PyMuPDF Documentation

version 1.9

Ruikai Liu

Jorj McKie

May 11, 2016

Contents

The PyMuPDF Documentation

Introduction

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

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

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

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

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

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 (1.2), it was until only mid May 2015, that its creator and a few co-workers decided to elevate it to support the current release of MuPDF (first V1.7a then V1.8 in November 2016, and V1.9 and V1.9a since April 2016).

And we are determined to keep PyMuPDF also current in the future!

PyMuPDF runs and has been tested on Mac, Linux, Windows 7, Windows 10, Python 2 and Python 3 (x86 and x64 versions). Other platforms should work too as long as MuPDF and Python support them.

The main differences compared to version 1.2 are

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

Note on the Name `fitz`

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

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

License

PyMuPDF is distributed under GNU GPL V3 or later.

MuPDF is distributed under a variation of it: the **GNU AFFERO GPL V3**. While in earlier days this license has been more restrictive, version 3 is in effect not any more than GNU GPL. There are just some technical details on how / where you must make available any changes you might have made make to the **MuPDF library**. Other than that, nothing prevents you to distribute and even sell software you have built on the basis of MuPDF.

If you use the pre-built Windows binary installation, then both, the GNU AFFERO GPL and the UPX license automatically apply.

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 these bindings 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)`. All of the following image formats covered by MuPDF are thus also supported as inputs for pixmaps: BMP, JPEG, JXR, PNG, GIF, TIFF.
- The Pixmap constructor `fitz.Pixmap(data, len(data))` has been extended accordingly to support the above image formats as well (not just PNG as it did in version 1.8.0).
- `fitz.Rect` objects 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 method of `fitz.Document.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 method `fitz.Document.select(list)` removes all pages from an open 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 an alias of `fitz.Document()`.
- New method `fitz.Pixmap.getPNGData()` which will return a memory area formatted as a PNG image of the pixmap.
- New technical method `fitz.Pixmap.samplesRGB()` providing a `samples` version with alpha bytes stripped off (RGB colorspaces only).
- New technical method `fitz.Pixmap.samplesAlpha()` providing the alpha bytes only of the `samples` area.
- New iterator over a document's set of pages.

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 or PyPI, that fit your Python version.

Option 1: Install from Sources

Step 1: Download PyMuPDF

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

Step 2: Download and Generate MuPDF 1.9

Download `mupdf-1.9a-source.tar.gz` from MuPDF version 1.9a source, now and unzip / decompress it. Call the resulting folder `mupdf`.

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.

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

Controlling the Binary File Size. Since version 1.9, MuPDF includes support for many dozens of additional fonts for all sorts of alphabets like Chinese, Corean, Kyrillic, Indonesian, etc. If you do not (want to) control the amount of such support, the resulting binaries for MuPDF will be quite big and easily approach 20 MB. If you feel you do not need everything here, you need to manually change header file `fitz.h` (contained in directory `/include/mupdf`) by inserting `#define` statements like this:

```
#ifndef MUDPF_FITZ_H
#define MUDPF_FITZ_H

#define NOTO_SMALL          // choose smaller set of supported fonts
#define TOFU_CJK            // exclude Android specific font

#ifdef __cplusplus
extern "C" {
#endif
...
```

With the above two `#define` statements, the PyMuPDF binary should come down to a one digit MB size.

You may want to consult additional installation hints on PyMuPDF's main page on Github.com.

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`.

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, `.so` on Linux, and so forth.

Depending on your OS, your compiler and your font support choice (see above), this binary can be quite large: ranging from 8 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 4 MB to 9 MB without impacting functionality or execution speed.

Option 2: Install from Binaries

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, just download the zip file corresponding to your Python version, or get it from PyPI.

Step 2: Install PyMuPDF

Open a command prompt at the unzipped folder's top level and enter `python setup.py install`.

You are done after 2 seconds.

This process requires no compiler or Visual Studio and is **very** fast. The only pre-requisite is, that your Python configuration matches the zip file.

Targeting Parallel Python Installations

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

They contain **shebang lines** specifying 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.

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 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. Where this binding has added functionality, we usually decode string to unicode.

