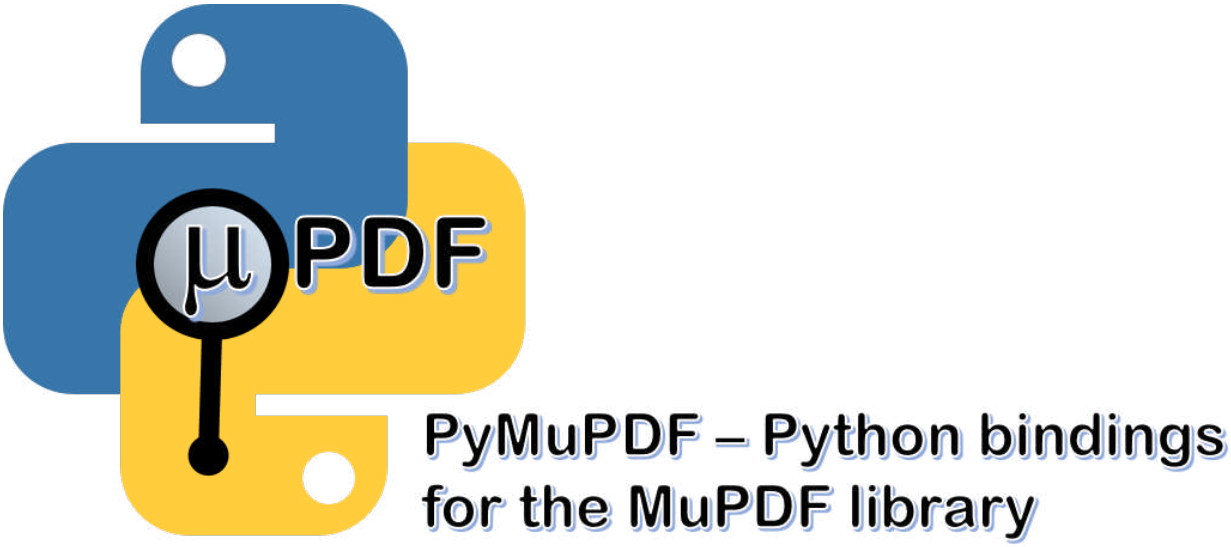


PyMuPDF 1.10 Documentation

Release 1.10.0

Jorj X. McKie

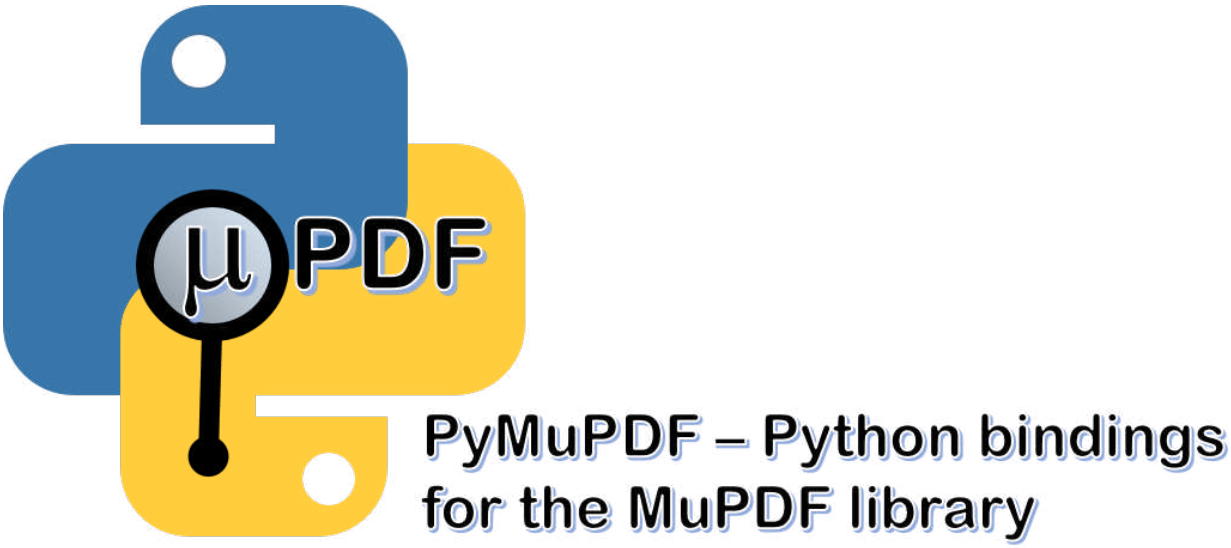
Mar 15, 2017

CONTENTS

CHAPTER

ONE

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 archive), FB2 and EPUB (e-book) formats.

These are files with extensions `*.pdf`, `*.xps`, `*.oxps`, `*.cbz`, `*.fb2` or `*.epub` (so in essence, with this binding you can develop **e-book viewers in Python** ...)

PyMuPDF provides access to many important functions of MuPDF from within a Python environment, and we are continuously seeking to expand this function set.

MuPDF stands out among all similar products for its top rendering capability and unsurpassed processing speed. At the same time, its "lightweight" makes it an excellent choice for platforms where resources are typically limited, like smartphones.

Check this out yourself and 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) - since it has changed its library basis to MuPDF!

While PyMuPDF has been available since several years for an earlier version of MuPDF (v1.2, called **fitz-python** then), it was until only mid May 2015, that its creator and a few co-workers decided to elevate it

to support current releases of MuPDF (first v1.7a, then v1.8 in November 2016, v1.9 and v1.9a since April 2016, and v1.10a in December 2016).

PyMuPDF runs and has been tested on Mac, Linux, Windows 7, Windows 10, Python 2 and Python 3, 32bit and 64bit versions. Other platforms should work too, if MuPDF and Python support them.

PyMuPDF is hosted on GitHub in this repository. Because we rely on MuPDF's C library, installation consists of two separate steps, which is the reason why we cannot support Python's standard `pip` process:

1. Installation of MuPDF: this involves downloading the source from their website and then compiling it on your machine.
2. Installation of PyMuPDF: this step is normal Python procedure. Usually you will have to adapt the `setup.py` to point to correct `include` and `lib` directories of your generated MuPDF.

For the Windows platform we were able to combine these steps and generate binaries. This installation material is contained in a separate GitHub repository and obsoletes all other download and generation work. You only need to choose which Python version and bitness you want and then download the respective ZIP file (about 3 MB).

For installation details check out the respective chapter.

We also are registered on PyPI.

There exist several demo and example programs in the main repository, ranging from simple code snippets to full-featured utilities, like text extraction, PDF joiners and bookmark maintenance.

Interesting **PDF output** functions have been added recently, covering metadata and bookmark maintenance, document restructuring, annotation handling and document output to memory areas.

1.1 Note on the Name `fitz`

The standard Python import statement for this library is `import fitz`. This has a historical reason:

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).

1.2 License

PyMuPDF is distributed under GNU GPL V3 or later.

MuPDF is distributed under a variation of it: the **GNU AFFERO GPL V3**.

Both licenses apply, when you generate and use PyMuPDF.

While in earlier days this license has been more restrictive, version 3 is effectively not any more than GNU GPL. There are just some technical details on how / where you must make available any changes have been made to the **MuPDF library** (which we haven't done and which you probably won't do either). Other

than that, nothing prevents you from distributing and even selling software you have built on the basis of (Py)MuPDF.

CHAPTER

TWO

CHANGES IN VERSION 1.10.0

2.1 MuPDF v1.10 Impact

MuPDF version 1.10 has a significant impact on our bindings. Some of the changes also affect the API - in other words, **you** as a PyMuPDF user.

- Link destination information has been reduced. Several properties of the `linkDest` class no longer contain valuable information. In fact, this class as a whole has been deleted from MuPDF's library and we in PyMuPDF only maintain it to provide compatibilty to existing code.
- In an effort to minimize memory requirements, several improvements have been built into MuPDF v1.10:
  - A new `config.h` file can be used to de-select unwanted features in the C base code. Using this feature we have been able to reduce the size of our binary `_fitz.o` / `_fitz.pyd` by about 50% (from 9 MB to 4.5 MB). When UPX-ing this, the size goes even further down to a very handy 2.3 MB.
  - The alpha (transparency) channel for pixmaps is now optional. Letting alpha default to `False` significantly reduces pixmap sizes (by 20% - CMYK, 25% - RGB, 50% - GRAY). Many `Pixmap` constructors therefore now accept an `alpha` boolean to control inclusion of this channel. Other pixmap constructors (e.g. those for file and image input) create pixmaps with no alpha alltogether. On the downside, save methods for pixmaps no longer accept a `savealpha` option: this channel will always be saved when present. In order to minimize code breaks, we have left this parameter in the call patterns - it will just be ignored.
- `DisplayList` and `TextPage` class constructors now **require the mediabox** of the page they are referring to (i.e. the `page.bound()` rectangle). There is no way to construct this information from other sources, therefore a source code change cannot be avoided in these cases. We assume however, that not many users are actually employing these rather low level classes explixitely. So the impact of that change should be minor.

2.2 Other Changes compared to Version 1.9.3

- The new *Document* method `write()` writes an opened PDF to memory (as opposed to a file, like `save()` does).

- An annotation can now be scaled and moved around on its page. This is done by modifying its rectangle.
- Annotations can now be deleted. *Page* contains the new method `deleteAnnot()`.
- Various annotation attributes can now be modified, e.g. content, dates, title (= author), border, colors.
- Method `Document.insertPDF()` now also copies annotations of source pages.
- The `Pages` class has been deleted. As documents can now be accessed with page numbers as indices (like `doc[n] = doc.loadPage(n)`), and document object can be used as iterators, the benefit of this class was too low to maintain it. See the following comments.
- `loadPage(n)` / `doc[n]` now accept arbitrary integers to specify a page number, as long as `n < pageCount`. So, e.g. `doc[-500]` is always valid and will load page `(-500) % pageCount`.
- A document can now also be used as an iterator like this: `for page in doc: ...<do something with "page"> ...`. This will yield all pages of `doc` as `page`.
- The *Pixmap* method `getSize()` has been replaced with property `size`. As before `Pixmap.size == len(Pixmap)` is `True`.
- In response to transparency (alpha) being optional, several new parameters and properties have been added to *Pixmap* and *Colorspace* classes to support determining their characteristics.
- The *Page* class now contains new properties `firstAnnot` and `firstLink` to provide starting points to the respective class chains, where `firstLink` is just a mnemonic synonym to method `loadLinks()` which continues to exist. Similarly, the new property `rect` is a synonym for method `bound()`, which also continues to exist.
- *Pixmap* methods `samplesRGB()` and `samplesAlpha()` have been deleted because pixmaps can now be created without transparency.
- *Rect* now has a property `irect` which is a synonym of method `round()`. Likewise, *IRect* now has property `rect` to deliver a *Rect* which has the same coordinates as floats values.
- Document has the new method `searchPageFor()` to search for a text string. It works exactly like the corresponding `Page.searchFor()` with page number as additional parameter.

CHAPTER

THREE

CHANGES IN VERSION 1.9.3

This version is also based on MuPDF v1.9a. Changes compared to version 1.9.2:

- As a major enhancement, annotations are now supported in a similar way as links. Annotations can be displayed (as pixmaps) and their properties can be accessed.
- In addition to the document `select()` method, some simpler methods can now be used to manipulate a PDF:
  - `copyPage()` copies a page within a document.
  - `movePage()` is similar, but deletes the original.
  - `deletePage()` deletes a page
  - `deletePageRange()` deletes a page range
- `rotation` or `setRotation()` access or change a PDF page's rotation, respectively.
- Available but undocumented before, *IRect*, *Rect*, *Point* and *Matrix* support the `len()` method and their coordinate properties can be accessed via indices, e.g. `IRect.x1 == IRect[2]`.
- For convenience, documents now support simple indexing: `doc.loadPage(n) == doc[n]`. The index may however be in range `-pageCount < n < pageCount`, such that `doc[-1]` is the last page of the document.

CHAPTER

FOUR

CHANGES IN VERSION 1.9.2

This version is also based on MuPDF v1.9a. Changes compared to version 1.9.1:

- `fitz.open()` (no parameters) creates a new empty **PDF** document, i.e. if saved afterwards, it must be given a `.pdf` extension.
- *Document* now accepts all of the following formats (`Document` and `open` are synonyms):
  - `open()`,
  - `open(filename)` (equivalent to `open(filename, None)`),
  - `open(filetype, area)` (equivalent to `open(filetype, stream = area)`).

  Type of memory area `stream` may be `str` (Python 2), `bytes` (Python 3) or `bytearray` (Python 2 and 3). Thus, e.g. `area = open("file.pdf", "rb").read()` may be used directly (without first converting it to bytearray).
- New method `Document.insertPDF()` (PDFs only) inserts a range of pages from another PDF.
- `Document` objects doc now support the `len()` function: `len(doc) == doc.pageCount`.
- New method `Document.getPageImageList()` creates a list of images used on a page.
- New method `Document.getPageFontList()` creates a list of fonts referenced by a page.
- New pixmap constructor `fitz.Pixmap(doc, xref)` creates a pixmap based on an opened PDF document and an XREF number of the image.
- New pixmap constructor `fitz.Pixmap(cspace, spix)` creates a pixmap as a copy of another one `spix` with the colorspace converted to `cspace`. This works for all colorspace combinations.
- Pixmap constructor `fitz.Pixmap(colorspace, width, height, samples)` now allows `samples` to also be `str` (Python 2) or `bytes` (Python 3), not only `bytearray`.

CHAPTER

FIVE

CHANGES IN VERSION 1.9.1

This version of PyMuPDF is based on MuPDF library source code version 1.9a published on April 21, 2016.

Please have a look at MuPDF's website to see which changes and enhancements are contained herein.

Changes in version 1.9.1 compared to version 1.8.0 are the following:

- New methods `getRectArea()` for both `fitz.Rect` and `fitz.IRect`
- Pixmaps can now be created directly from files using the new constructor `fitz.Pixmap(filename)`.
- The Pixmap constructor `fitz.Pixmap(image)` has been extended accordingly.
- `fitz.Rect` can now be created with all possible combinations of points and coordinates.
- PyMuPDF classes and methods now all contain __doc__ strings, most of them created by SWIG automatically. While the PyMuPDF documentation certainly is more detailed, this feature should help a lot when programming in Python-aware IDEs.
- A new document method of `getPermits()` returns the permissions associated with the current access to the document (print, edit, annotate, copy), as a Python dictionary.
- The identity matrix `fitz.Identity` is now **immutable**.
- The new document method `select(list)` removes all pages from a document that are not contained in the list. Pages can also be duplicated and re-arranged.
- Various improvements and new members in our demo and examples collections. Perhaps most prominently: `PDF_display` now supports scrolling with the mouse wheel, and there is a new example program `wxTableExtract` which allows to graphically identify and extract table data in documents.
- `fitz.open()` is now an alias of `fitz.Document()`.
- New pixmap method `getPNGData()` which will return a bytearray formatted as a PNG image of the pixmap.
- New pixmap method `samplesRGB()` providing a `samples` version with alpha bytes stripped off (RGB colorspaces only).
- New pixmap method `samplesAlpha()` providing the alpha bytes only of the `samples` area.
- New iterator `fitz.Pages(doc)` over a document's set of pages.

- New matrix methods `invert()` (calculate inverted matrix), `concat()` (calculate matrix product), `preTranslate()` (perform a shift operation).
- New `IRect` methods `intersect()` (intersection with another rectangle), `translate()` (perform a shift operation).
- New `Rect` methods `intersect()` (intersection with another rectangle), `transform()` (transformation with a matrix), `includePoint()` (enlarge rectangle to also contain a point), `includeRect()` (enlarge rectangle to also contain another one).
- Documented `Point.transform()` (transform a point with a matrix).
- `Matrix`, `IRect`, `Rect` and `Point` classes now support compact, algebraic formulations for manipulating such objects.
- Incremental saves for changes are possible now using the call pattern `doc.save(doc.name, incremental=True)`.
- A PDF's metadata can now be deleted, set or changed by document method `setMetadata()`. Supports incremental saves.
- A PDF's bookmarks (or table of contents) can now be deleted, set or changed with the entries of a list using document method `setToC(list)`. Supports incremental saves.

CHAPTER

SIX

INSTALLATION

Installation generally encompasses downloading and generating PyMuPDF and MuPDF from sources.

This process consists of three steps described below under **"Option 1: Install from Sources"**.

If your operating system is Windows 7 or higher (x86 or x64), you can perform a binary setup, detailed out under **"Option 2: Install from Binaries"**. This process is **a lot faster** and requires no compiler, no Visual Studio, no download of MuPDF, even no download of PyMuPDF. You only need to download those binaries from PyMuPDF-optional-material that fit your Python version.

6.1 Option 1: Install from Sources

6.1.1 Step 1: Download PyMuPDF

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

6.1.2 Step 2: Download and Generate MuPDF 1.10a

Download `mupdf-1.10a-source.tar.gz` from MuPDF version 1.10a source, now and unzip / decompress it. Call the resulting folder `mupdf`. MuPDF sources are also available on GitHub.

Make sure you download the (sub-) version for which PyMuPDF has stated its compatibility. The various Linux flavors usually have their own specific ways to support download of packages which we cannot cover here. Do not hesitate posting inquiries to our web site or sending e-mail to the authors for getting support.

Put it inside `PyFitz` as a subdirectory for keeping everything in one place.

Controlling the Binary File Size:

Since version 1.9, MuPDF includes support for many dozens of additional, so-called NOTO ("no TOFU") fonts for all sorts of alphabets from all over the world like Chinese, Japanese, Corean, Kyrillic, Indonesian, Chinese etc. If you accept MuPDF's standard here, the resulting binary for PyMuPDF will be very big and easily approach 20 MB. The features actually needed by PyMuPDF in contrast only represent a fraction of this size: no more than 5 MB currently.

