PyMuPDF Documentation

version 1.8

Ruikai Liu

Jorj McKie

November 04, 2015

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 (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 it. This will give you a folder, let us call it `PyFitz`.

Step 2: Download MuPDF 1.8

Download MuPDF version 1.8 source, and unzip it. Let us call the resulting folder `mupdf18`.

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

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

On Linux and on Windows, this is not necessary: MuPDF object code is pregenerated and put in special directories:

- LibLinux - for the Linux-generated MuPDF libraries
- LibWin32 - for the Windows-generated MuPDF libraries

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.ToC()`	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

Entering the documents outline tree works like this:

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

This creates olItem as an Outline object.

Some Outline methods and attributes

Method / Attribute	Description
Outline.saveText()	Save table of contents as a text file
Outline.saveXML()	Save table of contents as a quasi-XML file
Outline.next	Next item of the same level
Outline.down	Next item one level down
Outline.title	Title of this item
Outline.dest	Destination ('where does this entry point to?')

Some Outline.dest attributes

Attribute	Description
Outline.dest.page	Target page number
Outline.dest.lt	Top-left corner of target rectangle
Outline.dest.rb	Bottem-right corner of target rectangle

MuPDF also supports outline destinations to other files and to URIs. See Outline.

In order to get a document's table of contents as a Python list, use the following function:

```
toc = doc.ToC()       # [[level, title, page], ...], or []
```

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 perform searches for text elements. At first, a page object must be created:

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

Here are some typical uses of Page objects:

Inspect the links on a Page

Here is an example that displays all links and their types:

```
#--------------------------------------------------------------------------------
# Get all links of the current page
#--------------------------------------------------------------------------------
ln = page.loadLinks()
#--------------------------------------------------------------------------------
# Links are organized as a single linked list. We need to check each occurrence
# to see what info we can get
#--------------------------------------------------------------------------------
while ln:
    if ln.dest.kind == fitz.LINK_URI:
            print '[LINK]URI: %s' % ln.dest.uri
    elif ln.dest.kind == fitz.LINK_GOTO:
        print '[LINK]jump to page %d' % ln.dest.page
    else:
        pass
    ln = ln.next
```

Render a Page

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

```
#--------------------------------------------------------------------------------
# Get the page's rectangle
#--------------------------------------------------------------------------------
rect = page.bound()
#--------------------------------------------------------------------------------
# create the smallest pixel area containing the rectangle
#--------------------------------------------------------------------------------
irect = rect.round()
#--------------------------------------------------------------------------------
# create an empty RGBA pixel map of the pixel area's size
#--------------------------------------------------------------------------------
pix = fitz.Pixmap(fitz.Colorspace(fitz.CS_RGB), irect)
pix.clearWith(255)             # Initialize with color "white" and "no transparency"
dev = fitz.Device(pix)         # Create a draw device for the pixel map
page.run(dev, fitz.Identity)   # finally render the page with no changes
#--------------------------------------------------------------------------------
# now pix contains an 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

Or we convert the image into a bitmap usable by dialog managers. **`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.

wxPython:

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

Tkinter:

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

Text extraction

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

```
dl = fitz.DisplayList()                  # create a DisplayList
ts = fitz.TextSheet()                    # create a TextSheet
tp = fitz.TextPage()                     # create a TextPage
dev = fitz.Device(ts, tp)                # create a text Device
irect = page.bound()                     # the page's visible rectangle
page.run(dev, fitz.Identity)             # run the page on the device
# now run the display list with the page's data
dl.run(dev, fitz.Identity, irect)

# 4 methods exist to extract the text now contained in the TextPage:

# (1) plain text: with line breaks, no formatting, no position info
text = tp.extractText()

# (2) html: line breaks, alignment, grouping, no formatting, no positioning
html = tp.extractHTML()

# (3) json: full formatting info (except colors and fonts) down to spans
xml = tp.extractJSON()

# (4) xml: full formatting info (except colors) down to individual characters
xml = tp.extractXML()
```

To give you an idea about the output of these alternatives, we did 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

If you are interested in the occurence of parts of text, you can determine, exactly where on a page a certain string appears:

```
# search for at most 4 page locations with specific contents
res = tp.search('MuPDF', hit_max = 4)
```

The result `res` will now be `[]` or a list of no more than 4 Rect rectangles that contain the string 'MuPDF'. The `hit_max` parameter (in our case set to 4) is optional (default is 16).

Output

Output capabilities of MuPDF (such as PDF generation) are currently very limited. However, a copy of the currently opened document can be created.

We support this with the method **`Document.save()`**. If the document had been successfully decrypted before, `save()` will 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 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 supports all options of MuPDF's standalone utility `mutool clean`.

Option	Effect

garbage = 1	garbage collect unused objects
garbage = 2	in addition to 1, compact xref tables
garbage = 3	in addition to 2, merge duplicate objects
clean = 1	clean content streams (avoid / use with care)
deflate = 1	deflate uncompressed streams
ascii = 1	convert data to ASCII format
linear = 1	create a linearized document version
expand = 1	create a decompressed version
incremental = 1	only save data that have changed

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

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 and 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

The list of PyMuPDF classes, accessible via the prefix `fitz.` if your import statement was `import fitz`

Class	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`**,** `irect`**)**
Constructor

`colorspace`
A number identifying the colorspace. Supported colorspaces are **`CS_RGB`**, **`CS_GRAY`** and **`CS_CMYK`**.

Type: int

`irect`
A IRect object representing the area of the image.

Type: instance