Import 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: the Python bindings for the MuPDF 1.9a library,
creation date 2016-05-10 18:09:34
>>>
```

Open a Document

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

```
doc = fitz.Document(filename)     # or fitz.open(filename) (since V1.9.0)
```

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 data, i.e. without using a file. See Document for details.

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

Some Document *Methods and Attributes*

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

Access Meta Data

`Document.metadata` is a Python dictionary with the following keys. For details of their meanings and formats consult the PDF manuals, e.g. Adobe PDF Reference sixth edition 1.7 November 2006. Further information can also be found in chapter Document. The meta data fields are of type string if not otherwise indicated. Be aware that not all of them may be present or 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

Work with Outlines

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

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

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

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

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

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

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

Work with Pages

Tasks that can be performed with a Page are at the core of MuPDF's functionality. Among other things, you can render a page, optionally zooming, rotating or shearing it. You can write it's image to files, 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)
```

Some typical uses of Page objects follow:

Inspect the Links on a Page

Here is how to get all links and their types:

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

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

Render a Page

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

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

Save the Page Image in a File

We can simply store the image in a PNG file:

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

Display the Image in Dialog Managers

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

wxPython:

```
bitmap = wx.BitmapFromBufferRGBA(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("RGBA", [pix.width, pix.height], pix.samples)
photo = ImageTk.PhotoImage(img)
```

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

Text Extraction

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

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

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

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

Text Searching

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

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

The variable `areas` will now contain a list of up to 32 Rect rectangles each of which surround an occurrence of string "mupdf" (case insensitive).

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

Output

Output capabilities of MuPDF (such as PDF generation) have improved in version 1.9. Output is supported for PDF documents only.

Re-arrange and Delete Pages

Method **`Document.select()`** accepts a list (or tuple) of integers as an argument. These integers must be in the range `0 <= i < countPage`. When executed, all pages not occurring in this list will be deleted. 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 (not) contain a certain text, ... whatever you may think of.

To make any such changes permanent, execute **`Document.save()`** (see below) and then close / re-open the original document for any further processing.

The saved document will contain all links, annotations and bookmarks referenced by its pages which still point to valid destinations.

Save

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

If you altered the document via **`Document.select()`**, then the resulting document will be saved. Do specify option `garbage=3` (see below) if many pages have been omitted by `select()`.

As part of `save()`, some clean-up will take place:

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

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

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

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

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

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

Close

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

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

Caution:

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

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

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

Example: Dynamically Clean up Corrupt PDF Documents

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

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

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

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

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

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

Classes

Colorspace

Represents the color space of a Pixmap.

Class API

class **Colorspace**

__init__ (self, colorspace)
Constructor

Parameters: **colorspace** -- A number identifying the colorspace. Possible values are **CS_RGB**, **CS_GRAY** and **CS_CMYK**.

Device

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

Class API

class **Device**

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

object
An object representing one of Pixmap, or DisplayList

Type: instance

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

textsheet
A TextSheet object.

Type: instance

textpage
A TextPage object.

Type: instance

DisplayList

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

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

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

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

Class API

class **`DisplayList`**

`__init__` `(self)`
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.

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.

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()	decrypts the document
Document.loadPage()	reads a page
Document.save()	saves a copy of the document
Document.getToC()	create a table of contents
Document.getPagePixmap()	create a pixmap from a page by its number
Document.getPageText()	extracts the text of a page by its number
Document.getPermits()	show permissions to access the document
Document.close()	closes the document
Document.select()	selects pages from a document, discards the rest
Document.isClosed	has document been closed?
Document.outline	first *Outline* item
Document.name	filename of document
Document.needsPass	require password to access data?
Document.isEncrypted	is document still encrypted?
Document.pageCount	the document's number of pages
Document.metadata	the document's meta data

Class API

class **Document**

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

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

Return type: Document

Returns: A Document object.

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

Parameters:

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

Return type: Document

Returns: A Document object.

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

Parameters: **password** (*string*) -- The password to be used.
Return type: int
Returns: True (1) if decryption with password was successfull, False (0) otherwise.

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

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

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

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

getPagePixmap (pno, matrix = fitz.Identity, colorspace = "rgb"**)**
Creates a pixmap from one of the document's pages - identified by number pno (zero-based).

