08.pdf	12033	11941	12041	12094	12053
sdw_2015_02.pdf	12984	12916	12991	12999	12999
Evolution.pdf	13181	13176	13181	13186	13183
sdw_2015_07.pdf	13776	13671	13781	13845	13800
sdw_2015_01.pdf	14325	14251	14318	14325	14320
sdw_2015_04.pdf	14886	14765	14881	14886	14862
sdw_2015_05.pdf	16109	16032	16118	16157	16126
sdw_2015_12.pdf	18664	18562	18666	18724	18687
sdw_2015_10.pdf	18268	18828	18908	18956	18915
sdw_2015_03.pdf	20727	20595	20734	20802	20755
sdw_2015_09.pdf	22861	22747	22862	22921	22888
sdw_2015_06.pdf	22898	22742	22903	22979	22932
sdw_2015_11.pdf	25024	27550	28137	28535	28374
Adobe.pdf	31712	49485	24446	26085	52505
Total Size	**257915**	**277720**	**254435**	**256966**	**282867**

These are our run time findings (in **seconds**, please note the European number convention: meaning of decimal point and comma is reversed):

Runtimes	tool			
file	fitz	pdfrw	pdftk	PyPDF2
Adobe.pdf	8,58	20,27	109,22	672,13
Evolution.pdf	0,14	0,41	1,20	1,32
PyMuPDF.pdf	0,20	0,17	0,74	0,88
sdw_2015_01.pdf	0,34	1,13	5,37	5,14
sdw_2015_02.pdf	0,40	1,49	6,48	6,49
sdw_2015_03.pdf	0,92	2,62	11,19	11,61
sdw_2015_04.pdf	0,45	2,14	7,16	6,91
sdw_2015_05.pdf	0,42	1,64	7,37	7,34
sdw_2015_06.pdf	1,24	3,22	13,67	13,86
sdw_2015_07.pdf	0,92	2,06	9,33	9,25
sdw_2015_08.pdf	0,50	1,86	8,31	8,31
sdw_2015_09.pdf	1,14	2,31	9,86	9,99
sdw_2015_10.pdf	0,86	1,80	3,10	6,38
sdw_2015_11.pdf	6,73	12,39	35,11	58,49
sdw_2015_12.pdf	0,59	2,10	8,51	8,91
total times	**23,43**	**55,61**	**236,62**	**827,00**

Timing Ratios			
1,00	2,37	10,10	35,30
	1,00	4,25	14,87
		1,00	3,50

If we leave out the Adobe manual, this table looks like

Runtimes	tool			
file	fitz	pdfrw	pdftk	PyPDF2
Evolution.pdf	0,14	0,41	1,20	1,32
PyMuPDF.pdf	0,20	0,17	0,74	0,88
sdw_2015_01.pdf	0,34	1,13	5,37	5,14
sdw_2015_02.pdf	0,40	1,49	6,48	6,49
sdw_2015_03.pdf	0,92	2,62	11,19	11,61
sdw_2015_04.pdf	0,45	2,14	7,16	6,91
sdw_2015_05.pdf	0,42	1,64	7,37	7,34
sdw_2015_06.pdf	1,24	3,22	13,67	13,86
sdw_2015_07.pdf	0,92	2,06	9,33	9,25
sdw_2015_08.pdf	0,50	1,86	8,31	8,31
sdw_2015_09.pdf	1,14	2,31	9,86	9,99
sdw_2015_10.pdf	0,86	1,80	3,10	6,38
sdw_2015_11.pdf	6,73	12,39	35,11	58,49
sdw_2015_12.pdf	0,59	2,10	8,51	8,91
total times	**14,85**	**35,34**	**127,40**	**154,87**

Timing Ratios			
1,00	2,38	8,58	10,43
	1,00	3,60	4,38
		1,00	1,22

Observations

PyMuPDF is by far the fastest: on average 2.4 times faster than the second best (the pure Python tool pdfrw, **chapeau pdfrw!**), and 10 times faster than the command line tool pdftk.

Where PyMuPDF only requires less than 24 seconds to process all files, pdftk affords itself almost 4 minutes.