To cut off unneeded stuff from your MuPDF version, modify file `/include/mupdf/config.h` as follows:

```
#ifndef FZ_CONFIG_H

#define FZ_CONFIG_H

/ *
    Choose which plotters we need.
    By default we build the greyscale, RGB and CMYK plotters in,
    but omit the arbitrary plotters. To avoid building
    plotters in that aren't needed, define the unwanted
    FZ_PLOTTERS_... define to 0.
*/
/* #define FZ_PLOTTERS_G 1 */
/* #define FZ_PLOTTERS_RGB 1 */
/* #define FZ_PLOTTERS_CMYK 1 */
/* #define FZ_PLOTTERS_N 0 */

/*
    Choose which document agents to include.
    By default all but GPRF are enabled. To avoid building unwanted
    ones, define FZ_ENABLE_... to 0.
*/
/* #define FZ_ENABLE_PDF 1 */
/* #define FZ_ENABLE_XPS 1 */
/* #define FZ_ENABLE_SVG 1 */
/* #define FZ_ENABLE_CBZ 1 */
/* #define FZ_ENABLE_IMG 1 */
/* #define FZ_ENABLE_TIFF 1 */
/* #define FZ_ENABLE_HTML 1 */
/* #define FZ_ENABLE_EPUB 1 */
/* #define FZ_ENABLE_GPRF 1 */

/*
    Choose whether to enable JavaScript.
    By default JavaScript is enabled both for mutool and PDF interactivity.
*/
// #define FZ_ENABLE_JS 1

/*
    Choose which fonts to include.
    By default we include the base 14 PDF fonts,
    DroidSansFallback from Android for CJK, and
    Charis SIL from SIL for epub/html.
    Enable the following defines to AVOID including
    unwanted fonts.
*/
/* To avoid all noto fonts except CJK, enable: */
#define TOFU            // <======================== PyMuPDF

/* To skip the CJK font, enable: */
#define TOFU_CJK        // <======================== PyMuPDF

/* To skip CJK Extension A, enable: */
#define TOFU_CJK_EXT    // <======================== PyMuPDF
```

```
/* To skip the Emoji font, enable: */
#define TOFU_EMOJI      // <======================== PyMuPDF

/* To skip the ancient/historic scripts, enable: */
#define TOFU_HISTORIC   // <======================== PyMuPDF

/* To skip the symbol font, enable: */
/* #define TOFU_SYMBOL */

/* To skip the SIL fonts, enable: */
#define TOFU_SIL        // <======================== PyMuPDF

/* To skip the Base14 fonts, enable: */
/* #define TOFU_BASE14 */
/* (You probably really don't want to do that except for measurement purposes!
↪) */

/* ---------- DO NOT EDIT ANYTHING UNDER THIS LINE ---------- */

... ... ...

#endif /* FZ_CONFIG_H */
```

The above choice should bring down your binary file size to around 5 MB or less.

Generate MuPDF now.

The MuPDF source includes generation procedures / makefiles for numerous platforms. For Windows platforms, Visual Studio solution and project definitions are provided.

Consult additional installation hints on PyMuPDF's main page on Github.com. Among other things you will find a Wiki page with details on building the Windows binaries.

6.1.3 Step 3: Build / Setup PyMuPDF

Adjust the setup.py script as necessary. 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`.

6.1.4 Using UPX

Your PyMuPDF installation will end up with four files: `__init__.py`, `fitz.py`, `utils.py` and the binary `_fitz.xxx` in the `site-packages` directory. The extension of the binary will be `.pyd` on Windows and `.so` on Linux and other platforms.

Depending on your OS, your compiler and your font support choice (see above), this binary can be quite large and range from 5 MB to 20 MB. You can reduce this by applying the compression utility UPX to it,

which exists for many operating systems. UPX will reduce the size of `_fitz.xxx` by more than 50%. You will end up with 2.5 MB to 9 MB without impacting functionality or execution speed.

6.2 Option 2: Install from Binaries

This installation option is based on pre-built binaries for Python versions on Windows 7, 8 and 10 (32bit or 64bit). Supported Python versions include 2.7 and 3.1 through 3.6 (32bit and 64bit).

6.2.1 Step 1: Download Optional Material

Download PyMuPDF-optional-material. From directory `binary_setups` select the zip file corresponding to your configuration and unzip it anywhere you like. To reduce download time, directly download the zip file corresponding to your Python version.

6.2.2 Step 2: Install PyMuPDF

Open a command prompt at the unzipped folder's directory that contains `setup.py` and enter `python setup.py install` (or `py setup.py install` if you have the Python launcher, see below).

You are done within 2 seconds.

This process requires no compiler nor Visual Studio and is **very** fast. The only pre-requisite is, that your Python configuration matches the zip file.

6.2.3 MD5 Checksums

Binary download setup scripts contain an integrity check based on MD5 check sums.

The directory structure of each zip file `pymupdf-1.10.?.?-py??-x??.zip` is as follows:

During setup, the MD5 check sum of the four installation files `__init__.py`, `_fitz.pyd`, `utils.py` and `fitz.py` is being calculated and compared against the pre-calculated check sum contained in file `md5.txt`. If a mismatch is detected, the error message

```
md5 mismatch:  probable download error
```

is issued and setup is cancelled. In this case, please check your download for any problems.

6.2.4 Targeting Parallel Python Installations

Setup scripts for binary install support the Python launcher `py.exe` introduced with version 3.3.

They contain **shebang lines** that specify the intended Python version, and additional checks for detecting error situations.

This can be used to target the right Python version if you have several installed in parallel (and of course the Python launcher, too). Use the following statement to set up PyMuPDF correctly:

```
py setup.py install
```

The shebang line of `setup.py` will be interpreted by `py.exe` to automatically find the right Python, and the internal checks will make sure that version and bitness are as they sould be.

CHAPTER

SEVEN

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, CBZ and EPUB formats, so does PyMuPDF. Nevertheless we will only talk about PDF files for the sake of brevity. At places where indeed only PDF files are supported, this will be mentioned explicitely.

As for string handling, MuPDF will pass back any string as UTF-8 encoded - no exceptions.

7.1 Importing the Bindings

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

```
import fitz
```

You can check your version by printing the docstring:

```
>>> print (fitz.__doc__)
PyMuPDF 1.9.1: Python bindings for the MuPDF 1.9a library,
built on 2016-07-01 13:06:02
>>>
```

7.2 Opening a Document

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

```
doc = fitz.open(filename)      # or 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.

It is also possible to open a document from memory (bytearray) data, i.e. without using a file. See *Document* for details.

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

7.3 Some Document Methods and Attributes

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

7.4 Accessing 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. Be aware that not all of them may be present or do contain meaningfull data.

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

7.5 Working with Outlines

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

```
toc = doc.getToC()
```

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

`lvl` is the hierarchy level of the entry (starting from 1), `title` is the entry's title, and `page` the page number (1-based). Other parameters describe details of the bookmark target.

7.6 Working 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, shifting or shearing it. You can write it's image to files, 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 (0-based)
page = doc[n]                   # short form
```

Some typical uses of *Page* objects follow:

7.6.1 Inspecting the Links of 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()*.

7.6.2 Rendering 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
```

7.6.3 Saving the Page Image in a File

We can simply store the image in a PNG file:

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

7.6.4 Displaying 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:

```
bitmap = wx.BitmapFromBufferRGB(pix.width,  # image width
            pix.height,                     # image height
            pix.samples)                    # bytearray with pixel data
```

Tkinter:

```
# the following requires: "from PIL import Image"
img = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)
photo = ImageTk.PhotoImage(img)
```

Now, `photo` can be used as an image in TK.

7.6.5 Extracting Text

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 2).
- `xml`: full formatting info (except colors) down to single characters (!).

To give you an idea about the output of these alternatives, we did text example extracts. See the Appendix 2.

7.6.6 Searching Text

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 each of which surrounds one occurrence of string "mupdf" (not case sensitive).

Please also do have a look at the demo 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.

7.7 PDF Output

Since version 1.9, PyMuPDF provides several options to modify PDF documents (only). The *Document.save()* method automatically stores a document in its current (potentially modified) state on disk.

A PDF document can be modified unnoticed by the user in two ways:

- During open, integrity checks are used to determine the health of the PDF structure. Any errors will automatically be corrected to present a repaired document in memory for further processing. If this is the case, the document is regarded as being modified.
- After a document has been decrypted, the document in memory obviously has changed and also counts as being modified.

In these cases, the save method will store a repaired and / or decrypted version, and saving **must occur to a new file**.

The following describe some more intentional ways to manipulate PDF documents. Beyond those mentioned here, you can also modify the table of contents and meta information.

7.7.1 Re-arranging and Deleting Pages

There are several ways to manipulate the page tree of a PDF:

Methods *Document.deletePage()* and *Document.deletePageRange()* delete a page (range) specified by zero-based number(s).

Methods *Document.copyPage()* and *Document.movePage()* copy or move a page to another location of the document.

Method *Document.select()* accepts a list of integers as argument. These integers must be in the range `0 <= i < pageCount`. When executed, all pages not occurring in this list will be deleted. Only pages that do occur will remain - **in the sequence specified and as many times as specified**.

So you can easily create sub-PDFs of the first / last 10 pages, only odd or even pages (for doing double-sided printing), pages that do or do not contain a certain text, ... whatever you may think of.

The saved sub-document will contain all still valid links, annotations and bookmarks.

7.7.2 Joining PDF Documents

Method *Document.insertPDF()* inserts another PDF document at a specified place of the current one. Here is a simple example (`doc1` and `doc2` are openend PDF documents):

```
# append complete doc2 to the end of doc1
doc1.insertPDF(doc2)
```

More can be found in the *Document* chapter. Also have a look at PDFjoiner.py in the repository's *example* directory.

7.7.3 Saving

As mentioned before, `save()` will automatically save a decrypted and / or repaired copy.

If you altered something, then the resulting document will be saved.

Since MuPDF 1.9, you can also write changes back to the original file by specifying `incremental = True`. This process is **extremely fast**, since changes are **appended to the original file** - it will not be rewritten as a whole.

Document.save() supports all options of MuPDF's command line utility `mutool clean`, see the following table (corresponding `mutool clean` option is indicated as "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
garbage = 4	-gggg	in addition to 3, check for duplicate streams
clean = 1	-s	clean content streams
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	append changes to the original

Be ready to experiment a little if you want to fully exploit above options: like with `mutool clean`, not all combinations may always work (or work as expected): there are just too many ill-constructed PDF files out there ...

We have found, that the combination `mutool clean -gggg -z` yields excellent compression results and is very stable. In PyMuPDF this corresponds to `doc.save(filename, garbage=4, deflate=1)`.

7.8 Closing

It is often desirable to "close" a document to relinquish control of the underlying file to the OS, while your program is still running.

This can be achieved by the *Document.close()* method. Apart from closing the underlying file, buffer areas associated with the document will be freed (if the document has been created from memory data, 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 **can no longer be used**. These bindings protect against most such invalid uses by disabling properties and methods of the document. Any *Page* object (and every *Link* or *Annot* referencing it) will automatically also be disabled.

However, re-opening a previously closed file by a new *Document* is no problem. Have a look at the following valid example:

```
doc = fitz.open(f_old)                  # open a document
<... some statements ...>               # e.g. decryption
doc.save(fnew, garbage=4, deflate=1)    # save a cleaned version
doc.close()                             # close input file
os.remove(f_old)                        # remove it
os.rename(f_new, f_old)                 # rename the cleaned version
doc = fitz.open(f_old)                  # use it as input
```

7.9 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 or decompressed, one could dynamically invoke PyMuPDF to recover from problems like so:

```
import sys
from pdfrw import PdfReader
import fitz
from io import BytesIO

#---------------------------------------
# 'tolerant' PDF reader
#---------------------------------------
def reader(fname):
    ifile = open(fname, "rb")
    idata = ifile.read()                    # put in memory
    ifile.close()
    ibuffer = BytesIO(idata)                # convert to stream
    try:
        return PdfReader(ibuffer)           # let us try
    except:                                 # problem! heal it with PyMuPDF
        doc = fitz.open("pdf", idata)       # open and save a corrected
        c = doc.write(garbage = 4)          # version in memory
        doc.close()
        doc = idata = None                  # free storage
        ibuffer = BytesIO(c)                # convert to stream
        return PdfReader(ibuffer)           # let pdfrw retry
#---------------------------------------
# Main program
#---------------------------------------
pdf = reader("pymupdf.pdf")
print pdf.Info
# do further processing
```

With the command line utility `pdftk` (available for Windows only) a similar result can be achieved, see here. However, you must invoke it as a separate process via `subprocess.Popen`, using stdin and stdout as communication vehicles.

CHAPTER

EIGHT

CLASSES

8.1 Annot

Quote from the Adobe manual: "An annotation associates an object such as a note, sound, or movie with a location on a page of a PDF document, or provides a way to interact with the user by means of the mouse and keyboard."

This class supports accessing such annotations - not only for PDF files, but for all MuPDF supported document types. However, only a few methods and properties apply to non-PDF documents.

Attribute	Short Description
Annot.getPixmap()	image of the annotation as a pixmap
Annot.setInfo()	PDF only: change metadata of an annotation
Annot.setBorder()	PDF only: changes the border of an annotation
Annot.setFlags()	PDF only: changes the flags of an annotation
Annot.setRect()	PDF only: changes the rectangle of an annotation
Annot.setColors()	PDF only: changes the colors of an annotation
Annot.updateImage()	PDF only: applies border and color values to shown image
Annot.border	PDF only: border details
Annot.colors	PDF only: border / background and fill colors
Annot.flags	PDF only: annotation flags
Annot.info	PDF only: various information
Annot.lineEnds	PDF only: start / end appearance of line-type annotations
Annot.next	link to the next annotation
Annot.parent	page object of the annotation
Annot.rect	rectangle containing the annotation
Annot.type	PDF only: type of the annotation
Annot.vertices	PDF only: point coordinates of Polygons, PolyLines, etc.

Class API

class `Annot`

`getPixmap`(*matrix = fitz.Identity*, *colorspace = fitz.csRGB*, *alpha = False*)

Creates a pixmap from the annotation as it appears on the page in untransformed coordinates. The pixmap's *IRect* equals `Annot.rect.irect` (see below).

Parameters

- **matrix** (*Matrix*) – a matrix to be used for image creation. Default is the `fitz.Identity` matrix.
- **colorspace** (*Colorspace*) – a colorspace to be used for image creation. Default is `fitz.csRGB`.
- **alpha** (*bool*) – whether to include transparency information. Default is `False`.

Return type *Pixmap*

setInfo(*d*)

Changes the info dictionary. This is includes dates, contents, subject and author (title). Changes for `name` will be ignored.

Parameters **d** (*dict*) – a dictionary compatible with the `info` property (see below). All entries must be `unicode`, `bytes`, or strings. If `bytes` values are provided in Python 3, they will be treated as being UTF8 encoded.

setRect(*rect*)

Changes the rectangle of an annotation. The annotation can be moved around and both sides of the rectangle can be independently scaled. However, the annotation appearance will never get rotated, flipped or sheared.

Parameters **rect** (*Rect*) – the new rectangle of the annotation. This could e.g. be a rectangle `rect = Annot.rect * M` with a suitable *Matrix* M (only scaling and translating will yield the expected effect).

setBorder(*value*)

PDF only: Change border width and dashing properties. Any other border properties will be deleted.

Parameters **value** (*number or dictionary*) – a number or a dictionary specifying the desired border properties. If a dictionary, its `width` and `dashes` keys are used (see property `annot.border`). If a number is specified or a dictionary like `{"width": w}`, only the border width will be changed and any dashes will remain unchanged. Conversely, with a dictionary `{"dashes": [...]}`, only line dashing will be changed. To remove dashing and get a contiguous line, specify `{"dashes": []}`.

setFlags(*flags*)

Changes the flags of the annotation. See *Annotation Flags* for possible values and use the `|` operator to combine several.

Parameters **flags** (*int*) – an integer specifying the required flags.

setColors(*d*)

Changes the colors associated with the annotation.

Parameters **d** (*dict*) – a dictionary containing color specifications. For accepted dictionary keys and values see below. The most practical way should be to first make a copy of the `colors` property and then modify this dictionary as required.

updateImage()

Attempts to modify the displayed graphical image such that it coincides with the values currently

contained in the `border` and `colors` properties. This is achieved by modifying the contents stream of the associated appearance `XObject`. Not all possible formats of content streams are currently supported: if the stream contains invocations of yet other `XObject` objects, a `ValueError` is raised.

parent
The owning page object of the annotation.

Return type *Page*

rect
The rectangle containing the annotation in untransformed coordinates.

Return type *Rect*

next
The next annotation on this page or `None`.

Return type `Annot`

type
Meaningful for PDF only: A number and one or two strings describing the annotation type, like `[2, 'FreeText', 'FreeTextCallout']`. The second string entry is optional and may be empty. `[]` if not PDF. See the appendix *Annotation Types* for a list of possible values and their meanings.

Return type list

info
Meaningful for PDF only: A dictionary containing various information. All fields are unicode or strings (Python 2 or Python 3 respectively).

- `name` - e.g. for `[12, 'Stamp']` type annotations it will contain the stamp text like `Sold` or `Experimental`.
- `content` - a string containing the text for type `Text` and `FreeText` annotations. Commonly used for filling the text field of annotation pop-up windows. For `FileAttachment` annotations it contains the filename.
- `title` - a string containing the title of the annotation pop-up window. By convention, this is used for the annotation author.
- `creationDate` - creation timestamp.
- `modDate` - last modified timestamp.
- `subject` - subject, an optional string.

Return type dict

flags
Meaningful for PDF only: An integer whose low order bits contain flags for how the annotation should be presented. See section *Annotation Flags* for details.

Return type int

lineEnds

Meaningful for PDF only: A dictionary specifying the starting and the ending appearance of annotations of types `Line`, `PolyLine`, among others. An example would be `{'start': 'None', 'end': 'OpenArrow'}`. `{}` if not specified or not applicable. For possible values and descriptions in this list, see the Adobe manual, table 8.27 on page 630.

Return type dict

vertices

Meaningful for PDF only: A list containing point ("vertices") coordinates (each given by 2 floats specifying the x and y coordinate respectively) for various types of annotations:

- `Line` - the starting and ending coordinates (4 floats).
- `[2, 'FreeText', 'FreeTextCallout']` - 4 or 6 floats designating the starting, the (optional) knee point, and the ending coordinates.
- `PolyLine` / `Polygon` - the coordinates of the edges connected by line pieces (`2 * n` floats for `n` points).
- text markup annotations - `8 * n` floats specifying the `QuadPoints` of the `n` marked text spans (see Adobe manual, page 634).
- `Ink` - list of one to many sublists of vertex coordinates. Each such sublist represents a separate line in the drawing.

Return type list

colors

Meaningful for PDF only: A dictionary of two lists of floats in range `0 <= float <= 1` specifying the common (`common`) or `stroke` and the interior (`fill`) `non-stroke` colors. The common color is used for borders and everything that is actively painted or written (*"stroked"*). The fill color is used for the interior of objects like line ends, circles and squares. The lengths of these lists implicitely determine the colorspaces used: 1 = GRAY, 3 = RGB, 4 = CMYK. So `[1.0, 0.0, 0.0]` stands for RGB and color `red`. Both lists can be `[]` if not specified. The dictionary will be empty `{}` if no PDF. The value of each float is mapped to integer values from `0 (<=> 0.0)` to `255 (<=> 1.0)`.

Return type dict

border

Meaningful for PDF only: A dictionary containing border characteristics. It will be empty `{}` if not PDF or when no border information is provided. Technically, the PDF entries `/Border`, `/BS` and `/BE` will be checked to build this information. The following keys can occur:

- `width` - a float indicating the border thickness in points.
- `effect` - a list specifying a border line effect like `[1, 'C']`. The first entry "intensity" is an integer (from 0 to 2 for maximum intensity). The second is either 'S' for "no effect" or 'C' for a "cloudy" line.
- `dashes` - a list of integers (arbitrarily limited to 10) specifying a line dash pattern in user units (usually points). `[]` means no dashes, `[n]` means equal on-off lengths of `n` points,

longer lists will be interpreted as specifying alternating on-off length values. See the Adobe manual page 217 for more details.

- `style` - 1-byte border style: `S` (Solid) = solid rectangle surrounding the annotation, `D` (Dashed) = dashed rectangle surrounding the annotation, the dash pattern is specified by the `dashes` entry, `B` (Beveled) = a simulated embossed rectangle that appears to be raised above the surface of the page, `I` (Inset) = a simulated engraved rectangle that appears to be recessed below the surface of the page, `U` (Underline) = a single line along the bottom of the annotation rectangle.

Return type dict

8.1.1 Example

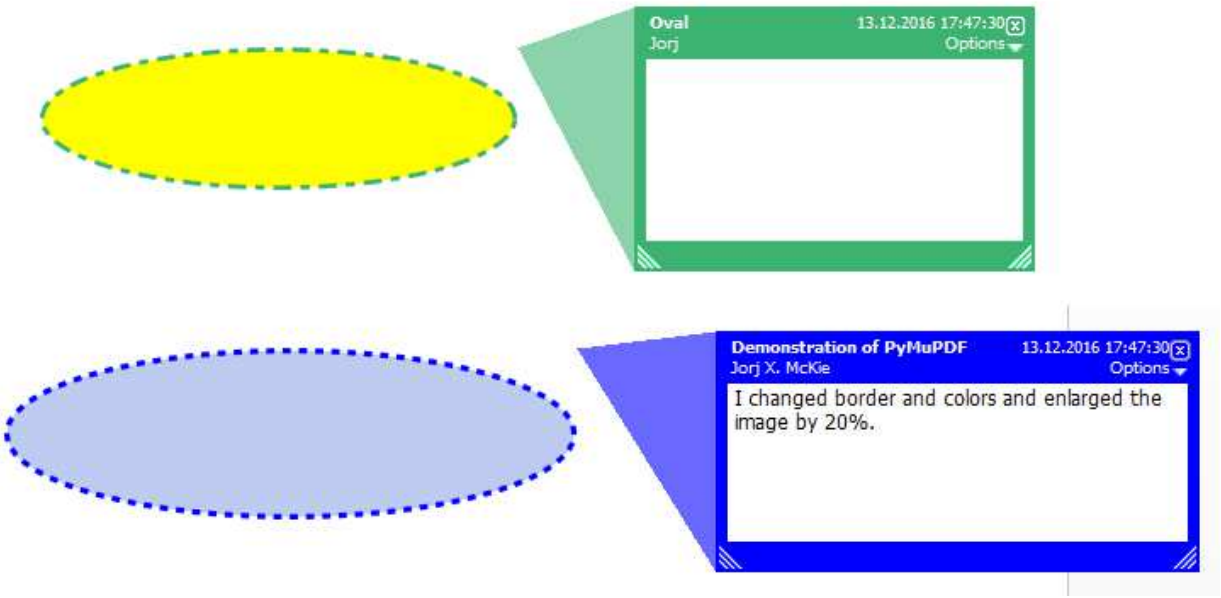
Change the graphical image of an annotation. Also update the "author" and the text to be shown in the popup window:

```
doc = fitz.open("circle-in.pdf")
page = doc[0]                           # page 0
annot = page.firstAnnot                 # get the annotation
annot.setBorder({"dashes": [3]})        # set dashes to "3 on, 3 off ..."

# set border / popup color to blue and fill color to some light blue
annot.setColors({"common":[0, 0, 1], "fill":[0.75, 0.8, 0.95]})
info = annot.info                       # get info dict
info["title"] = "Jorj X. McKie"         # author name in popup title

# text in popup window ...
info["content"] = "I changed border and colors and enlarged the image by 20%."
info["subject"] = "Demonstration of PyMuPDF"     # some readers also show this
annot.setInfo(info)                     # update info dict
r = annot.rect                          # take annot rect
r.x1 = r.x0 + r.width  * 1.2            # new location has same top-left
r.y1 = r.y0 + r.height * 1.2            # but 20% longer sides
annot.setRect(r)                        # update rectangle
annot.updateImage()                     # update appearance
doc.save("circle-out.pdf", garbage=4)   # save
```

This is how the circle annotation looks like, before and after the change:



8.2 Colorspace

Represents the color space of a *Pixmap*.

Class API

class `Colorspace`

`__init__`(*self*, *cno*)

Constructor

Parameters **`cno`** (*int*) – A number identifying the colorspace. Possible values are *CS_RGB*, *CS_GRAY* and *CS_CMYK*.

`name`

The name identifying the colorspace. Example: `fitz.csCMYK.name = 'DeviceCMYK'`.

Return type string

`n`

The number of bytes required to define the color of one pixel. Example: `fitz.csCMYK.n = 4`.

rtype int

Predefined Colorspaces

For saving some typing effort, there exist predefined colorspace objects for the three available cases.

- *csRGB* = `fitz.Colorspace(fitz.CS_RGB)`

- *csGRAY* = `fitz.Colorspace(fitz.CS_GRAY)`
- *csCMYK* = `fitz.Colorspace(fitz.CS_CMYK)`

8.3 Document

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

Since version 1.9.0 there exists an alias `open` for this class.

Method / Attribute	Short Description
Document.authenticate()	decrypt the document
Document.close()	close the document
Document.copyPage()	PDF only: copy a page to another location
Document.deletePage()	PDF only: delete a page by its number
Document.deletePageRange()	PDF only: delete a range of pages
Document.getPageFontList()	make a list of fonts on a page
Document.getPageImageList()	make a list of images on a page
Document.getPagePixmap()	create a pixmap of a page by page number
Document.getPageText()	extract the text of a page by page number
Document.getToC()	create a table of contents
Document.insertPDF()	PDF only: insert a page range from another PDF
Document.loadPage()	read a page
Document.movePage()	PDF only: move a page to another location
Document.save()	PDF only: save the document
Document.saveIncr()	PDF only: save the document incrementally
Document.searchPageFor()	search for a string on a page
Document.write()	PDF only: writes the document to memory
Document.select()	PDF only: select a subset of pages
Document.setMetadata()	PDF only: set the metadata
Document.setToC()	PDF only: set the table of contents (TOC)
Document.isClosed	has document been closed?
Document.isEncrypted	document (still) encrypted?
Document.metadata	metadata
Document.name	filename of document
Document.needsPass	require password to access data?
Document.openErrCode	> 0 if repair occurred during open
Document.openErrMsg	last error message if openErrCode > 0
Document.outline	first *Outline* item
Document.pageCount	number of pages
Document.permissions	show permissions to access the document

Class API

class `Document`

__init__(*self*[, *filename*])

Constructs a `Document` object from `filename`.

Parameters **filename** (*string*) – A string 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. If omitted or `None`, a new empty **PDF** document will be created.

Return type `Document`

Returns A `Document` object.

__init__(*self*, *filetype*, *stream*)

Constructs a `Document` object from memory `stream`.

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** (*bytearray, bytes or (Python 2 only) str*) – A memory area representing the content of a supported document type.

Return type `Document`

Returns A `Document` object.

authenticate(*password*)

Decrypts the document with the string `password`. If successful, 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 successful, `False (0)` otherwise. If successfull, indicator `isEncrypted` is set to `False`.

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) and `< doc.pageCount`. If `number < 0`, then page `number % pageCount` will be loaded (IAW `pageCount` will be added to `number` repeatedly, until the result is no longer negative). For example: in order to load the last page, you can specify `doc.loadPage(-1)`. After this you have `page.number == doc.pageCount - 1`.

Return type *Page*

Note: Conveniently, pages can also be loaded via indexes over the document: `doc.loadPage(n) == doc[n]`. Consequently, a document can also be used as an iterator over its pages, e.g. `for`

`page in doc: ...` and `for page in reversed(doc): ...` will yield the *Page* objects of `doc` as `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`, each entry of the list also contains a dictionary with *linkDest* details for each outline entry.

Return type list

Returns a list of lists. Each entry has the form `[lvl, title, page, dest]`. Its entries have the following meanings:

- lvl - hierarchy level (integer). The first entry has hierarchy level 1, and entries in a row increase by at most one level.
- title - title (string)
- page - 1-based page number (integer). Page numbers `< 1` either indicate a target outside this document or no target at all (see next entry).
- dest - included only if `simple = False` is specified. A dictionary containing details of the link destination.

getPagePixmap(*pno*, *matrix = fitz.Identity*, *colorspace = "rgb"*, *clip = None*, *alpha = False*)

Creates a pixmap from page `pno` (zero-based).

Parameters

- **pno** (*int*) – Page number, zero-based. Any value `< len(doc)` is acceptable.
- **matrix** (`Matrix`) – A transformation matrix - default is *Identity*.
- **colorspace** (*string*) – A string specifying the requested colorspace - default is `rgb`.
- **clip** (*IRect*) – An *IRect* to restrict rendering of the page to the rectangle's area. If not specified, the complete page will be rendered.
- **alpha** (*bool*) – Indicates whether transparency should be included. Leave it as `False` if not absolutely required, as it saves memory considerably (25% for RGB).

Return type *Pixmap*

getPageImageList(*pno*)

Returns a nested list of all image descriptions referenced by a page.

Parameters **pno** (*int*) – page number, zero-based. Any value `< len(doc)` is acceptable.

Return type list

Returns

a list of images shown on this page. Each entry looks like `[xref, gen, width, height, bpc, colorspace, alt. colorspace]`. Where `xref` is the image object number, `gen` its generation number (should usually be zero), `width` and `height` are the image dimensions, `bpc` denotes the number of bits per component (a typical value is 8), `colorspace` a string naming the colorspace (like `DeviceRGB`), and `alt. colorspace` is any alternate colorspace depending on the value of `colorspace`. See below how this information can be used to extract pages images as separate files. Another demonstration:

```
>>> doc = fitz.open("pymupdf.pdf")
>>> imglist = doc.getPageImageList(85)
>>> for img in imglist: print img
[1052, 0, 365, 414, 8, 'DeviceRGB', '']
>>> pix = fitz.Pixmap(doc, 1052)
>>> pix
fitz.Pixmap(fitz.csRGB, fitz.IRect(0, 0, 365, 414), 0)
```

getPageFontList (*pno*)

Returns a nested list of all fonts referenced by a page.

Parameters **pno** (*int*) – page number, zero-based. Any value `< len(doc)` is acceptable.

Return type list

Returns

a list of fonts referenced by this page. Each entry looks like `[xref, gen, type, basefont, name]`. Where `xref` is the image object number, `gen` its generation number (should usually be zero), `type` is the font type (like `Type1`, `TrueType`), `basefont` is the base font name und `name` is the PDF name of this font if given:

```
>>> doc = fitz.open("pymupdf.pdf")
>>> fontlist = doc.getPageFontList(85)
>>> for font in fontlist: print font
[100, 0, 'Type1', 'BVGEBM+NimbusSanL-Bold', '']
[102, 0, 'Type1', 'LMMQFJ+NimbusRomNo9L-Regu', '']
```

getPageText (*pno*, *output = "text"*)

Extracts the text of a page given its page number `pno` (zero-based).

Parameters

- **pno** (*int*) – Page number, zero-based. Any value `< len(doc)` is acceptable.
- **output** (*string*) – A string specifying the requested output format: text, html, json or xml. Default is `text`.

Return type String

select (*list*)

PDF only: Keeps only those pages of the document whose numbers occur in the list. Empty lists

or elements outside the range `0 <= page < doc.pageCount` will cause a `ValueError`. For more details see remarks at the bottom or this chapter.

Parameters **`list`** (*list*) – A list (or tuple) of page numbers (zero-based) to be included. Pages not in the list will be deleted (from memory) and become unavailable until the document is reopened. **Page numbers can occur multiple times and in any order:** the resulting sub-document will reflect the list exactly as specified.

Return type int

Returns Zero upon successful execution. All document information will be updated to reflect the new state of the document, like outlines, number and sequence of pages, etc. Changes become permanent only after saving the document. Incremental save is supported.

`setMetadata`(*m*)

PDF only: Sets or updates the metadata of the document as specified in `m`, a Python dictionary. As with method `select()`, these changes become permanent only when you save the document. Incremental save is supported.

Parameters **`m`** (*dict*) – A dictionary with the same keys as `metadata` (see below). All keys are optional. A PDF's format and encryption method cannot be set or changed, these keys therefore have no effect and will be ignored. If any value should not contain data, do not specify its key or set the value to `None`. If you use `m = {}` all metadata information will be cleared to `none`. If you want to selectively change only some values, modify `doc.metadata` directly and use it as the argument for this method.

Return type int

Returns Zero upon successful execution and `doc.metadata` will be updated.

`setToC`(*toc*)

PDF only: Replaces the **complete current outline** tree (table of contents) with a new one. After successful execution, the new outline tree can be accessed as usual via method `getToC()` or via property `outline`. Like with other output-oriented methods, changes become permanent only via `save()` (incremental save supported). Internally, this method consists of the following two steps. For a demonstration see example below.

- Step 1 deletes all existing bookmarks.
- Step 2 creates a new TOC from the entries contained in `toc`.

Parameters **`toc`** (*list*) –

A Python list with **all bookmark entries** that should form the new table of contents. Each entry of this list is again a list with the following format. Output variants of method `getToC()` are acceptable as input, too.

- `[lvl, title, page, dest]`, where
  - `lvl` is the hierarchy level (int > 0) of the item, starting with `1` and being at most 1 higher than that of the predecessor,

– `title` (str) is the title to be displayed.

– `page` (int) is the target page number **(attention: 1-based to support getToC()-output)**, must be in valid page range if positive. Set this to `-1` if there is no target, or the target is external.

– `dest` (optional) is a dictionary or a number. If a number, it will be interpreted as the desired height (in points) this entry should point to on `page` in the current document. Use a dictionary (like the one given as output by `getToC(simple = False)`) if you want to store destinations that are either "named", or reside outside this documennt (other files, internet resources, etc.).

Return type int

Returns `outline` and `getToC()` will be updated upon successful execution. The return code will either equal the number of inserted items (`len(toc)`) or the number of deleted items if `toc = []`.

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

PDF only: Saves the **current content of the document** under the name `outfile` (include path specifications as necessary). A document may have changed for a number of reasons: e.g. after a successful `authenticate`, a decrypted copy will be saved, and, in addition (even without optional parameters), some basic cleaning may also have occurred, e.g. broken xref tables have been repaired and earlier incremental changes have been resolved. If you executed any modifying methods like `select()`, `setMetadata()`, `setToC()`, etc., their results will also be reflected in the saved version.

Parameters

- **outfile** (*string*) – The file name to save to. Must be different from the original value value if `incremental=False`. When saving incrementally, `garbage` and `linear` **must be** `False / 0` and `outfile` **must equal** the original filename (for convenience use `doc.name`).
- **garbage** (*int*) – Do garbage collection: 0 = none, 1 = remove unused objects, 2 = in addition to 1, compact xref table, 3 = in addition to 2, merge duplicate objects, 4 = in addition to 3, check streams for duplication. Excludes `incremental`.
- **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. Excludes `garbage` and `linear`. Cannot be used for decrypted files and for files opened in repair mode (`openErrCode > 0`). In these cases saving to a new file is required.
- **ascii** (*int*) – Where possible make the output ASCII: 0 / False, 1 / True.
- **expand** (*int*) – 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. Excludes `incremental`.

Return type int

Returns Zero upon successful execution.

saveIncr()
PDF only: saves the document incrementally. This is a convenience abbreviation for `doc.save(doc.name, incremental = True)`.

Caution: A PDF may not be encrypted, but still be password protected against changes - see the `permissions` property. Performing incremental saves if `permissions["edit"] == False` can lead to unpredictable results. Save to a new file in such a case. We also consider raising an exception under this condition.

searchPageFor(*pno*, *text*, *hit_max = 16*)
Search for `text` on page number `pno`. Works exactly like the corresponding *Page.searchFor()*. Any integer `pno < len(doc)` is acceptable.

write(*garbage=0*, *clean=0*, *deflate=0*, *ascii=0*, *expand=0*, *linear=0*)
PDF only: Writes the **current content of the document** to a bytearray instead of to a file like `save()`. Obviously, you should be wary about memory requirements. The meanings of the parameters exactly equal those in *Document.save()*. The tutorial contains an example for using this method as a pre-processor to pdfrw.

Return type bytearray

Returns a bytearray containing the complete document data.

insertPDF(*docsrc*, *from_page = -1*, *to_page = -1*, *start_at = -1*, *rotate = -1*, *links = True*)
PDF only: Copies the page range **[from_page, to_page]** (including both) of the PDF document object `docsrc` into the current document. Inserts will start with page number `start_at`. Negative values can be used to indicate default values. All pages thus copied will be rotated as specified. Links can be excluded in the target, see below. All page numbers are zero-based.

Parameters

- **docsrc** (`Document`) – An opened PDF `Document` which must not be the current document object. However, it may refer to the same underlying file.
- **from_page** (*int*) – First page number in `docsrc`. Default is zero.
- **to_page** (*int*) – Last page number in `docsrc` to copy. Default is the last page.
- **start_at** (*int*) – First copied page will become page number `start_at` in the destination. If omitted, the page range will be appended to current document. If zero, the page range will be inserted before current first page.
- **rotate** (*int*) – All copied pages will be rotated by the provided value (degrees). If you do not specify a value (or `-1`), the original will not be changed.

Otherwise it must be an integer multiple of 90 (not checked). Rotation is clockwise if `rotate` is positive, else counter-clockwise.

- **links** (*bool*) – Choose whether (internal and external) links should be included with the copy. Default is `True`. An **internal** link is included only if its destination is one of the copied pages.

Return type int

Returns Zero upon successful execution.

Note: If `from_page > to_page`, pages will be copied in reverse order. If `0 <= from_page == to_page`, then one page will be copied.

Note: `doc2` bookmarks **will not be copied**. It is easy however, to recover a table of contents for the resulting document. Look at the examples below and at program `PDFjoiner.py` in the *examples* directory: it can join PDF documents and at the same time piece together respective parts of the tables of contents.

---

deletePage(*pno*)

PDF only: Deletes a page given by its 0-based number in range `0 <= pno < pageCount`.

Parameters **pno** (*int*) – the page to be deleted.

deletePageRange(*from_page = -1*, *to_page = -1*)

PDF only: Deletes a range of pages specified as 0-based numbers. Every negative value will first be replaced by `pageCount - 1`. After that, condition `0 <= from_page <= to_page < pageCount` must be true. If the parameters are equal, one page will be deleted.

Parameters

- **from_page** (*int*) – the first page to be deleted.
- **to_page** (*int*) – the last page to be deleted.

copyPage(*pno*, *to = -1*)

PDF only: Copies a page to another location.

Parameters

- **pno** (*int*) – the page to be copied. Number must be in range `0 <= pno < pageCount`.
- **to** (*int*) – the page number in front of which to insert the copy. To insert at end of document (default), specify a negative value.

movePage(*pno*, *to = -1*)

PDF only: Moves (copies and then deletes) a page to another location.

Parameters

- **pno** (*int*) – the page to be moved. Number must be in range `0 <= pno < pageCount`.

- **to** (*int*) – the page number in front of which to insert the moved page. To insert at end of document (default), specify a negative value. Must not equal (or evaluate to) `pno` or `pno + 1`.

close()
Releases objects and space allocations associated with the document. If created from a file, also closes `filename` (releasing control to the OS).

outline
Contains the first *Outline* entry of the document (or `None`). Can be used as a starting point to walk through all outline items. Accessing this property for encrypted, not authenticated documents will raise an `AttributeError`.

Return type *Outline*

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

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. Precludes incremental saves if set.

Return type bool

isEncrypted
This indicator initially equals `needsPass`. After successful authentication, it is set to `False` to reflect the situation.

Return type bool

permissions
Shows the permissions to access the document. Contains a dictionary likes this:

```
>>> doc.permissions
{'print': True, 'edit': True, 'note': True, 'copy': True}
```

The keys have the obvious meaning of permissions to print, change, annotate and copy the document, respectively.

Return type dict

metadata
Contains the document's meta data as a Python dictionary or `None` (if `isEncrypted = True` and `needPass=True`). 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 the PDF keys `/Creator`, `/Producer`, `/CreationDate`, `/ModDate`, `/Title`, `/Author`, `/Subject`, and `/Keywords` respectively.

- `format` contains the PDF version (e.g. 'PDF-1.6').
- `encryption` either contains `None` (no encryption), or a string naming an encryption method (e.g. `'Standard V4 R4 128-bit RC4'`). Note that an encryption method may be specified **even if** `needsPass = False`. In such cases not all permissions will probably have been granted. Check dictionary `permissions` for details.
- If the date fields contain valid data (which need not be the case at all!), they are strings in the PDF-specific timestamp format "D:<TS><TZ>", where
  - <TS> is the 12 character ISO timestamp `YYYYMMDDhhmmss` (`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'`, note the apostrophies!).
- A Paraguayan value might hence look like `D:20150415131602-04'00'`, which corresponds to the timestamp April 15, 2015, at 1:16:02 pm local time Asuncion.

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. Function `len(doc)` will also deliver this result.

Return type int

openErrCode

If `openErrCode > 0`, errors have occurred while opening / parsing the document, which usually means document structure issues. In this case incremental save cannot be used.

Return type int

openErrMsg

Contains either an empty string or the last error message if `openErrCode > 0`. Together with any other error messages of MuPDF's C library, it will also appear on `SYSERR`.

Return type string

Note: For methods that change the structure of a PDF (`insertPDF()`, `select()`, `copyPage()`, `deletePage()` and others), be aware that objects or properties in your program may have been invalidated or orphaned. Examples are *Page* objects and their children (links and annotations), variables holding old page counts, tables of content and the like. Remember to keep such variables up to date or delete orphaned objects.

8.3.1 Remarks on `select()`

Page numbers in the list need not be unique nor be in any particular sequence. This makes the method a versatile utility to e.g. select only the even or the odd pages, re-arrange a document from back to front, duplicate it, and so forth. In combination with text search or extraction you can also omit / include pages with no text or containing a certain text, etc.

You can execute several selections in a row. The document structure will be updated after each method execution.

Any of those changes will become permanent only with a `doc.save()`. If you have de-selected many pages, consider specifying the `garbage` option to eventually reduce the resulting document's size (when saving to a new file).

Also note, that this method **preserves all links, annotations and bookmarks** that are still valid. In other words: deleting pages only deletes references which point to de-selected pages. Page number of bookmarks (outline items) are automatically updated when a TOC is retrieved again with `getToC()`. If a bookmark's destination page happened to be deleted, then its page number in `getToC()` will be set to `-1`.

The results of this method can of course also be achieved using combinations of methods `copyPage()`, `deletePage()` and `movePage()`. While there are many cases, when these methods are more practical, `select()` is easier and safer to use when many pages are involved.

8.3.2 `select()` Examples

In general, any list of integers within the document's page range can be used. Here are some illustrations.

Delete pages with no text:

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(len(doc)))                    # list of page numbers

for page in doc:
    if not page.getText():                   # page contains no text
        r.remove(page.number)                # remove page number from list

if len(r) < len(doc):                        # did we actually delete anything?
    doc.select(r)                            # apply the list
doc.save("out.pdf", garbage = 4)             # save result to new PDF, OR

# update the original document ... *** VERY FAST! ***
doc.saveIncr()
```

Create a sub document with only the odd pages:

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(0, len(doc), 2))
doc.select(r)                                # apply the list
doc.save("oddpages.pdf", garbage = 4)        # save sub-PDF of the odd pages
```

Concatenate a document with itself:

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(len(doc)))
r += r                                  # turn PDF into a copy of itself
doc.select(r)
doc.save("any-any.pdf")                 # contains doubled <any.pdf>
```

Create document copy in reverse page order (well, don't try with a million pages):

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(len(doc) - 1, -1, -1))
doc.select(r)
doc.save("back-to-front.pdf")
```

8.3.3 `setMetadata()` Example

Clear metadata information. If you do this out of privacy / data protection concerns, make sure you save the document as a new file with `garbage > 0`. Only then the old `/Info` object will also be physically removed from the file. In this case you may also want to clear any XML metadata inserted by some PDF editors:

```
>>> import fitz
>>> doc=fitz.open("pymupdf.pdf")
>>> doc.metadata              # look at what we currently have
{'producer': 'rst2pdf, reportlab', 'format': 'PDF 1.4', 'encryption': None,
↪'author':
'Jorj X. McKie', 'modDate': "D:20160611145816-04'00'", 'keywords': 'PDF, XPS,␣
↪EPUB, CBZ',
'title': 'The PyMuPDF Documentation', 'creationDate': "D:20160611145816-04'00'
↪",
'creator': 'sphinx', 'subject': 'PyMuPDF 1.9.1'}
>>> doc.setMetadata({})       # clear all fields
0
>>> doc.metadata              # look again to show what happened
{'producer': 'none', 'format': 'PDF 1.4', 'encryption': None, 'author': 'none
↪',
'modDate': 'none', 'keywords': 'none', 'title': 'none', 'creationDate': 'none
↪',
'creator': 'none', 'subject': 'none'}
>>> doc._delXmlMetadata()     # clear any XML metadata
0
>>> doc.save("anonymous.pdf", garbage = 4)        # save anonymized doc
0
```

8.3.4 `setToC()` Example

This shows how to modify or add a table of contents. Also have a look at `csv2toc.py` and `toc2csv.py` in the examples directory:

```
>>> import fitz
>>> doc = fitz.open("test.pdf")
>>> toc = doc.getToC()
>>> for t in toc: print(t)                                  # show what we have
...
[1, 'The PyMuPDF Documentation', 1]
[2, 'Introduction', 1]
[3, 'Note on the Name fitz', 1]
[3, 'License', 1]
>>> toc[1][1] += " modified by setToC"                      # modify something
>>> doc.setToC(toc)                                        # replace outline tree
3                                                          # number of bookmarks␣
↪inserted
>>> for t in doc.getToC(): print(t)                        # demonstrate it worked
...
[1, 'The PyMuPDF Documentation', 1]
[2, 'Introduction modified by setToC', 1]                  # <<< this has changed
[3, 'Note on the Name fitz', 1]
[3, 'License', 1]
```

8.3.5 `insertPDF()` Examples

(1) Concatenate two documents including their TOCs:

```
doc1 = fitz.open("file1.pdf")           # must be a PDF
doc2 = fitz.open("file2.pdf")           # must be a PDF
pages1 = len(doc1)                      # save doc1's page count
toc1 = doc1.getToC(simple = False)      # save TOC 1
toc2 = doc2.getToC(simple = False)      # save TOC 2
doc1.insertPDF(doc2)                    # doc2 at end of doc1
for t in toc2:                          # increase toc2 page numbers
    t[2] += pages1                      # by old len(doc1)
doc1.setToC(toc1 + toc2)                # now result has total TOC
```

Obviously, similar ways can be found in more general situations. Just make sure that hierarchy levels in a row do not increase by more than one. Inserting dummy bookmarks before and after `toc2` segments would heal such cases.

(2) More examples:

```
# insert 5 pages of doc2, where its page 21 becomes page 15 in doc1
doc1.insertPDF(doc2, from_page = 21, to_page = 25, start_at = 15)

# same example, but source pages are rotated and in reverse order
doc1.insertPDF(doc2, from_page = 25, to_page = 21, start_at = 15, rotate = 90)
```

```
# insert doc2 pages in front of doc1
doc1.insertPDF(doc2, from_page = 21, to_page = 25, start_at = 0)
```

8.3.6 Other Examples

Extract all page-referenced images of a PDF into separate PNG files:

```
for i in range(len(doc)):
    imglist = doc.getPageImageList(i)
    for img in imglist:
        xref = img[0]                    # xref number
        pix = fitz.Pixmap(doc, xref)     # make pixmap from image
        if pix.colorspace != "DeviceCMYK": # can be saved as PNG
            pix.writePNG("p%s-%s.png" % (i, xref))
        else:                            # CMYK: must convert first
            pix0 = fitz.Pixmap(fitz.csRGB, pix)
            pix0.writePNG("p%s-%s.png" % (i, xref))
            pix0 = None                  # free Pixmap resources
        pix = None                       # free Pixmap resources
```

Rotate all pages of a PDF:

```
for page in doc:
    page.setRotation(90)
```

8.4 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.

Identity is a constant, an "immutable" object. So, all of its matrix properties are read-only and its methods are disabled.

If you need a do-nothing matrix as a starting point, use `fitz.Matrix(1, 1)` or `fitz.Matrix(0)` instead, like so:

```
>>> fitz.Matrix(0).preTranslate(2, 5)
fitz.Matrix(1.0, 0.0, -0.0, 1.0, 2.0, 5.0)
```

8.5 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 / Method	Short Description
IRect.getRect()	return a *Rect* with same coordinates
IRect.getRectArea()	calculate the area of the rectangle
IRect.intersect()	common part with another rectangle
IRect.translate()	shift rectangle
IRect.height	height of the rectangle
IRect.rect	equals result of method getRect()
IRect.width	width of the rectangle
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.

Parameters

- **x0** (*int*) – Top-left x coordinate.
- **y0** (*int*) – Top-left y coordinate.
- **x1** (*int*) – Bottom-right x coordinate.
- **y1** (*int*) – Bottom-right y coordinate.

getRect()

A convenience function returning a *Rect* with the same coordinates as floating point values.

Return type *Rect*

getRectArea(*unit = 'pt'*)

Calculates the area of the rectangle.

Parameters **unit** (*string*) – Specify the unit: pt (square pixel points, default) or mm (square millimeters).

Return type float

intersect(*ir*)

The intersection (common rectangular area) of the current rectangle and ir is calculated and replaces the current rectangle. If either rectangle is empty, the result is also empty. If one of the rectangles is infinite, the other one is taken as the result - and hence also infinite if both rectangles were infinite.

Parameters **ir** (*IRect*) – Second rectangle.

translate(*tx*, *ty*)
Modifies the rectangle to perform a shift in x and / or y direction.

Parameters

- **tx** (*int*) – Number of pixels to shift horizontally. Negative values mean shifting left.
- **ty** (*int*) – Number of pixels to shift vertically. Negative values mean shifting down.

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

8.5.1 Remark

A rectangle's coordinates can also be accessed via index, e.g. `r.x0 == r[0]`.

8.5.2 IRect Algebra

A number of arithmetics operations have been defined for the `IRect` class.

- **Addition:** `ir + x` where `ir` is an `IRect` and `x` is a number, `Rect` or `IRect`. The result is a new `IRect` with added components of the operands. If `x` is a number, it is added to all components of `ir`.
- **Subtraction:** analogous to addition.
- **Negation:** `-ir` is a new `IRect` with negated components of `ir`.

- **Inclusion:** `ir | x` is the new `IRect` that also includes `x`, which can be a `Rect`, `IRect` or `Point`.
- **Intersection:** `ir & x` is a new `IRect` containing the area common to `ir` and `x` which can be a `Rect` or `IRect`.
- **Multiplication:** `ir * m` is a new `IRect` containing `ir` transformed with matrix `m`.

8.5.3 Examples

Algebra provides handy ways to perform inclusion and intersection checks between Rects, IRects and Points.

Example 1:

```
>>> ir = fitz.IRect(10, 10, 410, 610)
>>> ir
fitz.IRect(10, 10, 410, 610)
>>> ir.height
600
>>> ir.width
400
>>> ir.getRectArea(unit = 'mm')      # calculate area in square millimeters
29868.51852
```

Example 2:

```
>>> m = fitz.Matrix(45)
>>> ir = fitz.IRect(10, 10, 410, 610)
>>> ir * m                           # rotate rectangle by 45 degrees
fitz.IRect(-425, 14, 283, 722)
>>>
>>> ir | fitz.Point(5, 5)            # enlarge rectangle to contain a point
fitz.IRect(5, 5, 410, 610)
>>>
>>> ir + 5                           # shift the rect by 5 points
fitz.IRect(15, 15, 415, 615)
>>>
>>> ir & fitz.Rect(0.0, 0.0, 15.0, 15.0)
fitz.IRect(10, 10, 15, 15)
```

Example 3:

```
>>> # test whether two rectangle are disjoint
>>> if list(r1 & r2) == [0, 0, 0, 0]: print("disjoint rectangles")
>>>
>>> # test whether r2 containes x (x is Point, Rect or IRect)
>>> if list(r2 | x) == list(r2): print("x is contained in r2")
```

8.6 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.uri	link destination
Link.isExternal	link destination
Link.next	points to next link
Link.dest	points to link destination details

Class API

class Link

rect
The area that can be clicked in untransformed coordinates.

Return type *Rect*

isExternal
A bool specifying whether the link target is outside (`True`) of the current document.

Return type bool

uri
A string specifying the link target. The meaning of this property should be evaluated in conjunction with property `isExternal`. The value may be `None`, in which case `isExternal == False`. If `uri` starts with `file://`, `mailto:`, or an internet resource name, `isExternal` is `True`. In all other cases `isExternal == False` and `uri` points to an internal location. In case of PDF documents, this should either be `#nnnn` to indicate a 1-based (!) page number `nnnn`, or a named location. The format varies for other document types, e.g. `uri = '../FixedDoc.fdoc#PG_2_LNK_1'` for page number 2 (1-based) in an XPS document.

Return type str

next
The next `Link` or `None`

Return type `Link`

dest
The link destination details object.

Return type *linkDest*

8.7 linkDest

Class representing the *dest* property of an outline entry or a link. Describes the destination to which such entries point.

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 a 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
Target destination name if *linkDest.kind* is *LINK_GOTOR* and *linkDest.page* is -1.

Return type string

fileSpec
Contains the filename and path this link points to, if *linkDest.kind* is *LINK_GOTOR* or *LINK_LAUNCH*.

Return type string

flags
A bitfield 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. But the flags indicate which of the values were actually specified, see *Link Destination Flags*.

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 (as opposed to e.g. a local file specification in URI format).

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 *Enumerations* 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 action to perform (e.g. a javascript, see Adobe PDF documentation). Standard actions provided are `NextPage`, `PrevPage`, `FirstPage`, and `LastPage`.

Return type string

newWindow
If true, the destination should be launched in a new window.

Return type bool

page
The page number (in this or the target document) this destination points to. Only set if *linkDest.kind* is *LINK_GOTOR* or *LINK_GOTO*. May be `-1` if *linkDest.kind* is *LINK_GOTOR*. In this case *linkDest.dest* contains the **name** of a destination in the target document.

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

8.8 Matrix

Matrix is a row-major 3x3 matrix used by image transformations in MuPDF (which complies with the respective concepts laid down in the Adobe manual). 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 float 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. If you however choose to do this, do have a look at the **remarks** further down or at the Adobe manual.

Method / Attribute	Description
Matrix.preRotate()	perform a rotation
Matrix.preScale()	perform a scaling
Matrix.preShear()	perform a shearing (skewing)
Matrix.preTranslate()	perform a translation (shifting)
Matrix.concat()	perform a matrix multiplication
Matrix.invert()	calculate the inverted matrix
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*, *sx*, *sy*[, *shear*])

Constructor. Creates a matrix with scale or shear factors `sx`, `sy` in x and y direction, respectively. The boolean `shear` controls the meaning of the other two paramters. `fitz.Matrix(1, 1)` creates a modifyable version of the *Identity* matrix, which looks like `[1, 0, 0, 1, 0, 0]`.

Parameters

- **`sx`** (*float*) – Scale or shear factor in x direction as controlled by `shear`.
- **`sy`** (*float*) – Scale or shear factor in y direction as controlled by `shear`.
- **`shear`** (*bool*) – Controls whether `sx` and `sy` should be treated as scale or as shear factors. If shear is `False` (default), the scaling matrix `[sx, 0, 0, sy, 0, 0]` will be created. If `shear` is `True`, the shearing matrix `[1, sx, sy, 1, 0, 0]` will be created.

__init__(*self*, *m*)
Constructor. Creates **a new copy** of matrix m.

Parameters **m** (*Matrix*) – The matrix to copy from.

__init__(*self*, *deg*)
Constructor. Creates a matrix that performs a rotation by `deg` degrees. See method `preRotate()` for details. `fitz.Matrix(0)` creates a modifyable version of the *Identity* matrix.

Parameters **deg** (*float*) – Rotation degrees.

preRotate(*deg*)
Modify the matrix to perform a counterclockwise rotation for positive `deg` degrees, else clockwise. The matrix elements of an identity matrix will change in the following way:

`[1, 0, 0, 1, 0, 0] -> [cos(deg), sin(deg), -sin(deg), cos(deg), 0, 0]`.

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

preScale(*sx*, *sy*)
Modify the matrix to scale by the zoom factors sx and sy. Has effects on attributes `a` thru `d` only: `[a, b, c, d, e, f] -> [a*sx, b*sx, c*sy, d*sy, e, f]`.

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`.

preShear(*sx*, *sy*)
Modify the matrix to perform a shearing, i.e. transformation of rectangles into parallelograms (rhomboids). Has effects on attributes `a` thru `d` only: `[a, b, c, d, e, f] -> [c*sy, d*sy, a*sx, b*sx, e, f]`.

Parameters

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

preTranslate(*tx*, *ty*)
Modify the matrix to perform a shifting / translation operation along the x and / or y axis. Has effects on attributes `e` and `f` only: `[a, b, c, d, e, f] -> [a, b, c, d, tx*a + ty*c, tx*b + ty*d]`.

Parameters

- **tx** (*float*) – Translation effect in X direction. See attribute `e`.
- **ty** (*float*) – Translation effect in Y direction. See attribute `f`.

concat(*m1*, *m2*)

Calculate the matrix product `m1 * m2` and store the result in the current matrix. Any of `m1` or `m2` may be the current matrix. Be aware that matrix multiplication is not commutative. So the sequence of `m1`, `m2` is important.

Parameters

- **m1** (*Matrix*) – First (left) matrix.
- **m2** (*Matrix*) – Second (right) matrix.

invert(*m*)

Calculate the matrix inverse of `m` and store the result in the current matrix. Returns `1` if `m` is not invertible ("degenerate"). In this case the current matrix **will not change**. Returns `0` if `m` is invertible, and the current matrix is replaced with the inverted `m`.

Parameters **m** (*Matrix*) – Matrix to be inverted.

Return type int

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 left-right flip will (additionally) occur.

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, an up-down flip will (additionally) occur.

Type float

e

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

Type float

f

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

Type float

8.8.1 Remarks 1

For a matrix `m`, properties `a` to `f` can also be accessed by index, e.g. `m.a == m[0]` and `m[0] = 1` has the same effect as `m.a = 1`.

8.8.2 Remarks 2

Obviously, changes of matrix properties and execution of matrix methods can be combined, i.e. executed consecutively. This is done by multiplying the respective matrices.

Matrix multiplications are **not commutative**, i.e. execution sequence determines the result: a **shift-rotate** is not equal a **rotate-shift** in general. So it can easily become unclear which result a transformation will yield. E.g. if you apply `preRotate(x)` to an arbitrary matrix `[a, b, c, d, e, f]` you will get the matrix `[a*cos(x)+c*sin(x), b*cos(x)+d*sin(x), -a*sin(x)+c*cos(x), -b*sin(x)+d*cos(x), e, f]` ...

In order to keep results foreseeable for a series of transformations, Adobe recommends the following sequence (see page 206 of their manual):

1. Shift ("translate")
2. Rotate
3. Scale or shear ("skew")

8.8.3 Matrix Algebra

A number of arithmetics operations have been defined for the `Matrix` class. In what follows, m, m1, m2 are matrices:

- **Addition:** with `m1 + m2` is a new matrix containing `[m1.a + m2.a, ..., m1.f + m2.f]`
- **Subtraction:** analogous to addition
- **Multiplication:** `m1 * m2` is a new matrix calculated as `concat(m1, m2)`
- **Negation:** `-m` is the new matrix `[-m.a, -m.b, ...]`
- **Inversion:** `~m` is the new matrix such that `m * ~m = fitz.Identity`. If `m` is degenerate (not invertible), `~m` will be `[0, 0, 0, 0, 0, 0]`.
- **Absolute Value:** `abs(m)` is a float containing the Euclidean norm of `m`. Typically used for testing whether two matrices are "almost equal", like `abs(m1 - m2) < epsilon`.
- **Non-Zero-Test:** You can test whether a matrix is all zero (`[0, 0, 0, 0, 0, 0]`): `if not ~m: print "m is not invertible"`

This makes the following operations possible:

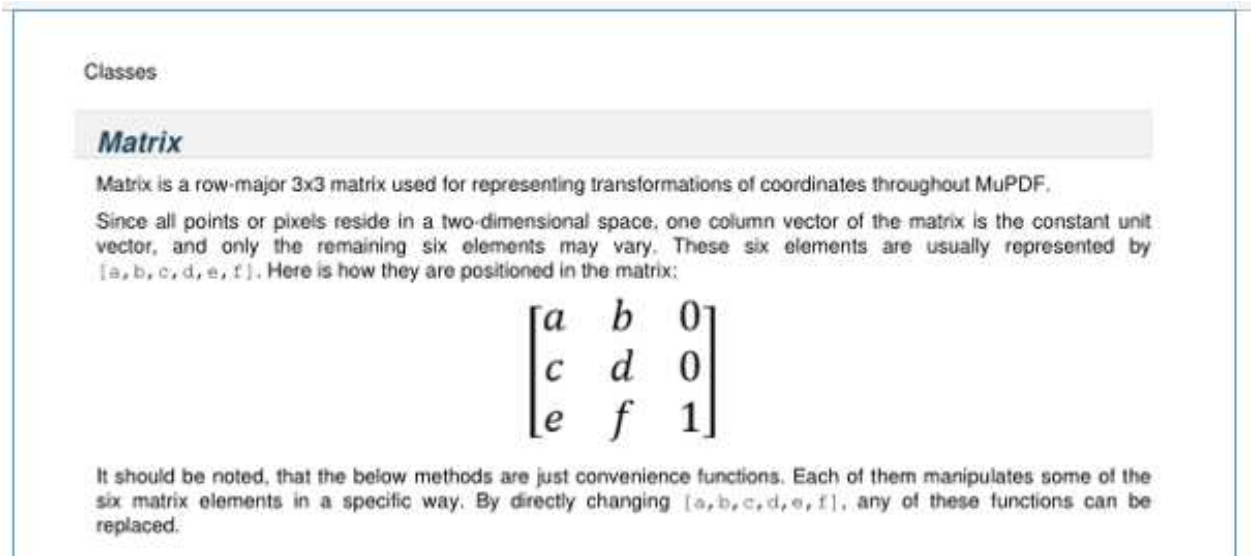
```
>>> import fitz
>>> m45p = fitz.Matrix(45)           # rotate 45 degrees counterclockwise
>>> m45m = fitz.Matrix(-45)          # rotate 45 degrees clockwise
>>> m90p = fitz.Matrix(90)           # rotate 90 degrees counterclockwise
```