Parameters:
- **pno** (*int*) -- Page number, zero-based
- **matrix** (*Matrix*) -- A transformation matrix - default is Identity.
- **colorspace** (*string*) -- A string specifying the requested colorspace - default is rgb.

Return type: Pixmap

getPageText (pno, output = "text"**)**
Extracts the text of a page given its page number pno (zero-based).

Parameters:
- **pno** (*int*) -- Page number, zero-based
- **output** (*string*) -- A string specifying the requested output format: text, html, json or xml. Default is text.

Return type: String

getPermits ()
Shows the permissions to access the document. Returns a dictionary likes this:

```
>>> doc.getPermits()
{'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

select (list**)**
Retains only those pages in the document that 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. Only PDF documents are supported by this method.

Parameters: **list** (*list*) -- A list (or tuple) of integers (zero-based) naming the pages to be included. Pages not occurring in the list will be deleted (from memory) and become unavailable until the document is reopened. Page numbers can occur multiple times and be in any order.

Return type: int

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

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

Parameters:

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

Return type: int

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

close ()

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

outline

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

Return type: Outline

isClosed

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

Return type: int

needsPass

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

Return type: bool

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

Return type: bool

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

Return type: dict

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

Return type: string

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

Return type: int

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 even or odd pages, re-arrange a document from back to front, duplicate it, and so forth. In combination with text extraction you can also omit / include pages with no text or certain text, etc.

You can execute several selections in a row. The document structure will be kept updated.

This also means, that any document information from before this method may no longer be valid. This is especially true for old page objects, table of contents and the `pageCount` property.

Any of those actions will become permanent only with a `doc.save()`. Do not forget to specify the `garbage=3` option to eventually reduce the resulting document's size.

It should also be noted, that this method **preserves all links, annotations and bookmarks** that are still valid. In other words: deleting pages only deletes references that would otherwise point to nowhere.

Examples

Create a document copy deleting pages with no text:

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(doc.pageCount))                  # list of all pages
for p in fitz.Pages(doc):
    txt = p.getText()                           # get the page's text
    if not txt:                                 # nothing there
        r.remove(p.number)                      # remove page number from list
doc.select(r)                                   # apply the list
doc.save("out.pdf", garbage=3)                  # save the resulting PDF
```

Create a sub document with the odd pages:

```
import fitz
doc = fitz.open("any.pdf")
r = [i for i in list(range(doc.pageCount)) if i%2 > 0]
doc.select(r)                                   # apply the list
doc.save("any-odd.pdf", garbage=3)              # save PDF with the odd pages
```

Concatenate a document with itself:

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(doc.pageCount))
r += r                                          # turn PDF into a copy of itself
doc.select(r)
doc.save("out.pdf")                             # contains pages 0,...,n,0,...,n of <any.pdf>
```

Create document copy in reverse page order:

```
import fitz
doc = fitz.open("any.pdf")
r = list(range(doc.pageCount-1, -1, -1))
doc.select(r)
doc.save("out.pdf")
```

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 and methods are hidden.

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

```
>>> m = fitz.Matrix(0).preRotate(45)
>>> m
fitz.Matrix(0.707106769085, 0.707106769085, -0.707106769085, 0.707106769085, 0.0, 0.0)
>>>
```

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

Class API

class **IRect**

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

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 (pixel points, default) or mm (square millimeters).

Return type: float

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

Type: int

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

Type: int

x0
X-coordinate of the top left corner.

Type: int

y0
Y-coordinate of the top left corner.

Type: int

x1
X-coordinate of the bottom right corner.

Type: int

y1
Y-coordinate of the bottom right corner.

Type: int

Example:

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import fitz
>>> irect = fitz.IRect(10, 10, 410, 610)
>>> irect
fitz.IRect(10, 10, 410, 610)
>>> irect.height
600
>>> irect.width
400
>>> irect.getRectArea(unit = 'mm')
29868.51852
```

Link

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

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

Class API

class **Link**

rect
The area that can be clicked in untransformed coordinates.

Return type: Rect

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

Return type: int

next
The next `Link` or `None`

Return type: Link

linkDest

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

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

Class API

class **linkDest**

dest
Destination of `linkDest`.