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** (IRect) -- 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.ToC()`	Creates 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`	Is document is 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, 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

save (`outfile`, `garbage=0`, `clean=0`, `deflate=0`, `incremental=0`, `ascii=0`, `expand=0`, `linear=0`)

Saves a copy of the document under `outfile` (include path specifications as necessary). 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 will also have occurred, e.g. broken xref tables will have been corrected as far as possible.

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.

ToC ()

A convenience function that creates a table of contents from the `outline` entries. If none exist `[]` will be returned, otherwise a Python list `[[level, title, page], [...], ...]`. Note that the title entries have already been decoded to unicode here. Page numbers are 1-based, but zero if and only if the entry points to a place outside this document.

Return type: list

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.

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`).

Return type: bool

metadata
Contains the document's meta data as a Python dictionary. 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'`). Note that if the document is encrypted, the other meta data values may be encrypted, too.
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=0`**,** `y0=0`**,** `x1=0`**,** `y1=0`**)**
Constructor. The default values will create an empty rectangle. 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

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 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

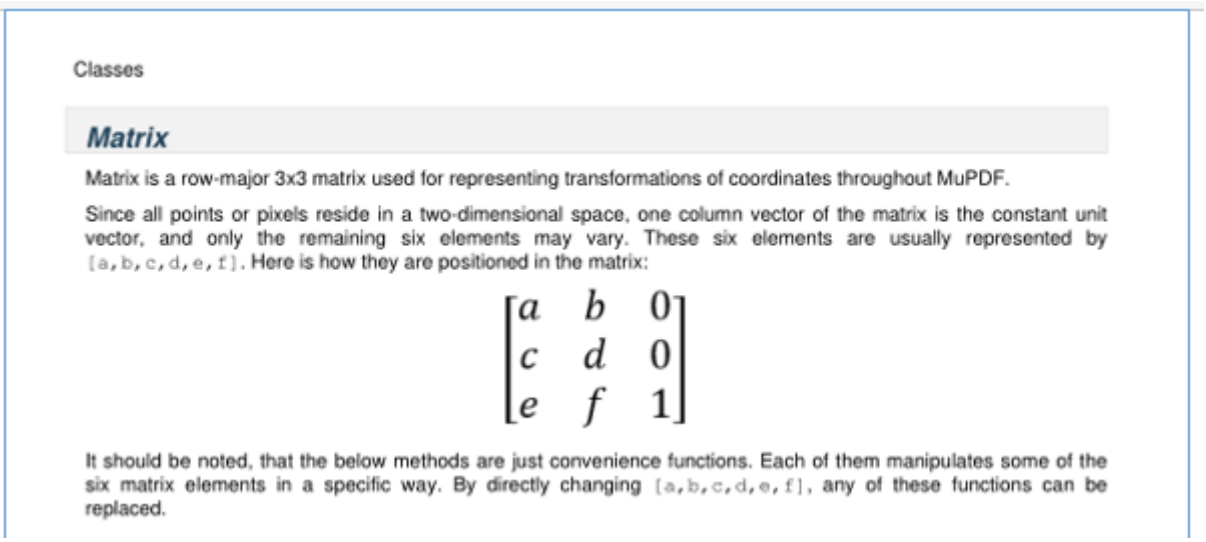
f