```
>>>
>>> abs(m90p - m45p * m45p)            # should be (close to) zero
8.429369702178807e-08
>>>
>>> abs(m45p * m45m - fitz.Identity)   # should be (close to) zero
2.1073424255447017e-07
>>>
>>> abs(m45p - ~m45m)                  # should be (close to) zero
2.384185791015625e-07
>>>
>>> m90p * m90p * m90p * m90p          # should be 360 degrees = fitz.Identity
fitz.Matrix(1.0, -0.0, 0.0, 1.0, 0.0, 0.0)
```

8.8.4 Examples

Here are examples to illustrate some of the effects achievable. The following pictures start with a page of the PDF version of this help file. We show what happens 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.

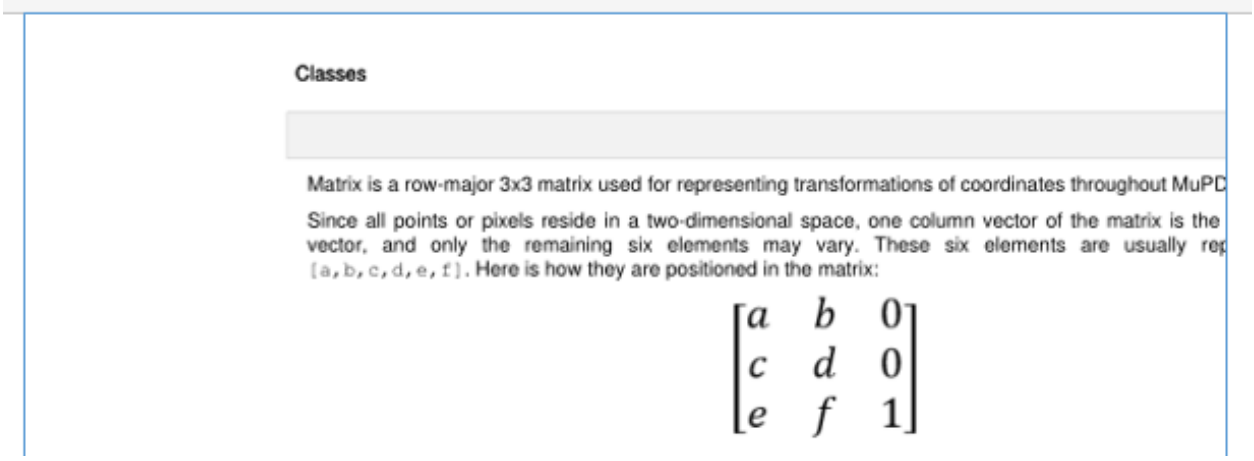
Since all points or pixels reside in a two-dimensional space, one column vector of the matrix is the constant unit vector, and only the remaining six elements may vary. These six elements are usually represented by `[a,b,c,d,e,f]`. Here is how they are positioned in the matrix:

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

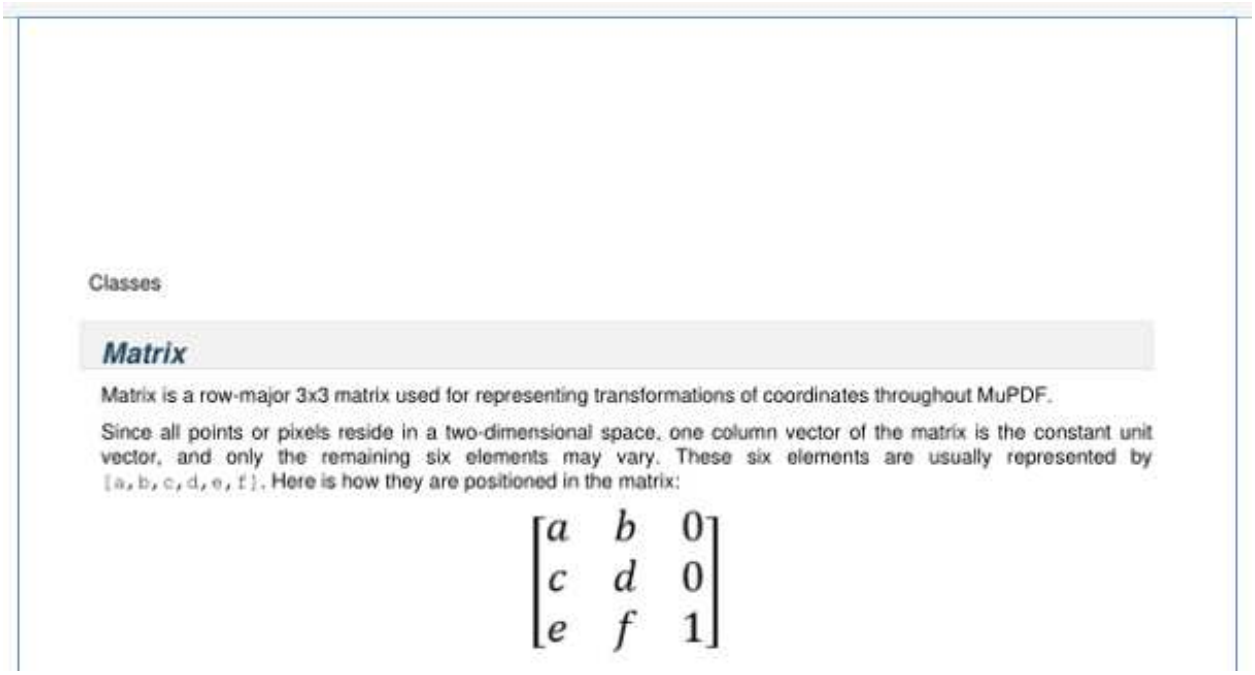
It should be noted, that the below methods are just convenience functions. Each of them manipulates some of the six matrix elements in a specific way. By directly changing `[a,b,c,d,e,f]`, any of these functions can be replaced.

8.8.5 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`.



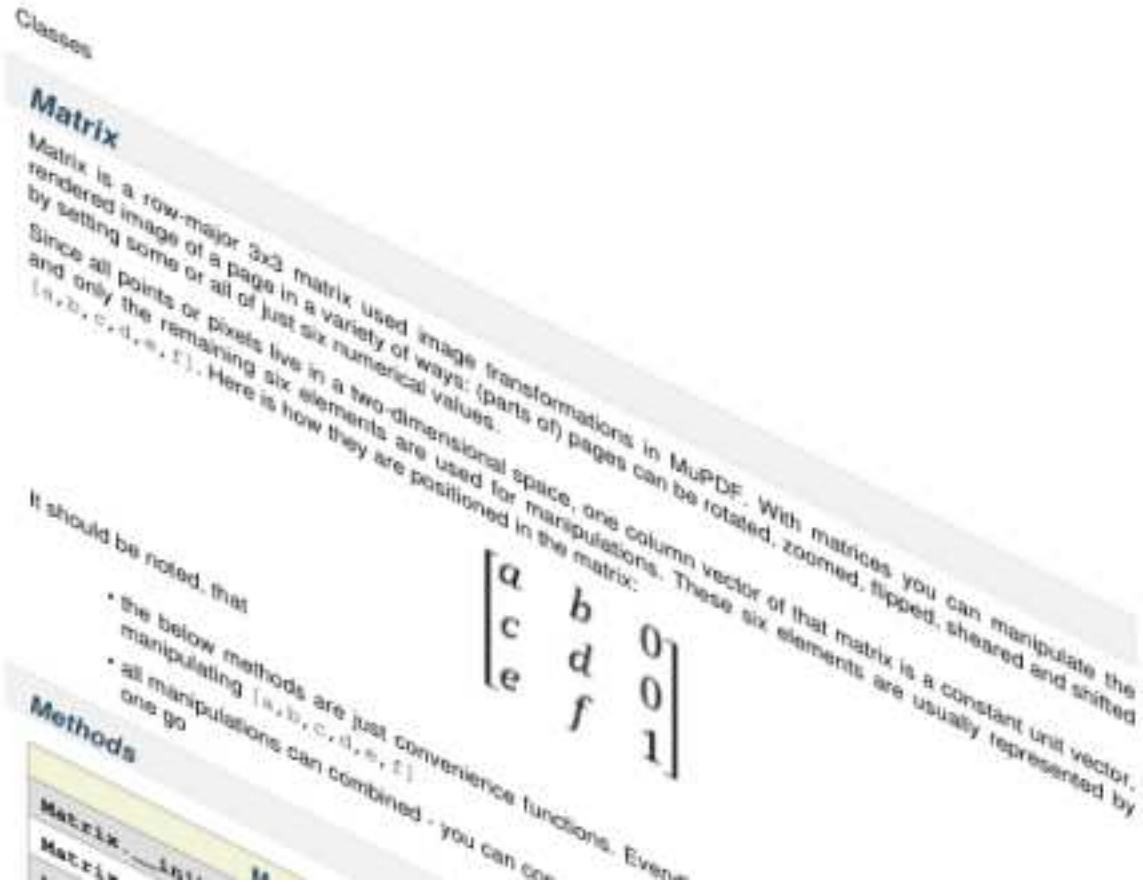
8.8.6 Flipping

Flip the page left-right (`a = -1`).

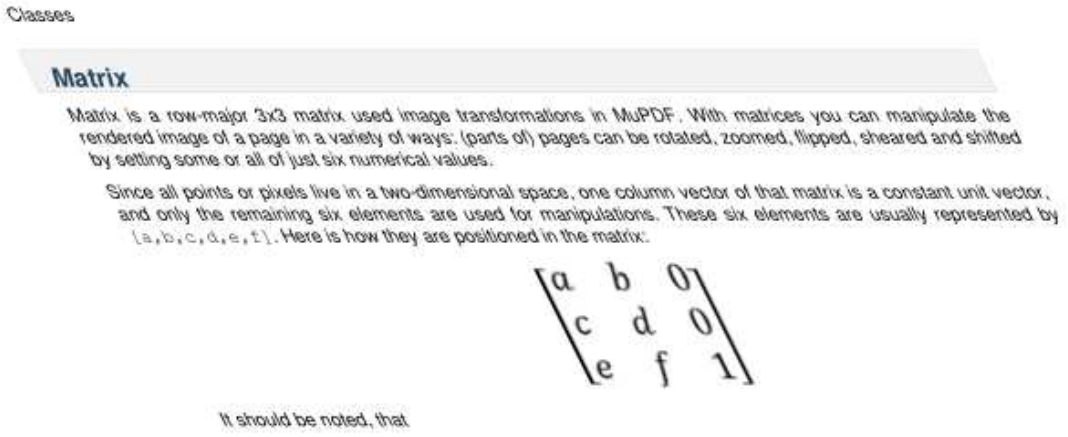
Flip up-down (`d = -1`).

8.8.7 Shearing

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

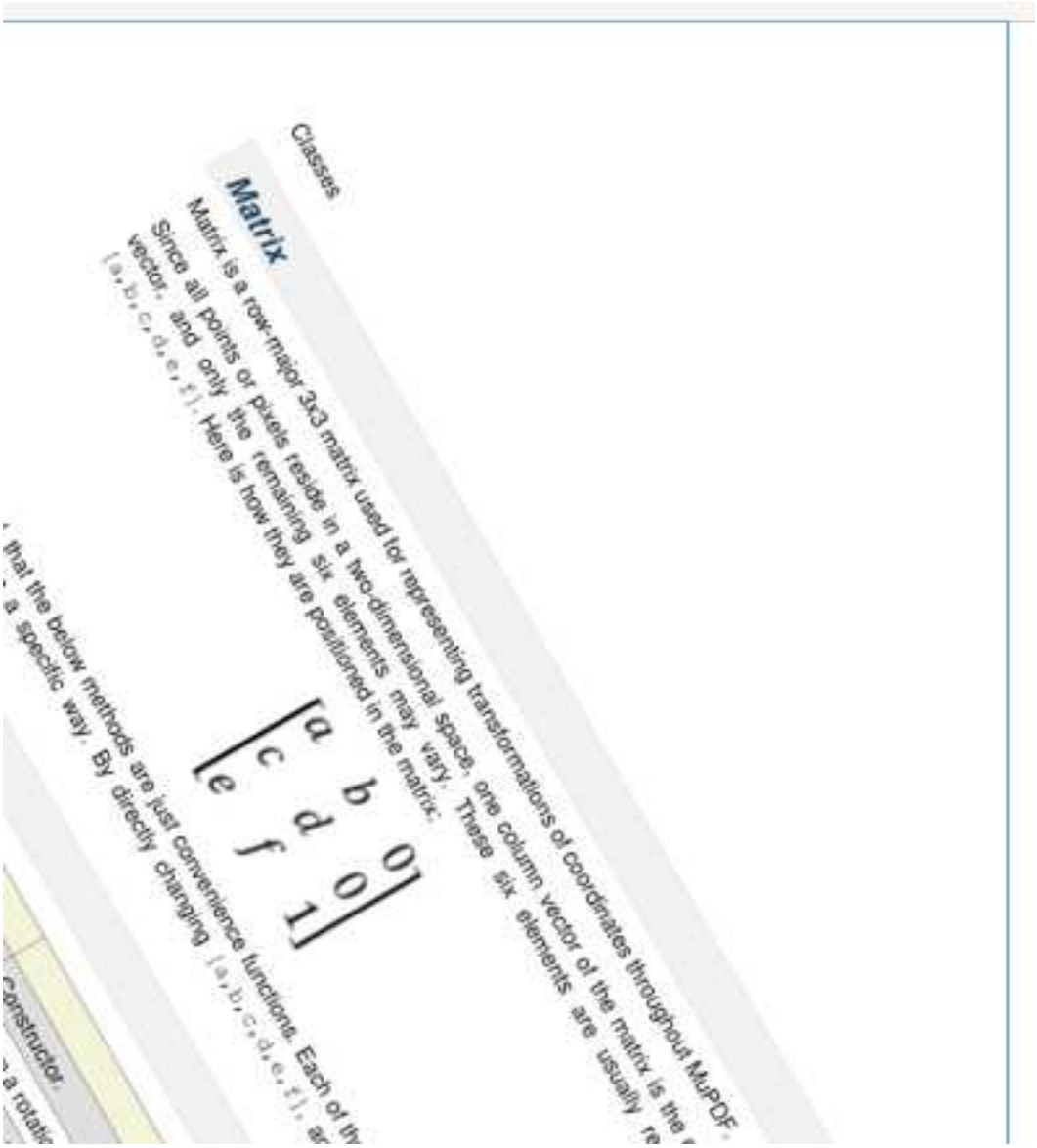


Second a shear in X direction (`c = 0.5`).



8.8.8 Rotating

Finally a rotation by 30 clockwise degrees (`preRotate(-30)`).



8.9 Outline

`outline` (or "bookmark"), 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" or downward direction. The full tree of all outline items for e.g. a conventional table of contents (TOC) can be recovered by following these "pointers".

Method / Attribute	Short Description
Outline.saveText()	prints a conventional TOC to a file
Outline.saveXML()	prints an XML-like TOC to a file
Outline.down	next item downwards
Outline.next	next item same level
Outline.page	page number (0-based)
Outline.title	title
Outline.uri	string further specifying the outline target
Outline.isExternal	target is outside this document
Outline.is_open	whether sub-outlines are open or collapsed
Outline.isOpen	whether sub-outlines are open or collapsed
Outline.dest	points to link destination details

Class API

class Outline

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

Return type *Outline*

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

Return type *Outline*

page
The page number (0-based) this bookmark points to.

Return type int

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

Return type string

is_open
Or isOpen - an indicator showing whether any sub-outlines should be expanded (True) or be collapsed (False). This information should be interpreted by PDF display software accordingly.

Return type bool

saveText (*filename*)
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 hierarchy level of the entry. Page numbers are 1-based in this case. page = -1 can occur if the destination is outside this document or undefined (uri == None).

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

saveXML (*filename*)
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.

isExternal
A bool specifying whether the target is outside (`True`) of the current document.

Return type bool

uri
A string specifying the link target. The meaning of this property should be evaluated in conjunction with `isExternal`. The value may be `None`, in which case `isExternal == False`. If `uri` starts with `file://`, `mailto:`, or an internet resource name, `isExternal` is `True`. In all other cases `isExternal == False` and `uri` points to an internal location. In case of PDF documents, this should either be `#nnnn` to indicate a 1-based (!) page number `nnnn`, or a named location. The format varies for other document types, e.g. `uri = '../FixedDoc.fdoc#PG_21_LNK_84'` for page number 21 (1-based) in an XPS document.

Return type str

dest
The link destination details object.

Return type *linkDest*

8.10 Page

Class representing one document page. A page object is created by *Document.loadPage()* (or equivalently via indexing the document like `doc[n]`).

Method / Attribute	Short Description
Page.bound()	rectangle (mediabox) of the page
Page.deleteAnnot()	PDF only: delete an annotation
Page.getLinks()	get all links
Page.deleteLink()	PDF only: delete a link
Page.insertLink()	PDF only: insert a new link
Page.updateLink()	PDF only: modify a link
Page.getPixmap()	create a *Pixmap*
Page.getText()	extract the text
Page.run()	run a page through a device
Page.searchFor()	search for a string
Page.setRotation()	PDF only: set page rotation
Page.firstAnnot	first *Annot* on the page
Page.firstLink	first *Link* on the page
Page.number	page number
Page.parent	the owning document object
Page.rect	rectangle (mediabox) of the page
Page.rotation	PDF only: page rotation

Class API

class Page

bound()
Determine the rectangle ("mediabox", before transformation) of the page.

Return type *Rect*

rect
Contains the rectangle ("mediabox", before transformation) of the page. Same as result of method `bound()`.

Return type *Rect*

deleteAnnot(*annot*)
PDF only: Delete the specified annotation from the page and (for all document types) return the next one.

Parameters **annot** (*Annot*) – the annotation to be deleted.

Return type *Annot*

Returns the next annotation of the deleted one.

deleteLink(*linkdict*)
PDF only: Delete the specified link from the page. The parameter must be a dictionary of format as provided by the `getLinks()` method (see below).

Parameters **linkdict** (*dict*) – the link to be deleted.

insertLink(*linkdict*)
PDF only: Insert a new link on this page. The parameter must be a dictionary of format as provided by the `getLinks()` method (see below).

Parameters **linkdict** (*dict*) – the link to be inserted.

updateLink(*linkdict*)
PDF only: Modify the specified link. The parameter must be a dictionary of format as provided by the `getLinks()` method (see below).

Parameters **linkdict** (*dict*) – the link to be modified.

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

Return type list

Returns A list of dictionaries. The entries are in the order as specified during PDF generation. For a description of the dictionary entries see below. Always use this method if you intend to make changes to the links of a page.

getText(*output = 'text'*)
Retrieves the text of a page. Depending on the output parameter, the results of the *TextPage* extract methods are returned.

If `output = 'text'` is specified, plain 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 `output = 'json'` (`= TextPage.extractJSON()`) and re-arranges text according to the Western reading layout convention "from top-left to bottom-right".

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"*, *clip = None*, *alpha = False*)
Creates a Pixmap from the page.

Parameters

- **matrix** (*Matrix*) – A *Matrix* object. Default is the *Identity* matrix.
- **colorspace** (*string*) – Defines the required colorspace, one of `GRAY`, `CMYK` or `RGB` (default).
- **clip** (*IRect*) – An `Irect` to restrict rendering of the page to the rectangle's area. If not specified, the complete page will be rendered.
- **alpha** (*bool*) – An bool indicating whether an alpha channel should be included in the pixmap. Leave it as `False` if you do not absolutely need transparency. This will save a lot of memory (25% in case of RGB).

Return type *Pixmap*

Returns Pixmap of the page.

setRotation(*rot*)
PDF only: Sets the rotation of the page.

Parameters **rot** (*int*) – An integer specifying the required rotation in degrees. Should be a (positive or negative) multiple of 90.

Returns zero if successfull, `-1` if not a PDF.

searchFor(*text*, *hit_max = 16*)
Searches for `text` on a page. Identical to *TextPage.search()*.

Parameters

- **text** (*string*) – Text to searched for. Upper / lower case is ignored.
- **hit_max** (*int*) – Maximum number of occurrences accepted.

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. Set it to *Identity* if no transformation is desired.

rotation
PDF only: contains the rotation of the page in degrees and `-1` for other document types.

Return type int

firstLink
Contains the first *Link* of a page (or `None`).

Return type *Link*

firstAnnot
Contains the first *Annot* of a page (or `None`).

Return type *Annot*

number
The page number.

Return type int

parent
The owning document object.

Return type *Document*

8.10.1 Description of `getLinks()` Entries

Each entry of the `getLinks()` list is a dictionay with the following keys:

- **kind:** (required) an integer indicating the type of link. This is one of `fitz.LINK_NONE`, `fitz.LINK_GOTO`, `fitz.LINK_GOTOR`, `fitz.LINK_LAUNCH`, `fitz.LINK_NAME`, or `fitz.LINK_URI`.
- **from:** (required) a `fitz.Rect` describing the "hot area" location on the page's visible representation (where the cursor changes to a hand image, usually).
- **page:** a 1-based integer indicating the destination page. Required for `fitz.LINK_GOTO` and `fitz.LINK_GOTOR`, ignored otherwise.
- **to:** (optional) a `fitz.Point` specifying the destination location on the provided page. Default is `fitz.Point(0, 0)`. Meaningful only for `fitz.LINK_GOTO` and `fitz.LINK_GOTOR`, ignored otherwise.
- **file:** a string specifying the destination file. Required and meaningful only for `fitz.LINK_GOTOR` and `fitz.LINK_LAUNCH` (otherwise ignored).
- **uri:** a string specifying the destination internet resource. Required and meaningful only for `fitz.LINK_URI` (otherwise ignored).
- **name:** a string specifying a named destination of the current PDF. Required and meaningful only for `fitz.LINK_NAME` (otherwise ignored). Each PDF has the following 4 predefined names: "LastPage", "FirstPage", "PrevPage" and "NextPage".
- **xref:** an integer specifying the PDF cross reference entry of the link object. Do not change this entry in any way. Required for link deletion and update, otherwise ignored.
- **id:** an integer specifying the Python object id of the link (function `id()`). Do not change this entry in any way. Required for link deletions only, ignored otherwise.
- **type:** a string representation of link kind. It is always optional and only exists for documentation and debugging purposes.

8.11 Pixmap

Pixmaps ("pixel maps") are objects at the heart of MuPDF's rendering capabilities. They represent plane rectangular sets of pixels. Each pixel is described by a number of bytes ("components") plus an (optional since v1.10.0) alpha byte.

In PyMuPDF, there exist several ways to create a pixmap. Except one, all of them are available as overloaded constructors. A pixmap can be created ...

1. from a document page (via methods *Page.getPixmap()* or *Document.getPagePixmap()*)
2. empty based on *Colorspace* and *IRect* information
3. from an image file
4. from an in-memory image (bytearray)
5. from a memory area of plain pixels
6. from an image inside a PDF document
7. as a copy of another pixmap

Note: A number of image formats are supported as input using the **file** or **in-memory constructors**. For a list see section below.

Have a look at the **example** section to see some pixmap usage "at work".

Method / Attribute	Short Description
Pixmap.clearWith()	clears (parts of) a pixmap
Pixmap.copyPixmap()	copy parts of another pixmap
Pixmap.gammaWith()	applies a gamma factor to the pixmap
Pixmap.getPNGData()	returns a PNG as a memory area
Pixmap.invertIRect()	invert the pixels of a given area
Pixmap.tintWith()	tints a pixmap with a color
Pixmap.writeImage()	saves a pixmap in a variety of image formats
Pixmap.writePNG()	saves a pixmap as a PNG file
Pixmap.alpha	indicates whether transparency is included
Pixmap.colorspace	contains the *Colorspace*
Pixmap.height	height of the region in pixels
Pixmap.interpolate	interpolation method indicator
Pixmap.irect	is the *IRect* of the pixmap
Pixmap.n	number of bytes per pixel including alpha byte
Pixmap.samples	the components data for all pixels
Pixmap.size	contains the pixmap's total length
Pixmap.stride	number of bytes of one image row
Pixmap.width	width of the region in pixels
Pixmap.x	X-coordinate of top-left corner of pixmap
Pixmap.xres	resolution in X-direction
Pixmap.y	Y-coordinate of top-left corner of pixmap
Pixmap.yres	resolution in Y-direction

Class API

class `Pixmap`

`__init__`(*self*, *colorspace*, *irect*, *alpha*)

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)` designates the top left corner of the pixmap. Note that the image area is **not initialized** and will contain crap data.

Parameters

- **`colorspace`** (*Colorspace*) – The colorspace of the pixmap.
- **`irect`** (*IRect*) – Specifies the pixmap's area and its location.
- **`alpha`** (*bool*) – Specifies whether transparency bytes should be included. Default is `False`.

`__init__`(*self*, *doc*, *xref*)

This constructor creates a pixmap with origin `(0, 0)` from an image contained in PDF document `doc` identified by its XREF number.

Parameters

- **doc** (*Document*) – an opened **PDF** document.
- **xref** (*int*) – the XREF number of the image.

__init__(*self*, *colorspace*, *sourcepix*)

This constructor creates a new pixmap as a copy of another one, `sourcepix`. If the two colorspaces differ, a conversion will take place. Any combination of supported colorspaces is possible. The result will have the same alpha as the source.

Parameters

- **colorspace** (*Colorspace*) – The colorspace of the pixmap.
- **sourcepix** (`Pixmap`) – the source pixmap.

__init__(*self*, *filename*)

This constructor creates a pixmap from the image contained in file `filename`. The image type is determined automatically.

Parameters **filename** (*string*) – Path / name of the file. The origin of the resulting pixmap is `(0, 0)`.

__init__(*self*, *img*)

This constructor creates a non-empty pixmap from `img`, which is assumed to contain a supported image as a bytearray. The image type is determined automatically.

Parameters **img** (*bytearray*) – Data containing a complete, valid image in one of the supported formats. E.g. this may have been obtained from a statement like `img = bytearray(open('somepic.png', 'rb').read())`. The origin of the resulting pixmap is (0,0).

__init__(*self*, *colorspace*, *width*, *height*, *samples*, *alpha*)

This constructor creates a non-empty pixmap from `samples`, which is assumed to contain an image in "plain pixel" format. This means that each pixel is represented by `n` bytes (as controlled by the `colorspace` parameter). The origin of the resulting pixmap is (0,0). This method is useful when raw image data are provided by some other program - see examples below.

Parameters

- **colorspace** (*Colorspace*) – Colorspace of the image. Together with `alpha` this parameter controls the interpretation of the `samples` area: for *CS_GRAY*, *CS_RGB* and *CS_CMYK*, `1 + alpha`, `3 + alpha` or `4 + alpha` bytes in samples will be assumed to define one pixel, respectively. Calling this number `n`, the following must evaluate to `True`: `n * width * height == len(samples)`.
- **width** (*int*) – Width of the image in pixels
- **height** (*int*) – Height of the image in pixels

- **samples** (*bytearray, bytes or str*) – `bytearray`, `bytes` or `string` (Python 2 only) containing consecutive bytes describing all pixels of the image.
- **alpha** (*bool*) – a transparency channel is included in samples.

clearWith(*value*[, *irect*])

Clears an area specified by the *IRect* `irect` within a pixmap. To clear the whole pixmap omit `irect`.

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) if present. Default is 0 (black).
- **irect** (*IRect*) – An `IRect` object specifying the area to be cleared.

tintWith(*red*, *green*, *blue*)

Colorizes (tints) a pixmap with a color provided as a value triple (red, green, blue). Use this method only for *CS_GRAY* or *CS_RGB* colorspaces. A `TypeError` exception will otherwise be raised.

If the colorspace is *CS_GRAY*, `(red + green + blue)/3` will be taken as the tinting value.

Parameters

- **red** (*int*) – The `red` component. Values from 0 to 255 are valid.
- **green** (*int*) – The `green` component. Values from 0 to 255 are valid.
- **blue** (*int*) – The `blue` component. Values from 0 to 255 are valid.

gammaWith(*gamma*)

Applies a gamma factor to a pixmap, i.e. lightens or darkens it.

Parameters **gamma** (*float*) – `gamma = 1.0` does nothing, `gamma < 1.0` lightens, `gamma > 1.0` darkens the image.

invertIRect(*irect*)

Invert the color of all pixels in an area specified by *IRect* `irect`. To invert everything, use `getIRect()` or omit this parameter.

Parameters **irect** (*IRect*) – The area to be inverted.

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 (provided each is either *CS_GRAY* or *CS_RGB*), but currently **must** have the same alpha property. The copy mechanism automatically adjusts discrepancies between source and target pixmap like so:

If copying from *CS_GRAY* to *CS_RGB*, the source gray-shade value will be put into each of the three rgb component bytes. If the other way round, (r + g + b) / 3 will be taken as the gray-shade value of the target.

Between the specified `irect` and the target pixmap's *IRect*, an "intersection" rectangle is calculated at first. Then the corresponding data of this intersection are being copied. If the intersection is empty, nothing will happen.

If you want your `source` pixmap image to land at a specific position of the target, set its `x` and `y` attributes to the top left point of the desired rectangle before copying. See the example below for how this works.

Parameters

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

writePNG(*filename*)

Saves a pixmap as a PNG file. Please note that only grayscale and RGB colorspaces can be saved in PNG format (this is a MuPDF restriction). CMYK colorspaces must either be saved as `*.pam` files or be converted. Since MuPDF v1.10a the `savealpha` option is no longer supported and will be ignored with a warning.

Parameters **filename** (*string*) – The filename to save as (the extension `png` must be specified).

getPNGData()

Returns the pixmap as an image area (bytearray) in PNG format. Please note that only grayscale and RGB colorspaces can be produced in PNG format (this is a MuPDF restriction). CMYK colorspaces must be converted first. Since MuPDF v1.10a the `savealpha` option is no longer supported and will be ignored with a warning.

Return type bytearray

writeImage(*filename*, *output="png"*)

Saves a pixmap as an image file. This method is an extension to `writePNG()`. Depending on the output chosen, some or all colorspaces are supported and different file extensions can be chosen. Please see the table below. Since MuPDF v1.10a the `savealpha` option is no longer supported and will be ignored with a warning.

Parameters

- **filename** (*string*) – The filename to save to. Depending on the chosen output format, possible file extensions are `.pam`, `.pbm`, `.pgm`, `ppm`, `.pnm`, `.png` and `.tga`.
- **output** (*string*) – The requested image format. The default is `png` for which this function is equivalent to `writePNG()`. Other possible values are `pam`, `pnm` and `tga`.

alpha

Indicates whether this pixmap contains transparency information

Return type bool

colorspace

The colorspace of the pixmap.

Return type str

stride
Contains the length of one row of image data in `samples`. This is primarily used for calculation purposes. The following expressions are `True`: `len(samples) == height * stride`, `width * n == stride`, `Colorspace.nbytes + alpha == n`.

Return type int

irect
Contains the *IRect* of the pixmap.

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 this means, `samples` is a bytearray like `..., R, G, B, A, ...`, and the four byte values R, G, B, A define one pixel.

This area can also be used by other graphics libraries like PIL (Python Imaging Library) to do additional processing like saving the pixmap in other image formats. See example 3.

Return type bytearray

size
Contains the total length of the pixmap. This will generally equal `len(pix.samples) + 60`. The following will evaluate to `True`: `len(pixmap) == pixmap.size`.

Return type int

width
The width of the region in pixels. For compatibility reasons, `w` is also supported.

Return type int

height
The height of the region in pixels. For compatibility reasons, `h` is also supported.

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 colorspace and alpha (see remark above). `Pixmap.n - Pixmap.alpha == Pixmap.colorspace.n` is always `True`.

Return type int

xres
Horizontal resolution in dpi (dots per inch).

Return type int

yres
Vertical resolution in dpi.

Return type int

interpolate
An information-only boolean flag set to `True` if the image will be drawn using "linear interpolation". If `False` "nearest neighbour sampling" will be used.

Return type bool

8.11.1 Supported Pixmap Construction Image Types

The following file types are supported as input to construct pixmaps: BMP, JPEG, GIF, TIFF, JXR, and PNG.

8.11.2 Details on Saving Images with `writeImage()`

The following table shows possible combinations of file extensions, output formats and colorspaces of method `writeImage()`:

output =	CS_GRAY	CS_RGB	CS_CMYK
"pam"	.pam	.pam	.pam
"pnm"	.pnm, .pgm	.pnm, .ppm	invalid
"png"	.png	.png	invalid
"tga"	.tga	.tga	invalid

Note: Not all image file types are available, or at least common on all platforms, e.g. PAM is mostly unknown on Windows. Especially pertaining to CMYK colorspaces, you can always convert a CMYK pixmap to an RGB-pixmap with `rgb_pix = fitz.Pixmap(fitz.csRGB, cmyk_pix)` and then save that as a PNG.

8.11.3 Pixmap Example Code Snippets

Example 1

This shows how pixmaps can be used for purely graphical, non-PDF purposes. The script reads a PNG picture and creates a new PNG file which consist of 3 * 4 tiles of the original one:

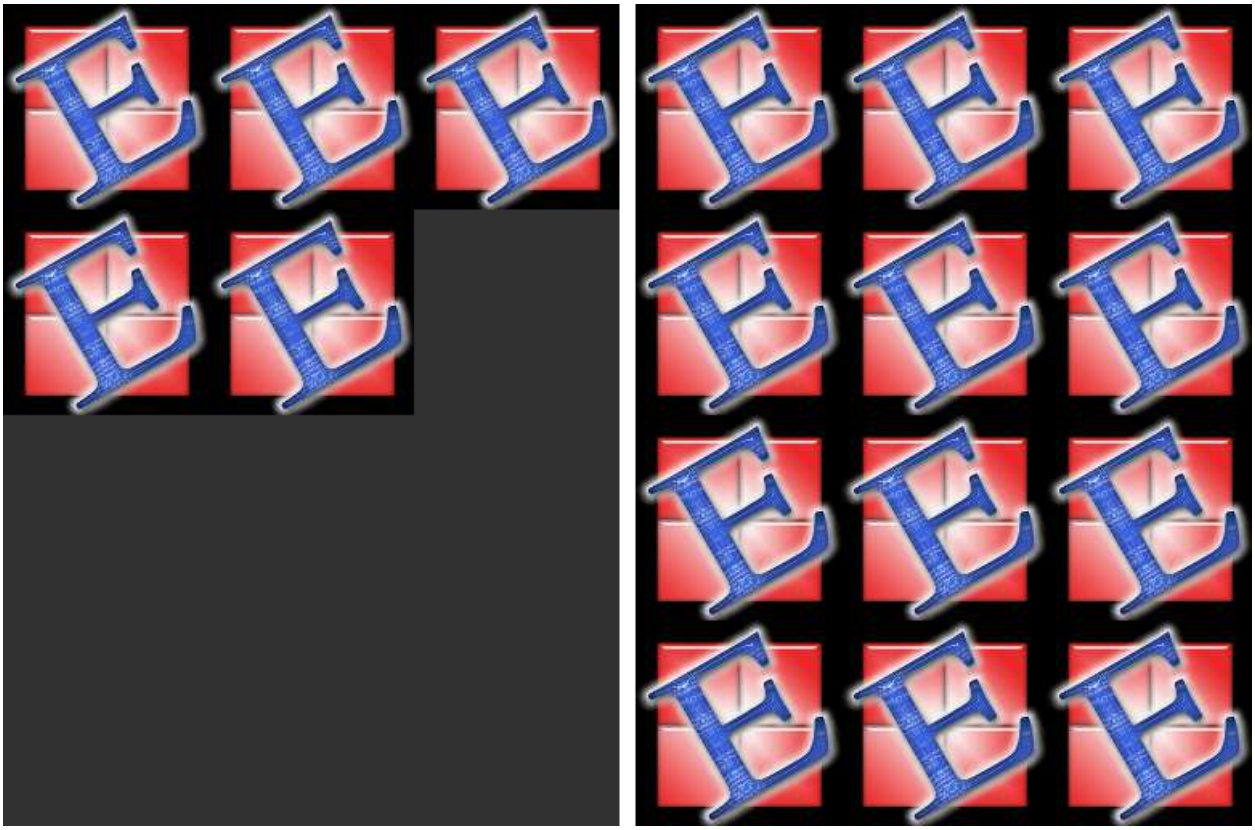
```
import fitz
# create a pixmap of a picture
pix0 = fitz.Pixmap("editra.png")