By far the slowest tool is PyPDF2 - it is more than 35 times slower than PyMuPDF and 15 times slower than pdfrw! The main reason for PyPDF2's bad look comes from the Adobe manual. It obviously is slowed down by the linear file structure and the immense amount of bookmarks of this file. If we take out this special case, then PyPDF2 is 10.5 times slower than PyMuPDF, 4.4 times slower than pdfrw and 1.2 times slower than pdftk.

If we look at the output PDFs, there is one surprise:

Each tool created a PDF of similar size as the original. Apart from the Adobe case, PyMuPDF always created the smallest output. Adobe's manual is an exception:

The pure Python tools **reduced** its size by more than 20% (and yielded a document which is no longer linearized)!

PyMuPDF and pdftk in contrast **drastically increased** the size by 40% to about 50 MB (also no longer linearized).

So far, we have no explanation of what is happening here.

A Glimpse on PyMuPDF's Performance Part 2: Text Extraction

We also have compared text extraction speed with other tools.

The following table shows a run time comparison. PyMuPDF's methods appear as "fitz (TEXT)" and "fitz (JSON)" respectively. The tool `pdftotext.exe` of the Xpdf toolset appears as "xpdf".

- **extractText():** basic text extraction without layout re-arrangement (using `GetText(..., output = "text")`)
- **pdftotext:** a command line tool of the **Xpdf** toolset (which also is the basis of Poppler's library)
- **extractJSON():** text extraction with layout information (using `GetText(..., output = "json")`)
- **pdfminer:** a pure Python PDF tool specialized on text extraction tasks

All tools have been used with their most basic, fanciless functionality - no layout re-arrangements, etc.

For demonstration purposes, we have included a version of `GetText(doc, output = "json")`, that also re-arranges the output according to occurrence on the page.

Here are the results using the same test files as above (again: decimal point and comma reversed):

Text Extraction	Tool				
File	1 fitz (TEXT)	2 fitz (bare JSON)	3 fitz (sorted JSON)	4 xpdf	5 pdfminer
Adobe.pdf	3,86	4,94	5,73	10,57	213,10
Evolution.pdf	0,21	0,26	0,31	0,48	12,70
PyMuPDF.pdf	0,08	0,11	0,13	0,96	4,54
sdw_2015_01.pdf	0,74	0,86	0,98	1,29	42,48
sdw_2015_02.pdf	0,81	1,00	1,04	1,39	46,84
sdw_2015_03.pdf	1,47	1,63	1,71	2,34	149,51
sdw_2015_04.pdf	0,94	1,10	1,21	1,67	79,11
sdw_2015_05.pdf	0,80	0,96	1,09	1,32	47,47
sdw_2015_06.pdf	1,50	1,65	1,77	2,22	135,13
sdw_2015_07.pdf	0,79	0,93	1,02	1,40	53,87
sdw_2015_08.pdf	0,78	0,90	1,02	1,33	46,78
sdw_2015_09.pdf	1,50	1,60	1,73	2,29	155,52
sdw_2015_10.pdf	0,90	1,02	1,09	1,86	72,03
sdw_2015_11.pdf	2,03	2,15	2,23	4,31	161,43
sdw_2015_12.pdf	0,88	1,02	1,12	1,61	68,03
Total Times	**17,28**	**20,11**	**22,17**	**35,04**	**1288,52**

Timing Relationships				
1,0	1,2	1,3	2,0	74,6
	1,0	1,1	1,7	64,1
		1,0	1,6	58,1
			1,0	36,8

Again, (Py-) MuPDF is the fastest around. It is two times faster than xpdf.

JSON output is 1.7 times faster than xpdf, and even the "re-arranging" version is 1.6 times faster.

`pdfminer`, as a pure Python solution, of course is comparatively slow: MuPDF is 75 (64, 58) times faster and xpdf is 37 times faster. These observations in order of magnitude coincide with the statements on this web site.

A Glimpse on PyMuPDF's Performance Part 3: Image Rendering

We have tested rendering speed of MuPDF against the `pdftopng.exe`, a command lind tool of the **Xpdf** toolset, which is the PDF code basis of **Poppler**.