Causes a vertical shift effect: Each `Point(x, y)` will be shifted down to become `Point(x, y - f)`. Note that negative values of `f` will shift up.

Type: float

Examples

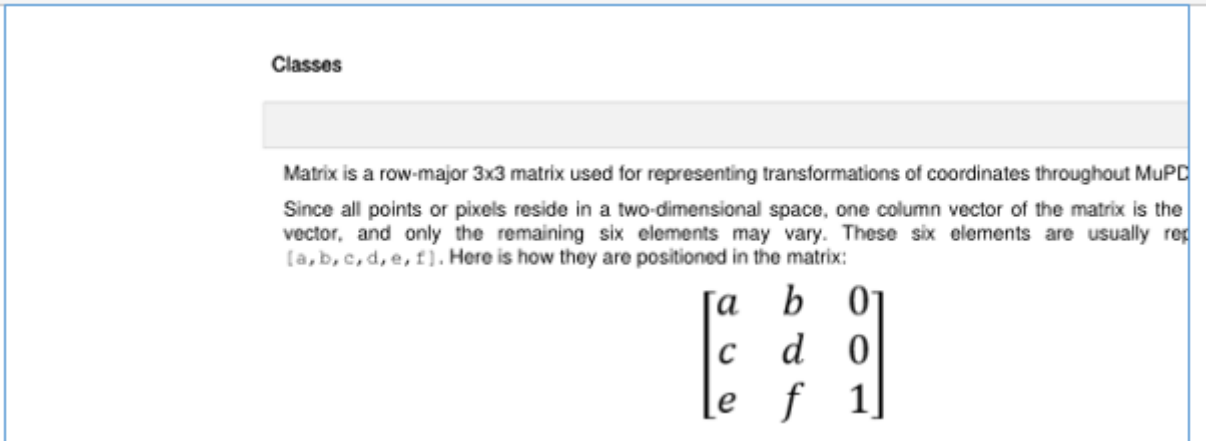
Here are examples to illustrate some of the effects achievable with matrices. The following pictures start with a page of the PDF version of this help file. We show what will happen when a matrix is being applied (though always full pages are created, only parts are displayed here to save space).

This is the original page image

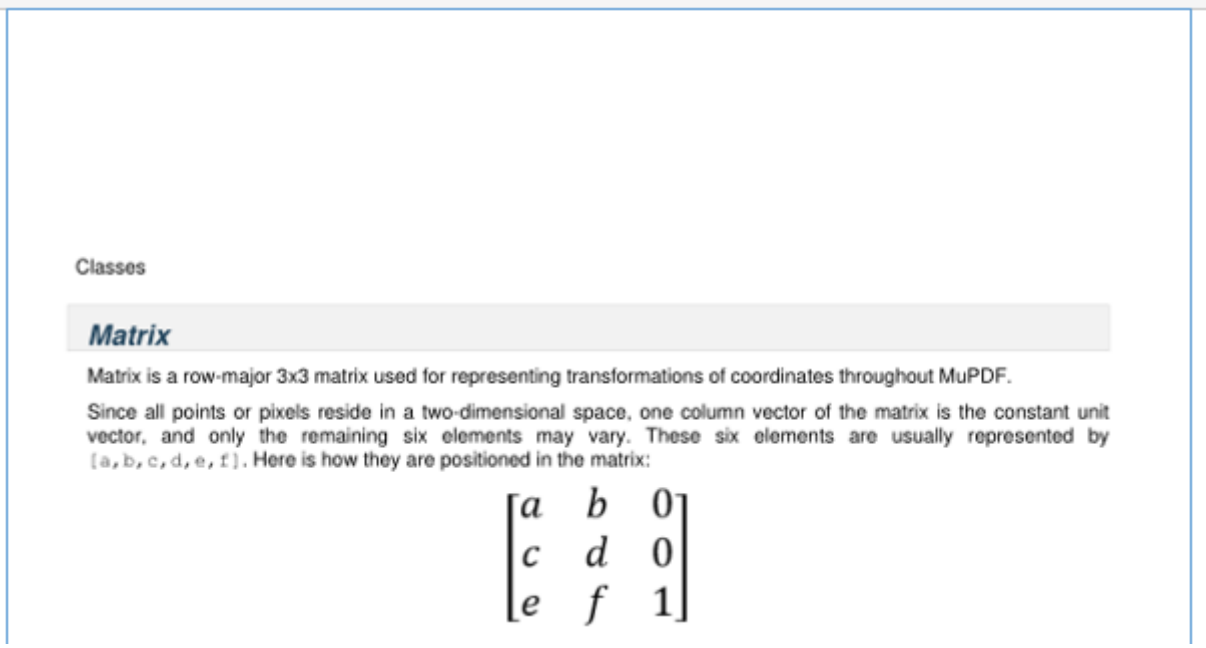


Shifting

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

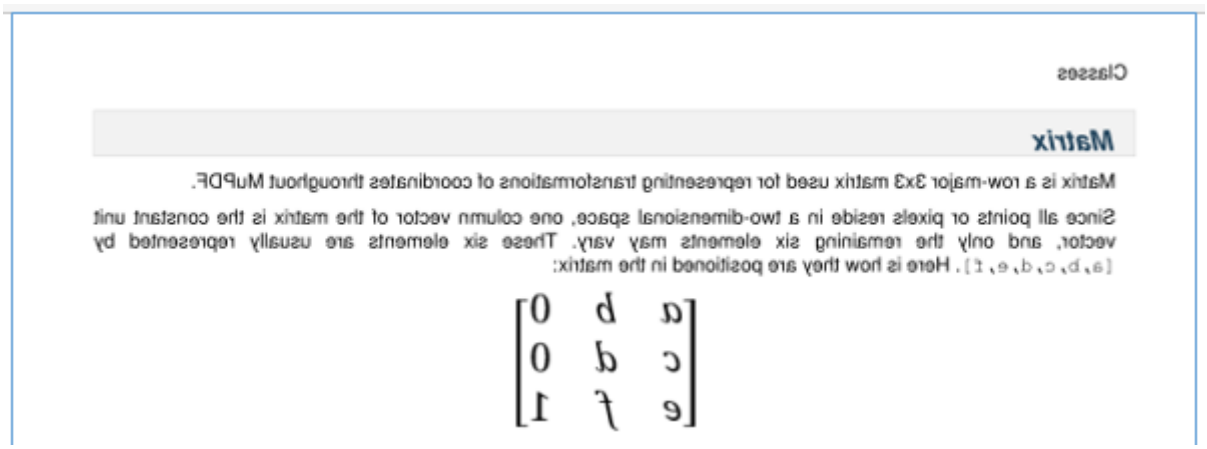


Next we do a down shift by 100 pixels: `f = 100`

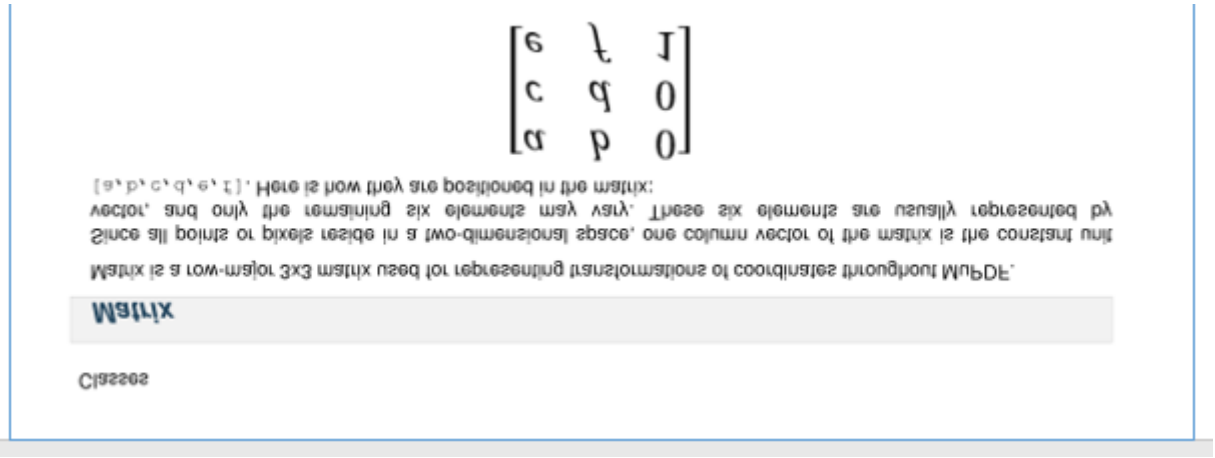


Flipping

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

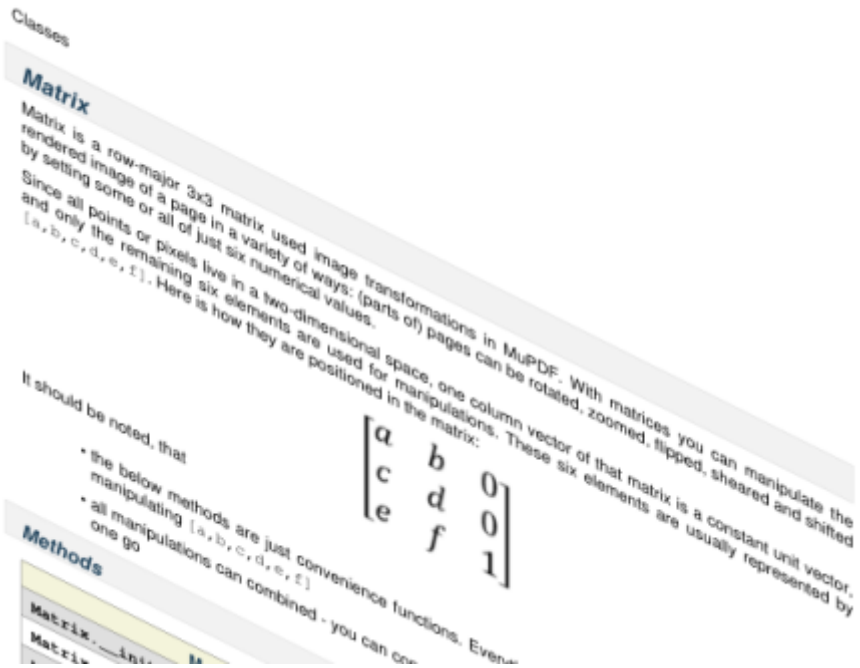


Flip horizontally (`d = -1`)



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.

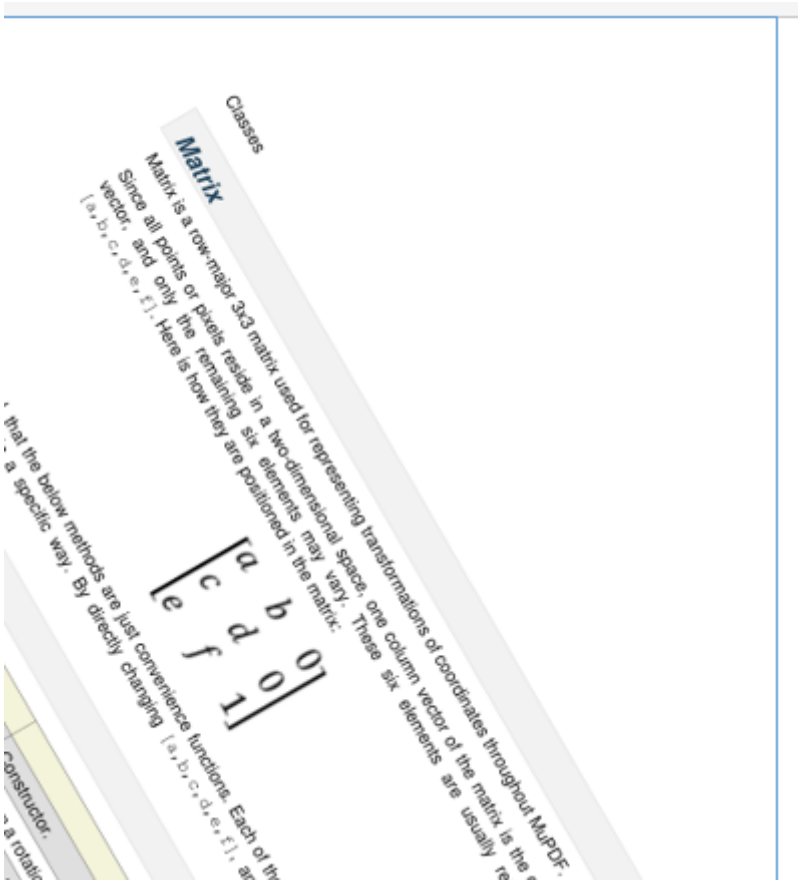
Since all points or pixels live in a two-dimensional space, one column vector of that matrix is a constant unit vector, and only the remaining six elements are used for manipulations. These six elements are usually represented by `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

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

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 all the links in 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 all the links in a page.

Return type: list

Returns: A python list of Link. An empty list is returned if there's no link in the page.

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 set of pixels for a 2 dimensional region. Each pixel consists of n bytes ("components"), plus always an alpha. The data is in premultiplied alpha when rendering, but non-premultiplied for colorspace conversions and rescaling.