Return type: Link

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

Return type: string

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

Return type: int

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

Return type: bool

isUri
Specifies whether this destination is an internet resource.

Return type: bool

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

Return type: int

lt
The top left Point of the destination.

Return type: Point

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

Return type: int

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

Return type: bool

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

Return type: int

rb
The bottom right Point of this destination.

Return type: Point

uri
The name of the URI this destination points to.

Return type: string

Matrix

Matrix is a row-major 3x3 matrix used by image transformations in MuPDF (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 numerical values.

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

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

It should be noted, that

- the below methods are just convenience functions - everything they do, can also be achieved by directly manipulating `[a,b,c,d,e,f]`
- all manipulations can be combined - you can construct a matrix that does a rotate **and** a shear **and** a scale **and** a shift, etc. in one go. 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.__init__()	Constructor.
Matrix.preRotate()	Perform a rotation
Matrix.preScale()	Perform a scaling
Matrix.preShear()	Perform a shearing
Matrix.a	Zoom factor X direction
Matrix.b	Shearing effect Y direction
Matrix.c	Shearing effect X direction
Matrix.d	Zoom factor Y direction
Matrix.e	Horizontal shift
Matrix.f	Vertical shift

Class API

class **Matrix**

__init__ (self, 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), matrix `[sx, 0, 0, sy, 0, 0]` will be created. If shear is True, 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` and `d` only.

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

preShear (`sx, sy`**)**
Modify the matrix to perform a shearing, i.e. transformation of rectangles into parallelograms (rhomboids). Has effects on attributes `b` and `c` only.

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

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

Remarks

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** does in general not equal a **rotate - shift**. So it can become quite 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 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]`.

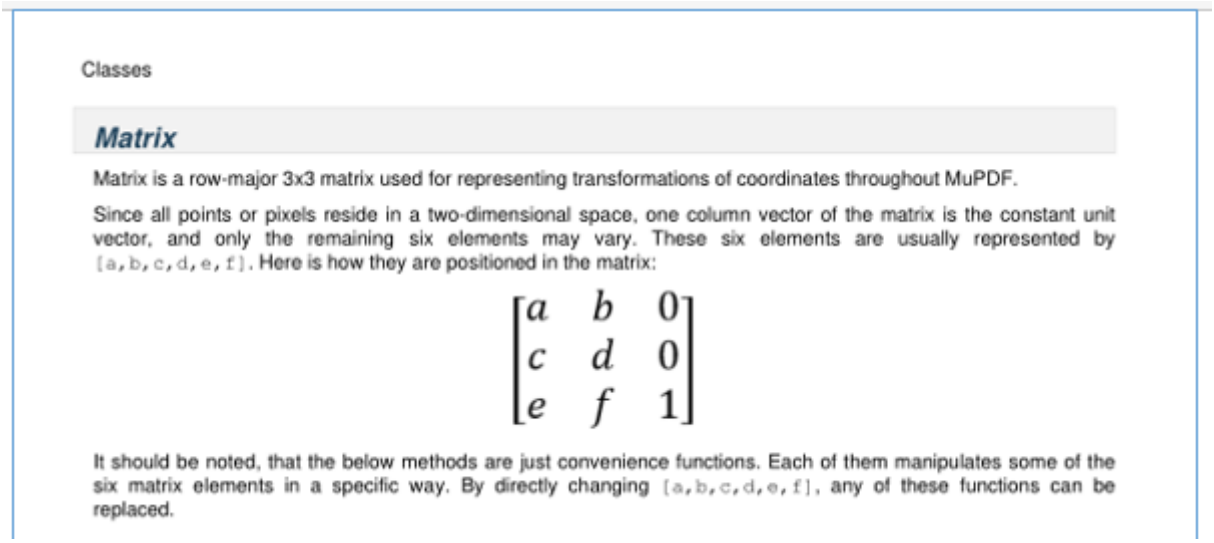
In order to obtain expected results, Adobe therefore recommends the following sequence when applying multiple matrix transformations (see page 206 of their manual):

- Shift ("translate")
- Rotate
- Scale or shear ("skew")

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.