MuPDF invocation using a resolution of 150 pixels (Xpdf default):

```
mutool draw -o t%d.png -r 150 file.pdf
```

Xpdf invocation:

```
pdftopng.exe file.pdf ./
```

The resulting runtimes can be found here (again: meaning of decimal point and comma reversed):

Run Time (sec)	Tool	
File	mudraw	xpdf
Adobe.pdf	107,99	505,27
Evolution.pdf	41,05	108,33
PyMuPDF.pdf	5,74	21,82
sdw_2015_01.pdf	29,93	76,81
sdw_2015_02.pdf	29,82	74,68
sdw_2015_03.pdf	31,95	85,89
sdw_2015_04.pdf	29,16	78,09
sdw_2015_05.pdf	30,63	77,56
sdw_2015_06.pdf	30,77	87,89
sdw_2015_07.pdf	33,06	78,74
sdw_2015_08.pdf	31,26	75,95
sdw_2015_09.pdf	35,48	84,37
sdw_2015_10.pdf	29,47	77,13
sdw_2015_11.pdf	31,97	80,96
sdw_2015_12.pdf	31,30	80,68
Total	**529,61**	**1594,18**

MuPDF is between 2.7 and 4.7 (on average 3.0) times faster than Xpdf.

Example Outputs of Text Extraction Methods

This chapter provides background on the text extraction methods of PyMuPDF.

Information of interest are

- what do they provide?
- what do they imply (processing time / data sizes)?

General structure of a TextPage

Text information contained in a TextPage adheres to the following hierarchy:

```
<page> (width and height)
    <block> (its rectangle)
         <line> (its rectangle)
              <span> (its rectangle and font information)
                   <char> (its rectangle, (x, y) coordinates and value)
```

A text page consists of blocks (= roughly paragraphs).

A block consists of lines.

A line consists of spans.

A span consists of characters with the same properties. E.g. a different font will cause a new span.

GetText(page, output = "text")

This is the output of a page of this tutorial's PDF version:

```
Tutorial

This tutorial will show you the use of MuPDF in Python step by step.

Because MuPDF supports not only PDF, but also XPS, OpenXPS and EPUB formats, so does PyMuPDF

Nevertheless we will only talk about PDF files for the sake of brevity.
...
```

GetText(page, output = "html")

The HTML version looks like this:

```
<div class="page">
<div class="block"><p>
<div class="metaline"><div class="line"><div class="cell" style="width:0%;align:left"><span
</div></p></div>
<div class="block"><p>
<div class="line"><div class="cell" style="width:0%;align:left"><span class="s1">This tutori
</div></p></div>
<div class="block"><p>
<div class="line"><div class="cell" style="width:0%;align:left"><span class="s1">Because MuP
<div class="line"><div class="cell" style="width:0%;align:left"><span class="s1">Nevertheles
</div></p></div>
...
```

GetText(page, output = "json")

JSON output looks like so:

```
{
 "len":35,"width":595.2756,"height":841.8898,
 "blocks":[
  {"type":"text","bbox":[40.01575, 53.730354, 98.68775, 76.08236],
```

```
   "lines":[
      {"bbox":[40.01575, 53.730354, 98.68775, 76.08236],
       "spans":[
         {"bbox":[40.01575, 53.730354, 98.68775, 76.08236],
          "text":"Tutorial"
         }
       ]
      }
   ]
  },
  {"type":"text","bbox":[40.01575, 79.300354, 340.6957, 93.04035],
   "lines":[
      {"bbox":[40.01575, 79.300354, 340.6957, 93.04035],
       "spans":[
         {"bbox":[40.01575, 79.300354, 340.6957, 93.04035],
          "text":"This tutorial will show you the use of MuPDF in Python step by step."
         }
       ]
      }
   ]
  },
...
```

GetText(page, output = "xml")