Method / Attribute	Short Description
`Pixmap.clearWith()`	Clears a pixmap (with given value)
`Pixmap.writePNG()`	Saves a pixmap as a png file
`Pixmap.invertIRect()`	Invert the pixels of a given bounding box
`Pixmap.samples`	The components data for all pixels
`Pixmap.h`	Height of the region in pixels
`Pixmap.w`	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 components per pixel
`Pixmap.xres`	Resolution in X-direction
`Pixmap.yres`	Resolution in Y-direction
`Pixmap.interpolate`	Interpolation method indicator

Class API

class **`Pixmap`**

`clearWith (``self`**`, value=0)`**
Clears a pixmap.

Parameters: **value** (*int*) -- Values in the range 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.

`samples`
The color and transparency values for all pixels. Samples is a memory area of size `width * height * n` bytes. The first n bytes are components 0 to n-1 for the pixel at point (x,y). Each successive n bytes gives another pixel in scanline order. Subsequent scanlines follow each other with no padding. E.g. for an RGBA colorspace 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

`w`
The width of the region in pixels.

Return type: int

`h`
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 the chosen colorspace: **CS_GRAY** = 2, **CS_RGB** = 4, **CS_CMYK** = 5.

Return type: int

xres
Horizontal resolution in pixels per inch.

Return type: int

yres
Vertical resolution in pixels per inch

Return type: int

invertIRect (`self`**,** `irect`**)**
Invert 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 (`self`**,** `filename`**,** `savealpha=False`**)**
Save a pixmap as a png file.

Parameters:
- **filename** (*string*) -- The filename to save as (including extension).
- **savealpha** (*bool*) -- Save alpha or not.

interpolate