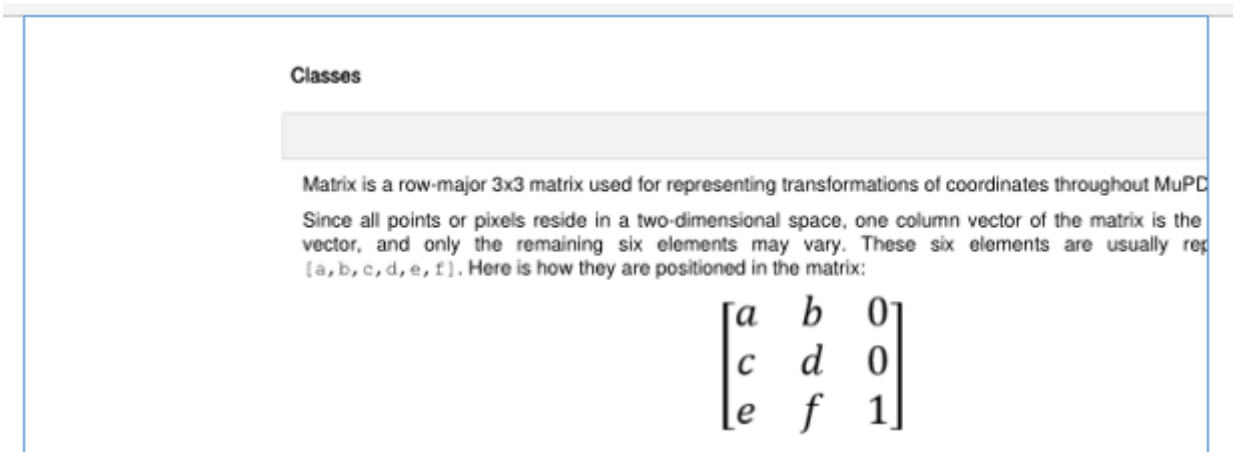
Since all points or pixels reside in a two-dimensional space, one column vector of the matrix is the constant unit vector, and only the remaining six elements may vary. These six elements are usually represented by [a,b,c,d,e,f]. Here is how they are positioned in the matrix:

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

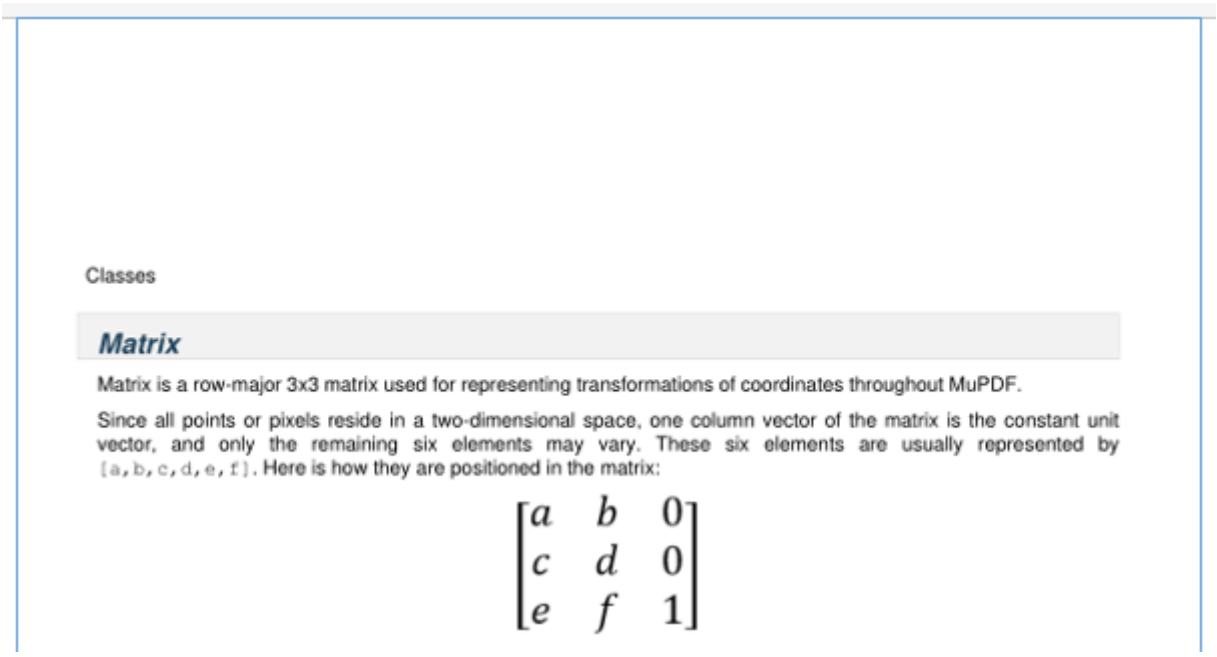
It should be noted, that the below methods are just convenience functions. Each of them manipulates some of the six matrix elements in a specific way. By directly changing [a,b,c,d,e,f], any of these functions can be replaced.