Now the XML version:

```
<page width="595.2756" height="841.8898">
<block bbox="40.01575 53.730354 98.68775 76.08236">
<line bbox="40.01575 53.730354 98.68775 76.08236">
<span bbox="40.01575 53.730354 98.68775 76.08236" font="Helvetica-Bold" size="16">
<char bbox="40.01575 53.730354 49.79175 76.08236" x="40.01575" y="70.85036" c="T"/>
<char bbox="49.79175 53.730354 59.56775 76.08236" x="49.79175" y="70.85036" c="u"/>
<char bbox="59.56775 53.730354 64.89575 76.08236" x="59.56775" y="70.85036" c="t"/>
<char bbox="64.89575 53.730354 74.67175 76.08236" x="64.89575" y="70.85036" c="o"/>
<char bbox="74.67175 53.730354 80.89575 76.08236" x="74.67175" y="70.85036" c="r"/>
<char bbox="80.89575 53.730354 85.34375 76.08236" x="80.89575" y="70.85036" c="i"/>
<char bbox="85.34375 53.730354 94.23975 76.08236" x="85.34375" y="70.85036" c="a"/>
<char bbox="94.23975 53.730354 98.68775 76.08236" x="94.23975" y="70.85036" c="l"/>
</span>
</line>
</block>
<block bbox="40.01575 79.300354 340.6957 93.04035">
<line bbox="40.01575 79.300354 340.6957 93.04035">
<span bbox="40.01575 79.300354 340.6957 93.04035" font="Helvetica" size="10">
<char bbox="40.01575 79.300354 46.12575 93.04035" x="40.01575" y="90.050354" c="T"/>
<char bbox="46.12575 79.300354 51.685753 93.04035" x="46.12575" y="90.050354" c="h"/>
<char bbox="51.685753 79.300354 53.90575 93.04035" x="51.685753" y="90.050354" c="i"/>
<char bbox="53.90575 79.300354 58.90575 93.04035" x="53.90575" y="90.050354" c="s"/>
<char bbox="58.90575 79.300354 61.685753 93.04035" x="58.90575" y="90.050354" c=" "/>
<char bbox="61.685753 79.300354 64.46575 93.04035" x="61.685753" y="90.050354" c="t"/>
<char bbox="64.46575 79.300354 70.02576 93.04035" x="64.46575" y="90.050354" c="u"/>
<char bbox="70.02576 79.300354 72.805756 93.04035" x="70.02576" y="90.050354" c="t"/>
<char bbox="72.805756 79.300354 78.36575 93.04035" x="72.805756" y="90.050354" c="o"/>
<char bbox="78.36575 79.300354 81.695755 93.04035" x="78.36575" y="90.050354" c="r"/>
<char bbox="81.695755 79.300354 83.91576 93.04035" x="81.695755" y="90.050354" c="i"/>
...
```

Resource Requirements of Text Extraction Methods

The four text extraction methods of a TextPage differ significantly: in terms of information they supply (see above), and in terms of resource requirements. More information of course means that more processing is required and a higher data volume is generated.

For testing performance, we have run several example PDFs through these methods and found the following information. This data is not statistically secured in any way - just take it as an idea for what you should expect to see.

As a low end example we took this manual's PDF version (45+ pages, text oriented, 500 KB). The high end case was Adobe's PDF manual (1310 pages, text oriented, 32 MB). The other test cases were Spektrum magazines of the year 2015 (the German version of Scientific American, 100+ pages, text with lots of complex interspersed images, 10 to 25 MB each).

Performance

Performance of text extraction has improved significantly in MuPDF 1.8! As of updating this documentation (mid November 2015), data hint at an improvement factor greater than 2. Especially the complex extraction methods now have a much lower effort penalty.

On a higher level Win10 machine (8 processors at 4 GHz, 8 GB RAM), `extractXML()` needs anything between 0.2 and 0.5 seconds per page. This means that you can extract extremely detailed text information of a complex 100-page magazine in less than a minute. This is faster than some other free text extraction tools like e.g. Nitro 3.

With `PDF2TextJS.py` of the example directory, you have a high performance text extraction utility with a high layout faithfulness!

Data Sizes

The sizes of the returned text strings follow this pattern (`extractText()` is set to 1):

```
(Text : HTML : JSON : XML) ~ (1 : 4 : 6 : 87)
```

The number 87 for `extractXML()` corresponds to values between 200 and 400 KB per page.

Index

X

Y