A 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

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=0`**,** `y=0`**)**
Constructor, defaulting to "top left".

`x`

Type: float

`y`

Type: float

Rect

`Rect` represents a rectangle defined by its top left and its bottom right Point objects, in coordinates: ((x0, y0), (x1, y1)).

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=0`**,** `y0=0`**,** `x1=0`**,** `y1=0`**)**
Constructor. The default values will create an empty rectangle.

`round ()`
Creates the smallest IRect that contains `Rect`.

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

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 (``basic=0`**`)`**
Extract the text from a `TextPage` object. Returns a string of the page's complete text. If the default value 0 for `basic` is used, the text is returned as close as possible to its natural reading order (top-left to bottom-right), and **unicode** encoded. This is based on the output of `extractXML`, see below. Usage of `basic=1` is provided primarily for debugging purposes. In this case no attempt is being made to adhere to a natutal reading sequence, instead the text is returned in the same sequence as the PDF creator specified it. In addition, in this case, the text string is UTF-8 encoded (as it is an original MuPDF value).

param `basic`**:** An integer specifying whether basic (`1` (`True`)) or advanced text output (the default) should be provided.

type `basic`**:** int

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 a single character. See the tutorial chapter for an example.

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.

Example Outputs of Text Extraction Methods

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.

TextPage.extractText()

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.
...
```