Shifting

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

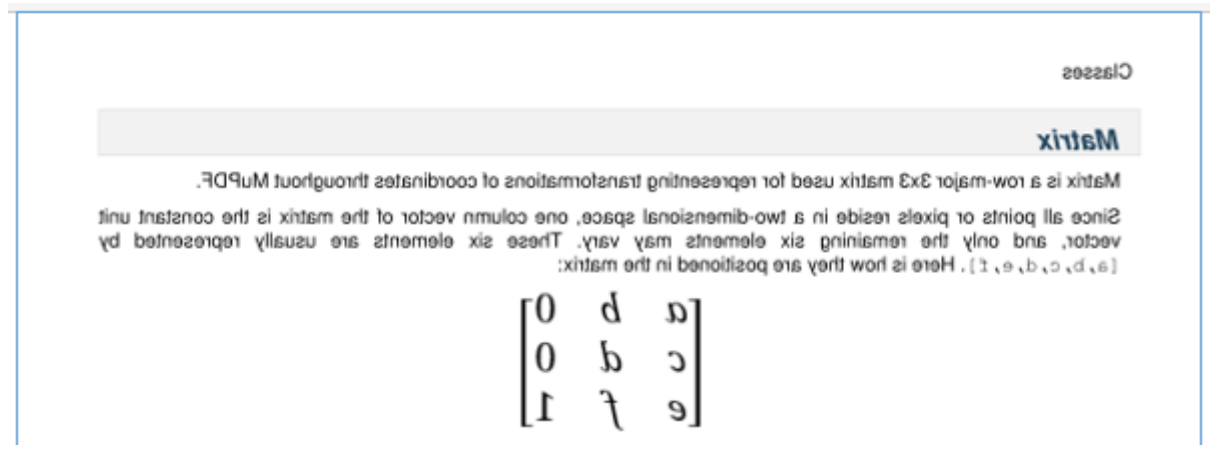


Next we do a down shift by 100 pixels: `f = 100`.