# set target colorspace and pixmap dimensions and create it
tar_width  = pix0.width * 3              # 3 tiles per row
tar_height = pix0.height * 4             # 4 tiles per column
tar_irect  = fitz.IRect(0, 0, tar_width, tar_height)
# create empty target pixmap
tar_pix    = fitz.Pixmap(fitz.csRGB, tar_irect, pix0.alpha)
# clear target with a very lively stone-gray (thanks and R.I.P., Loriot)
tar_pix.clearWith(90)

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

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

Here is the output, showing some intermediate picture and the final result:



Example 2

This shows how to create a PNG file from a numpy array (several times faster than most other methods):

```
import numpy as np
import fitz

#============================================================================
# create a fun-colored width * height PNG with fitz and numpy

#============================================================================
height = 150
width  = 100
bild=np.ndarray((height, width, 3), dtype=np.uint8)

for i in range(height):
    for j in range(width):
        # one pixel (some fun coloring)
        bild[i, j] = [(i+j)%256, i%256, j%256]

samples = bytearray(bild.tostring())     # get plain pixel data from numpy array
pix=fitz.Pixmap(fitz.csRGB, width, height, samples, alpha=False)
pix.writePNG("test.png")
```

Example 3

This shows how to interface with `PIL / Pillow` (the Python Imaging Library), thereby extending the

reach of image files that can be processed:

```
import fitz
from PIL import Image

pix = fitz.Pixmap(...)
... # any code here
# create and save a PIL image
img = Image.frombytes("RGB", [pix.width, pix.height], str(pix.samples))
img.save(filename, 'jpeg')

# an example for the opposite direction
# create a pixmap from any PIL-supported image file "some_image.xxx"

img = Image.open("some_image.xxx").convert("RGB")
samples = bytearray(img.tobytes())
pix = fitz.Pixmap(fitz.csRGB, img.size[0], img.size[1], samples, alpha=False)
```

8.12 Point

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

Attribute / Method	Short Description
`Point.transform()`	transform point with a matrix
`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

`__init__` (*self*, *p*)

Constructor. Makes a **new copy** of point `p`.

Parameters **`p`** (*Point*) – The point to copy from.

`transform`(*m*)

Applies matrix `m` to the point.

Parameters `m` (*Matrix*) – The matrix to be applied.

8.12.1 Remark

A point's `p` attributes `x` and `y` can also be accessed as indices, e.g. `p.x == p[0]`.

8.12.2 Point Algebra

A number of arithmetics operations have been defined for the `Point` class:

- **Addition:** `p + x` is a new `Point` with added coordinates of `p` and `x` (another `Point` or a number). If `x` is a number, it is added to both components of `p`.
- **Subtraction:** analogous to addition.
- **Negation:** `-p` is the point with negated coordinates of `p`.
- **Multiplication:** `p * m` means `p.transform(m)` for matrix `m`, however p is left untouched and a new point is returned.
- **Absolute Value:** `abs(p)` means the Euclidean norm of `p`, i.e. its length as a vector.

8.12.3 Examples

Example 1:

```
>>> point = fitz.Point(25, 30)
>>> point
fitz.Point(25.0, 30.0)
>>> m = fitz.Matrix(2, 2)
>>> point.transform(m)
fitz.Point(50.0, 60.0)
```

Example 2:

```
>>> fitz.Point(25, 30) + 5
fitz.Point(30.0, 35.0)
>>>
>>> fitz.Point(25, 30) + fitz.Point(1, 2)
fitz.Point(26.0, 32.0)
>>>
>>> abs(fitz.Point(25, 30))
39.05124837953327
```

8.13 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 four ways: as a pair of *Point* objects, as a tuple of four coordinates, or as an arbitrary combination of these.

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*.

A rectangle is called *empty* if x0 = x1 or y0 = y1, i.e. if its area is zero.

Methods / Attributes	Short Description
Rect.round()	create smallest *IRect* containing rectangle
Rect.transform()	transform rectangle with a matrix
Rect.intersect()	common part with another rectangle
Rect.includePoint()	enlarge rectangle to also contain a point
Rect.includeRect()	enlarge rectangle to also contain another one
Rect.getRectArea()	calculate rectangle area
Rect.height	rectangle height
Rect.irect	equals result of method round()
Rect.width	rectangle 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*)

__init__(*self*, *p1*, *x1*, *y1*)

__init__(*self*, *x0*, *y0*, *p2*)

__init__(*self*, *r*)
Overloaded constructors: p1, p2 stand for *Point* objects, r means another rectangle, while the other parameters mean float coordinates.

If r is specified, the constructor creates a **new copy** of r.

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

Return type *IRect*

transform(*m*)
Transforms rectangle with a matrix.

Parameters **m** (*Matrix*) – The matrix to be used for the transformation.

intersect(*r*)
The intersection (common rectangular area) of the current rectangle and r is calculated and replaces the current rectangle. If either rectangle is empty, the result is also empty. If one of

the rectangles is infinite, the other one is taken as the result - and hence also infinite if both rectangles were infinite.

Parameters **r** (*Rect*) – Second rectangle

includeRect(*r*)
The smallest rectangle containing the current one and `r` is calculated and replaces the current one. If either rectangle is infinite, the result is also infinite. If one is empty, the other will be taken as the result (which will be empty if both were empty).

Parameters **r** (*Rect*) – Second rectangle

includePoint(*p*)
The smallest rectangle containing the current one and point `p` is calculated and replaces the current one. To create a rectangle to contain a series of points, start with the empty `fitz.Rect(p1, p1)` and successively perform `includePoint` operations for the other points.

Parameters **p** (*Point*) – Point to include.

getRectArea(*unit = 'pt'*)
Calculates the area of the rectangle. The area of an infinite rectangle is always zero. So, at least one of `fitz.Rect(p1, p2)` and `fitz.Rect(p2, p1)` has a zero area.

Parameters **unit** (*string*) – Specify required unit: `pt` (pixel points, default) or `mm` (square millimeters).

Return type float

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

8.13.1 Remark

A rectangle's coordinates can also be accessed via index, e.g. `r.x0 == r[0]`.

8.13.2 Rect Algebra

A number of arithmetics operations have been defined for the `Rect` class.

- **Addition:** `r + x` where `r` is a `Rect` and `x` can be a `Rect`, `IRect` or a number. The result is a new `Rect` with added components of the operands. If `x` is a number, it is added to all components of `r`.
- **Subtraction:** analogous to addition.
- **Negation:** `-r` is a new `Rect` with negated components of `r`.
- **Inclusion:** `r | x` is the new `Rect` that also includes `x`, which can be an `IRect`, `Rect` or `Point`.
- **Intersection:** `r & x` is a new `Rect` containing the area common to `r` and `x` which can be an `IRect` or `Rect`.
- **Multiplication:** `r * m` is a new `Rect` containing `r` transformed with matrix `m`.

8.13.3 Examples

Example 1:

```
>>> p1 = fitz.Point(10, 10)
>>> p2 = fitz.Point(300, 450)
>>>
>>> fitz.Rect(p1, p2)
fitz.Rect(10.0, 10.0, 300.0, 450.0)
>>>
>>> fitz.Rect(10, 10, 300, 450)
fitz.Rect(10.0, 10.0, 300.0, 450.0)
>>>
>>> fitz.Rect(10, 10, p2)
fitz.Rect(10.0, 10.0, 300.0, 450.0)
>>>
>>> fitz.Rect(p1, 300, 450)
fitz.Rect(10.0, 10.0, 300.0, 450.0)
```

Example 2:

```
>>> 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)
```

Example 3:

```
>>> m = fitz.Matrix(45)
>>> r = fitz.Rect(10, 10, 410, 610)
>>> r * m
fitz.Rect(-424.2640686035156, 14.142135620117188, 282.84271240234375, 721.
↪2489013671875)
>>>
>>> r | fitz.Point(5, 5)
fitz.Rect(5.0, 5.0, 410.0, 610.0)
>>>
>>> r + 5
fitz.Rect(15.0, 15.0, 415.0, 615.0)
>>>
>>> r & fitz.Rect(0, 0, 15, 15)
fitz.Rect(10.0, 10.0, 15.0, 15.0)
```

As can be seen, all of the following evaluate to `True`:

- `r.round().x0 == int(math.floor(r.x0))`
- `r.round().y0 == int(math.floor(r.y0))`
- `r.round().x1 == int(math.ceil(r.x1))`
- `r.round().y1 == int(math.ceil(r.y1))`.

CHAPTER

NINE

LOW LEVEL FUNCTIONS AND CLASSES

Contains a number of functions and classes for the experienced user. To be used for special requirements or heavy performance requirements.

9.1 Functions

The following are miscellaneous functions either directly available under the binding name, i.e. can be invoked as `fitz.function`, or to be used by the experienced PDF programmer.

Function	Short Description
getPointDistance()	calculates the distance between two points
Document._delXmlMetadata()	deletes anly metadata object in XML format
Document._getPageObjNumber()	returns a pages XREF and generation number
Document._getPageXref()	synonym of `_getPageObjNumber()`
Document._getObjectString()	returns the string representing an object
Document._getNewXref()	creates and returns a new entry in the XREF
Document._updateObject()	inserts or updates a PDF object
Document._getXrefLength()	returns the length of the PDF XREF
Document._getXrefStream()	returns the the content of the stream
Document._getOLRootNumber()	returns / creates the outline root XREF

getPointDistance(*p1*, *p2*, *unit = "pt"*)

Calculates the distance between two points in either pixel points "pt" (default) or millimeters "mm". `fitz.getPointDistance(p1, p2) == fitz.getPointDistance(p2, p1)` always evaluates to `True`.

Parameters

- **p1** (*Point*) – First point
- **p2** (*Point*) – Second point
- **unit** (*str*) – Unit specification, "pt" or "mm"

Return type float

Document.**_delXmlMetadata**()

PDF documents only: Deletes any reference to an object containing XML-based metadata from the PDF root object. (Py-) MuPDF does not support XML-based metadata. Use this

if you want to make sure that the conventional metadata dictionary will be used exclusively. Many thirdparty PDF programs insert their own metadata in XML format and thus may override what you store in the conventional dictionary. This deletes any such reference, and the corresponding PDF object will be deleted during next garbage collection of the file.

Document.**_getPageObjNumber**(*pno*)
PDF documents only: Returns the XREF and generation number for a given page.

Parameters **pno** (*int.*) – Page number (zero-baed).

Return type list

Returns XREF and generation number of page pno as a list [xref, gen].

Document.**_getPageXref**(*pno*)
PDF documents only: Synonym for _getPageObjNumber().

Document.**_getObjectString**(*xref*)
PDF documents only: Returns the string representing an arbitrary object. For stream objects, only the non-stream part is returned. To get the stream content, use _getXrefStream() (see below).

Parameters **xref** (*int.*) – XREF number.

Return type string

Returns the string defining the object identified by xref.

Document.**_getNewXref**()
PDF documents only: Increases the XREF by one entry and returns the entry's number.

Return type int

Returns the number of the new XREF entry.

Document.**_updateObject**(*xref*, *obj_str*)
PDF documents only: Associates the object identified by string obj_str with the XREF number xref. If xref already pointed to an object, it will be replaced by the new object.

Parameters

- **xref** (*int.*) – XREF number.
- **obj_str** (*str.*) – a string containing a valid PDF object definition.

Return type int

Returns zero if successful, otherwise an exception will be raised.

Document.**_getXrefLength**()
PDF documents only: Returns the length of the XREF table.

Return type int

Returns the number of entries in the XREF table.

Document.**_getXrefStream**(*xref*)

PDF documents only: Returns the content stream of the object referenced by `xref`. If the object has / is no stream, an exception is raised.

Parameters **xref** (*int.*) – XREF number.

Return type str or bytes

Returns the (decompressed) stream of the object. This is a string in Python 2 and a `bytes` object in Python 3.

Document.**_getOLRootNumber**()

PDF documents only: Returns the XREF number of the /Outlines root object (this is **not** the first outline entry!). If this object does not exist, a new one will be created.

Return type int

Returns XREF number of the **/Outlines** root object.

9.2 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*, *clip*)

Constructor for either a pixel map or a display list device.

Parameters

- **object** (*Pixmap* or *DisplayList*) – one of `Pixmap` or `DisplayList`
- **clip** (*IRect*) – An optional *IRect* for `Pixmap` devices only to restrict rendering to a certain area of the page. If the complete page is required, specify `None`. For display list devices, this parameter must be omitted.

__init__(*self*, *textsheet*, *textpage*)

Constructor for a text page device.

Parameters

- **textsheet** (*TextSheet*) – `TextSheet` object
- **textpage** (*TextPage*) – `TextPage` object

9.3 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`

`__init__` (*self*, *mediabox*)
Create a new 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.

Parameters **`mediabox`** (*Rect*) – The page's rectangle - output of `page.bound()`.

Return type `DisplayList`

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

Parameters

- **`dev`** (*Device*) – Device
- **`ctm`** (*Matrix*) – Transformation 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 apply for tile objects contained in the display list.

9.4 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.

Note: All of the above can be achieved by using the appropriate *Document.getPageText()*, *Page.getText()* and *Page.searchFor()* methods.

9.5 TextSheet

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

CHAPTER

TEN

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 is accessible as `fitz.variable`.

10.1 Constants

csRGB
Predefined RGB colorspace `fitz.Colorspace(fitz.CS_RGB)`.

Return type *Colorspace*

csGRAY
Predefined GRAY colorspace `fitz.Colorspace(fitz.CS_GRAY)`.

Return type *Colorspace*

csCMYK
Predefined CMYK colorspace `fitz.Colorspace(fitz.CS_CMYK)`.

Return type *Colorspace*

CS_RGB
1 - Type of *Colorspace* is RGBA

Return type int

CS_GRAY
2 - Type of *Colorspace* is GRAY

Return type int

CS_CMYK
3 - Type of *Colorspace* is CMYK

Return type int

VersionBind
'1.10.x' - version of PyMuPDF (these bindings)

Return type string

VersionFitz
'1.10' - version of MuPDF

Return type string

VersionDate
ISO timestamp `YYYY-MM-DD HH:MM:SS` when these bindings were built.

Return type string

Note: The docstring of `fitz` contains information of the above which can be retrieved like so: `print(fitz.__doc__)`, and a possible response should look like: `PyMuPDF 1.10.0: Python bindings for the MuPDF 1.10 library, built on 2016-11-30 13:09:13`.

10.2 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 pp. 581.

LINK_NONE
0 - No destination

Return type int

LINK_GOTO
1 - Points to a place in this document.