TextPage.extractHTML()

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">Neverthele
</div></p></div>
...
```

TextPage.extractJSON()

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."
         }
       ]
      }
   ]
  },
...
```

TextPage.extractXML()

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, not only in terms of information they supply (see above). More information of course means that more processing is required and a higher data volume is generated.

For testing performance, we have run 10 examples PDFs through these methods and found the following information. The following data is of course 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, completely text based, 32 MB). The other eight test cases were Spektrum

magazines January to August 2015 (the German version of Scientific American, 100+ pages, text with lots of interspersed images, 10 to 25 MB each).

Performance

Processing times of the extract methods roughly seem to follow this pattern, `extractText(basic=True)` being set to 1:

`(Text : HTML : JSON : XML) ~ (1 : 2 : 145 : 4120)`

On a higher level Win7 machine (8 processors at 4 GHz, 8 GB RAM), the figure 4120 for `extractXML()` corresponds to anything between 0.2 and 0.5 seconds per page. This still means that you can extract XML text information of a complex 100-page magazine in less than a minute. This is about 3 times faster than text extraction with other free PDF utilities, e.g. Nitro 3.

If you use `PDF2Text.py` from the examples library (a utility which converts PDFs into text files), you will see a performance based on `extractXML()`.

In the same directory you will also find a similar utility which is based on `extractJSON()`. This one is more than 20 times faster than `extractTest(basic=False)` (and thus 60+ times faster than Nitro)!

These are the details of our findings:

Processing Time Relationships			
Format:	**vs. Text**	**vs. HTML**	**vs. JSON**
Text	1		
HTML	2,1	1	
JSON	145,3	70,3	1
XML	4121,0	1998,3	28,6

Data Sizes

The sizes of the returned text strings follow this pattern, again `extractText(basic=True)` 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. The details can be seen here:

Data Size Relationships			
Format:	**vs. Text**	**vs. HTML**	**vs. JSON**
Text	1		
HTML	4,2	1	
JSON	6,1	1,4	1
XML	87,3	20,8	14,4

Examples "PyMuPDF.pdf" and "Adobe PDF Reference 1-7.pdf"

Our low and high end examples (contained in the numbers of the previous chapter) have the following detail data.

Example: PyMuPDF.pdf (45 p.)			
Processing Time Relationships			
Format:	**vs. Text**	**vs. HTML**	**vs. JSON**
Text	1		
HTML	2,2	1	
JSON	182,1	83,4	1
XML	4000,1	1831,2	22,0

Data Size Relationships			
Format:	**vs. Text**	**vs. HTML**	**vs. JSON**
Text	1		
HTML	4,8	1	
JSON	7,4	1,5	1
XML	90,1	18,6	12,1

Example: Adobe PDF Manual (1310 p.)			
Processing Time Relationships			
Format:	**vs. Text**	**vs. HTML**	**vs. JSON**
Text	1		
HTML	2,1	1	
JSON	154,0	72,9	1
XML	4070,0	1927,3	26,4

Data Size Relationships			
Format:	**vs. Text**	**vs. HTML**	**vs. JSON**
Text	1		
HTML	4,5	1	
JSON	6,7	1,5	1
XML	89,0	19,8	13,3

Index

Y