Return type int

LINK_URI
2 - Points to a URI - typically an internet resource.

Return type int

LINK_LAUNCH
3 - Launch (open) another document.

Return type int

LINK_NAMED
4 - Perform some action, like "FirstPage", "NextPage", etc.

Return type int

LINK_GOTOR
5 - Points to a place in another document.

Return type int

10.3 Link Destination Flags

Note: The rightmost byte of this integer is a bit field, so test the truth of these bits with the `&` operator.

LINK_FLAG_L_VALID
1 (bit 0) Top left x value is valid

Return type bool

LINK_FLAG_T_VALID
2 (bit 1) Top left y value is valid

Return type bool

LINK_FLAG_R_VALID
4 (bit 2) Bottom right x value is valid

Return type bool

LINK_FLAG_B_VALID
8 (bit 3) Bottom right y value is valid

Return type bool

LINK_FLAG_FIT_H
16 (bit 4) Horizontal fit

Return type bool

LINK_FLAG_FIT_V
32 (bit 5) Vertical fit

Return type bool

LINK_FLAG_R_IS_ZOOM
64 (bit 6) Bottom right x is a zoom figure

Return type bool

10.4 Annotation Types

Possible values (integer) for PDF annotation types. See chapter 8.4.5, pp. 615 of the Adobe manual for more details.

ANNOT_TEXT
0 - Text annotation

ANNOT_LINK
1 - Link annotation

ANNOT_FREETEXT
2 - Free text annotation

ANNOT_LINE
3 - Line annotation

ANNOT_SQUARE
4 - Square annotation

ANNOT_CIRCLE
5 - Circle annotation

ANNOT_POLYGON
6 - Polygon annotation

ANNOT_POLYLINE
7 - PolyLine annotation

ANNOT_HIGHLIGHT
8 - Highlight annotation

ANNOT_UNDERLINE
9 - Underline annotation

ANNOT_SQUIGGLY
10 - Squiggly-underline annotation

ANNOT_STRIKEOUT
11 - Strikeout annotation

ANNOT_STAMP
12 - Rubber stamp annotation

ANNOT_CARET
13 - Caret annotation

ANNOT_INK
14 - Ink annotation

ANNOT_POPUP
15 - Pop-up annotation

ANNOT_FILEATTACHMENT
16 - File attachment annotation

ANNOT_SOUND
17 - Sound annotation

ANNOT_MOVIE
18 - Movie annotation

ANNOT_WIDGET
19 - Widget annotation

ANNOT_SCREEN
20 - Screen annotation

ANNOT_PRINTERMARK
21 - Printer?s mark annotation

ANNOT_TRAPNET
22 - Trap network annotation

ANNOT_WATERMARK
23 - Watermark annotation

ANNOT_3D
24 - 3D annotation

10.5 Annotation Flags

Possible mask values for PDF annotation flags.

Note: Annotation flags is a bit field, so test the truth of its bits with the `&` operator. When changing flags for an annotation, use the `|` operator to combine several values. The following descriptions were extracted from the Adobe manual, pages 608 pp.

ANNOT_XF_Invisible
1 - If set, do not display the annotation if it does not belong to one of the standard annotation types and no annotation handler is available. If clear, display such an unknown annotation using an appearance stream specified by its appearance dictionary, if any.

ANNOT_XF_Hidden
2 - If set, do not display or print the annotation or allow it to interact with the user, regardless of its annotation type or whether an annotation handler is available. In cases where screen space is limited, the ability to hide and show annotations selectively can be used in combination with appearance streams to display auxiliary pop-up information similar in function to online help systems.

ANNOT_XF_Print
4 - If set, print the annotation when the page is printed. If clear, never print the annotation, regardless of whether it is displayed on the screen. This can be useful, for example, for annotations representing interactive pushbuttons, which would serve no meaningful purpose on the printed page.

ANNOT_XF_NoZoom
8 - If set, do not scale the annotation?s appearance to match the magnification of the page. The location of the annotation on the page (defined by the upper-left corner of its annotation rectangle) remains fixed, regardless of the page magnification.

ANNOT_XF_NoRotate
16 - If set, do not rotate the annotation?s appearance to match the rotation of the page. The upper-left corner of the annotation rectangle remains in a fixed location on the page, regardless of the page rotation.

ANNOT_XF_NoView
32 - If set, do not display the annotation on the screen or allow it to interact with the user. The annotation may be printed (depending on the setting of the Print flag) but should be considered hidden for purposes of on-screen display and user interaction.

ANNOT_XF_ReadOnly
64 - If set, do not allow the annotation to interact with the user. The annotation may be displayed or printed (depending on the settings of the NoView and Print flags) but should not respond to mouse clicks or change its appearance in response to mouse motions.

ANNOT_XF_Locked
128 - If set, do not allow the annotation to be deleted or its properties (including position and size) to be modified by the user. However, this flag does not restrict changes to the annotation?s contents, such as the value of a form field.

ANNOT_XF_ToggleNoView
256 - If set, invert the interpretation of the NoView flag for certain events. A typical use is to have an annotation that appears only when a mouse cursor is held over it.

ANNOT_XF_LockedContents
512 - If set, do not allow the contents of the annotation to be modified by the user. This flag does not restrict deletion of the annotation or changes to other annotation properties, such as position and size.

10.6 Annotation Line End Styles

The following descriptions are taken from the Adobe manual TABLE 8.27 on page 630.

ANNOT_LE_None
0 - No line ending.

ANNOT_LE_Square
1 - A square filled with the annotation?s interior color, if any.

ANNOT_LE_Circle
2 - A circle filled with the annotation?s interior color, if any.

ANNOT_LE_Diamond
3 - A diamond shape filled with the annotation?s interior color, if any.

ANNOT_LE_OpenArrow
4 - Two short lines meeting in an acute angle to form an open arrowhead.

ANNOT_LE_ClosedArrow
5 - Two short lines meeting in an acute angle as in the OpenArrow style (see above) and connected by a third line to form a triangular closed arrowhead filled with the annotation?s interior color, if any.

ANNOT_LE_Butt
6 - (PDF 1.5) A short line at the endpoint perpendicular to the line itself.

ANNOT_LE_ROpenArrow
7 - (PDF 1.5) Two short lines in the reverse direction from OpenArrow.

ANNOT_LE_RClosedArrow
8 - (PDF 1.5) A triangular closed arrowhead in the reverse direction from ClosedArrow.

ANNOT_LE_Slash
9 - (PDF 1.6) A short line at the endpoint approximately 30 degrees clockwise from perpendicular to the line itself.

CHAPTER

ELEVEN

APPENDIX 1: PERFORMANCE

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 we at least should provide some quantitative information to justify our bold comments on MuPDF's **top performance**.

Following are three sections that deal with different aspects of performance:

- document parsing
- text extraction
- image rendering

In each section, the same fixed set of PDF files is being processed by a set of tools. The set of tools varies - for reasons we will explain in the section.

Here is the list of files we are using. Each file name is accompanied by further information: **size** in bytes, number of **pages**, number of bookmarks (**toc** entries), number of **links**, **text** size as a percentage of file size, **KB** per page, PDF **version** and remarks. **text %** and **KB index** are indicators for whether a file is text or graphics oriented.

name	size	pages	toc size	links	text %	KB index	version	remarks
Adobe.pdf	32.472.771	1.310	794	32.096	8,0%	24	PDF 1.6	linearized, text oriented, **many** links / bookmarks
Evolution.pdf	13.497.490	75	15	118	1,1%	176	PDF 1.4	graphics oriented
PyMuPDF.pdf	479.011	47	60	491	13,2%	10	PDF 1.4	text oriented, many links
sdw_2015_01.pdf	14.668.972	100	36	0	2,5%	143	PDF 1.3	graphics oriented
sdw_2015_02.pdf	13.295.864	100	38	0	2,7%	130	PDF 1.4	graphics oriented
sdw_2015_03.pdf	21.224.417	108	35	0	1,9%	192	PDF 1.4	graphics oriented
sdw_2015_04.pdf	15.242.911	108	37	0	2,7%	138	PDF 1.3	graphics oriented
sdw_2015_05.pdf	16.495.887	108	43	0	2,4%	149	PDF 1.4	graphics oriented
sdw_2015_06.pdf	23.447.046	100	38	0	1,6%	229	PDF 1.4	graphics oriented
sdw_2015_07.pdf	14.106.982	100	38	2	2,6%	138	PDF 1.4	graphics oriented
sdw_2015_08.pdf	12.321.995	100	37	0	3,0%	120	PDF 1.4	graphics oriented
sdw_2015_09.pdf	23.409.625	100	37	0	1,5%	229	PDF 1.4	graphics oriented
sdw_2015_10.pdf	18.706.394	100	24	0	2,0%	183	PDF 1.5	graphics oriented
sdw_2015_11.pdf	25.624.266	100	20	0	1,5%	250	PDF 1.4	graphics oriented
sdw_2015_12.pdf	19.111.666	108	36	0	2,1%	173	PDF 1.4	graphics oriented

Decimal point and comma follow European convention

E.g. `Adobe.pdf` and `PyMuPDF.pdf` are clearly text oriented, all other files contain many more images.

11.1 Part 1: Parsing

How fast is a PDF file read and its content parsed for further processing? The sheer parsing performance cannot directly be compared, because batch utilities always execute a requested task completely, 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.

In order to yet find an answer to the question, we therefore measure the time to copy a PDF file to an output file with each tool, and doing nothing else.

These were the tools

All tools are either platform independent, or at least can run both, 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 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.

Tool	Description
PyMuPDF	tool of this manual, appearing as "fitz" in reports
pdfrw	a pure Python tool, is being used by rst2pdf, has interface to ReportLab
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 was used:

PyMuPDF:

```
doc = fitz.open("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
```

Observations

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

Runtime	Tool			
File	1 fitz	2 pdfrw	3 pdftk	4 PyPDF2
Adobe.pdf	5,25	21,06	112,39	692,23
Evolution.pdf	0,16	0,46	1,05	0,89
PyMuPDF.pdf	0,04	0,19	0,82	0,88
sdw_2015_01.pdf	0,23	1,23	5,41	6,45
sdw_2015_02.pdf	0,29	1,52	7,05	6,70
sdw_2015_03.pdf	0,51	2,77	11,49	11,98
sdw_2015_04.pdf	0,31	2,15	7,44	7,21
sdw_2015_05.pdf	0,35	1,69	7,60	7,59
sdw_2015_06.pdf	0,75	3,31	13,97	14,54
sdw_2015_07.pdf	0,37	2,11	10,17	9,72
sdw_2015_08.pdf	0,46	1,94	8,80	8,69
sdw_2015_09.pdf	0,79	2,35	10,58	10,42
sdw_2015_10.pdf	0,36	1,88	3,53	6,64
sdw_2015_11.pdf	2,41	12,69	37,12	60,40
sdw_2015_12.pdf	0,51	2,19	9,25	10,03
Gesamtergebnis	**12,78**	**57,54**	**246,66**	**854,36**
	1,00	4,50	19,30	66,85
		1,00	4,29	14,85
			1,00	3,46

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

Runtime	Tool			
File	1 fitz	2 pdfrw	3 pdftk	4 PyPDF2
Evolution.pdf	0,16	0,46	1,05	0,89
PyMuPDF.pdf	0,04	0,19	0,82	0,88
sdw_2015_01.pdf	0,23	1,23	5,41	6,45
sdw_2015_02.pdf	0,29	1,52	7,05	6,70
sdw_2015_03.pdf	0,51	2,77	11,49	11,98
sdw_2015_04.pdf	0,31	2,15	7,44	7,21
sdw_2015_05.pdf	0,35	1,69	7,60	7,59
sdw_2015_06.pdf	0,75	3,31	13,97	14,54
sdw_2015_07.pdf	0,37	2,11	10,17	9,72
sdw_2015_08.pdf	0,46	1,94	8,80	8,69
sdw_2015_09.pdf	0,79	2,35	10,58	10,42
sdw_2015_10.pdf	0,36	1,88	3,53	6,64
sdw_2015_11.pdf	2,41	12,69	37,12	60,40
sdw_2015_12.pdf	0,51	2,19	9,25	10,03
Gesamtergebnis	**7,53**	**36,48**	**134,28**	**162,13**

1,00	4,84	17,82	21,52
	1,00	3,68	4,44
		1,00	1,21

PyMuPDF is by far the fastest: on average 4.5 times faster than the second best (the pure Python tool pdfrw, **chapeau pdfrw!**), and almost 20 times faster than the command line tool pdftk.

Where PyMuPDF only requires less than 13 seconds to process all files, pdftk affords itself almost 4 minutes.

By far the slowest tool is PyPDF2 - it is more than 66 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 only 21.5 times slower than PyMuPDF, 4.5 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 pdfrw and PyPDF2 **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.

11.2 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):

Runtime	Tool				
File	1 fitz (TEXT)	2 fitz bareJSON	3 fitz sortJSON	4 xpdf	5 pdfminer
Adobe.pdf	5,16	5,53	6,27	12,42	216,32
Evolution.pdf	0,29	0,29	0,33	1,99	12,91
PyMuPDF.pdf	0,11	0,10	0,12	1,71	4,71
sdw_2015_01.pdf	0,95	0,98	1,12	2,84	43,96
sdw_2015_02.pdf	1,04	1,09	1,14	2,86	48,26
sdw_2015_03.pdf	1,81	1,92	1,97	3,82	153,51
sdw_2015_04.pdf	1,23	1,27	1,37	3,17	80,95
sdw_2015_05.pdf	1,00	1,08	1,15	2,82	48,65
sdw_2015_06.pdf	1,83	1,92	1,98	3,70	138,75
sdw_2015_07.pdf	0,99	1,11	1,16	2,93	55,59
sdw_2015_08.pdf	0,97	1,04	1,12	2,80	48,09
sdw_2015_09.pdf	1,92	1,97	2,05	3,84	159,62
sdw_2015_10.pdf	1,10	1,18	1,25	3,45	74,25
sdw_2015_11.pdf	2,37	2,39	2,50	5,82	166,14
sdw_2015_12.pdf	1,14	1,19	1,26	2,93	69,79
Gesamtergebnis	**21,92**	**23,08**	**24,82**	**57,10**	**1321,51**

1,00	1,05	1,13	2,60	60,28
	1,00	1,08	2,47	57,27
		1,00	2,30	53,24
			1,00	23,15

Again, (Py-) MuPDF is the fastest around. It is 2.3 to 2.6 times faster than xpdf.

`pdfminer`, as a pure Python solution, of course is comparatively slow: MuPDF is 50 to 60 times faster and xpdf is 23 times faster. These observations in order of magnitude coincide with the statements on this web site.

11.3 Part 3: Image Rendering

We have tested rendering speed of MuPDF against the `pdftopng.exe`, a command lind tool of the **Xpdf** toolset (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
```

PyMuPDF invocation:

```
zoom = 150.0 / 72.0
mat = fitz.Matrix(zoom, zoom)
def ProcessFile(datei):
    print "processing:", datei
    doc=fitz.open(datei)
    for p in fitz.Pages(doc):
```

```
        pix = p.getPixmap(matrix=mat)
        pix.writePNG("t-%s.png" % p.number)
        pix = None
    doc.close()
    return
```

Xpdf invocation:

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

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

Render Speed	tool		
file	mudraw	pymupdf	xpdf
Adobe.pdf	105,09	110,66	505,27
Evolution.pdf	40,70	42,17	108,33
PyMuPDF.pdf	5,09	4,96	21,82
sdw_2015_01.pdf	29,77	30,40	76,81
sdw_2015_02.pdf	29,67	30,00	74,68
sdw_2015_03.pdf	32,67	32,88	85,89
sdw_2015_04.pdf	30,07	29,59	78,09
sdw_2015_05.pdf	31,37	31,39	77,56
sdw_2015_06.pdf	31,76	31,49	87,89
sdw_2015_07.pdf	33,33	34,58	78,74
sdw_2015_08.pdf	31,83	32,73	75,95
sdw_2015_09.pdf	36,92	36,77	84,37
sdw_2015_10.pdf	30,08	30,48	77,13
sdw_2015_11.pdf	33,21	34,11	80,96
sdw_2015_12.pdf	31,77	32,69	80,68
Gesamtergebnis	**533,33**	**544,90**	**1594,18**

1	1,02	2,99
	1	2,93

- MuPDF and PyMuPDF are both about 3 times faster than Xpdf.
- The 2% speed difference between MuPDF (a utility written in C) and PyMuPDF is the Python overhead.

CHAPTER

TWELVE

APPENDIX 2: DETAILS ON TEXT EXTRACTION

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)?

12.1 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.

12.2 Output of `getText("text")`

This function extracts a page's plain **text in original order** as specified by the creator of the document (which may not be equal to a natural reading order!).

An example output 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.
...
```

12.3 Output of `getText("html")`

HTML output reflects the structure of the page's `TextPage` - without adding much other benefit. Again an example:

```
<div class="page">
<div class="block"><p>
<div class="metaline"><div class="line"><div class="cell" style="width:0%;
↪align:left"><span class="s0">Tutorial</span></div></div>
</div></p></div>
<div class="block"><p>
<div class="line"><div class="cell" style="width:0%;align:left"><span class=
↪"s1">This tutorial will show you the use of MuPDF in Python step by step.</
↪span></div></div>
</div></p></div>
<div class="block"><p>
<div class="line"><div class="cell" style="width:0%;align:left"><span class=
↪"s1">Because MuPDF supports not only PDF, but also XPS, OpenXPS and EPUB␣
↪formats, so does PyMuPDF.</span></div></div>
<div class="line"><div class="cell" style="width:0%;align:left"><span class=
↪"s1">Nevertheless we will only talk about PDF files for the sake of brevity.
↪</span></div></div>
</div></p></div>
...
```

12.4 Output of `getText("json")`

JSON output reflects the structure of a `TextPage` and provides position details (`bbox` - boundary boxes in pixel units) for every block, line and span. This is enough information to present a page's text in any required reading order (e.g. from top-left to bottom-right). The output can obviously be made usable by `text_dict = json.loads(text)`. Have a look at our example program `PDF2textJS.py`. Here is how it looks like:

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

12.5 Output of `getText(output="xml")`

The XML version takes the level of detail even a lot deeper: every single character is provided with its position detail, and every span also contains font information:

```
<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"/>
...
```

The method's output can be processed by one of Python's XML modules. We have successfully tested `lxml`. See the demo program `fontlister.py`. It creates a list of all fonts of a document including font size and where used on pages.

12.6 Performance

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.

To begin with, all four methods are **very** fast in relation to what is there on the market. In terms of processing speed, we couldn't find a faster (free) tool.

Relative to each other, `xml` is about 2 times slower than `text`, the other three range between them. E.g. `json` needs about 13% - 14% more time than `text`.

Look into the previous chapter **Appendix 1** for more performance information.

INDEX