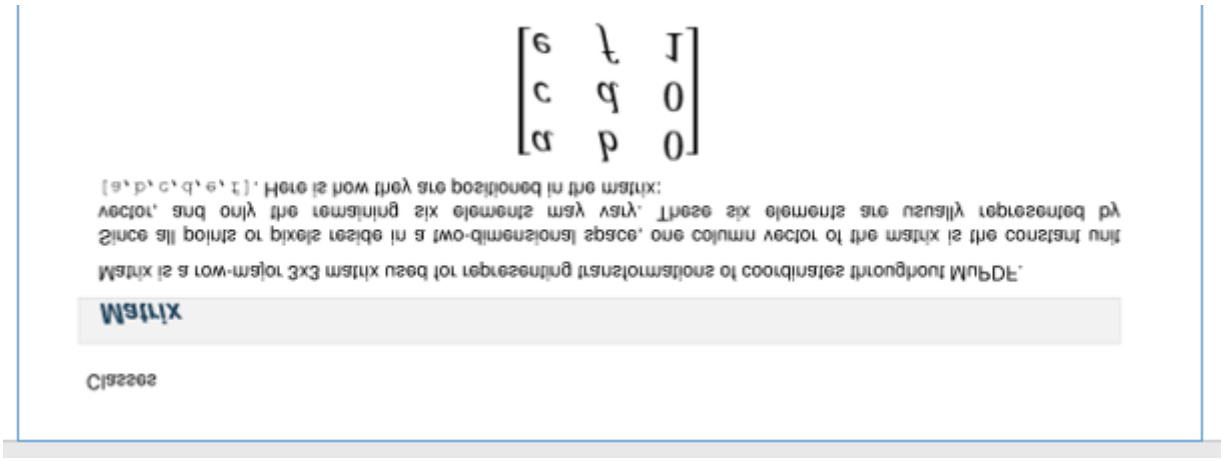


Flipping

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

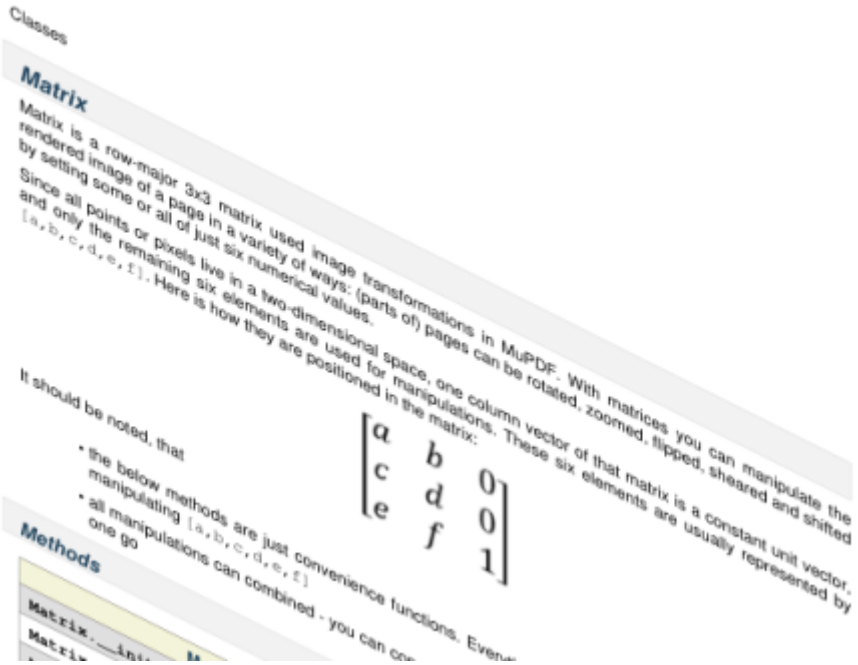


Flip up-down (`d = -1`).

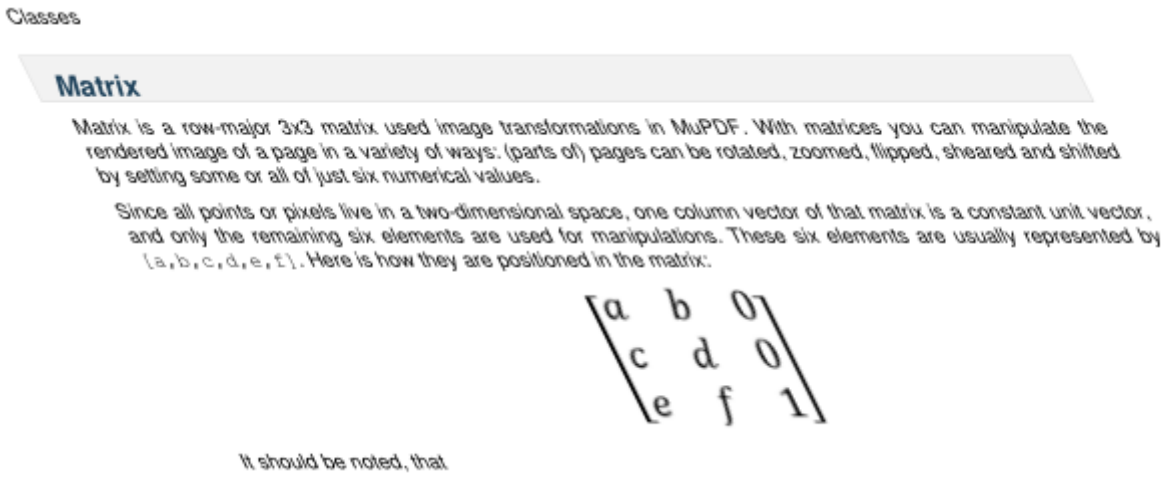


Shearing

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

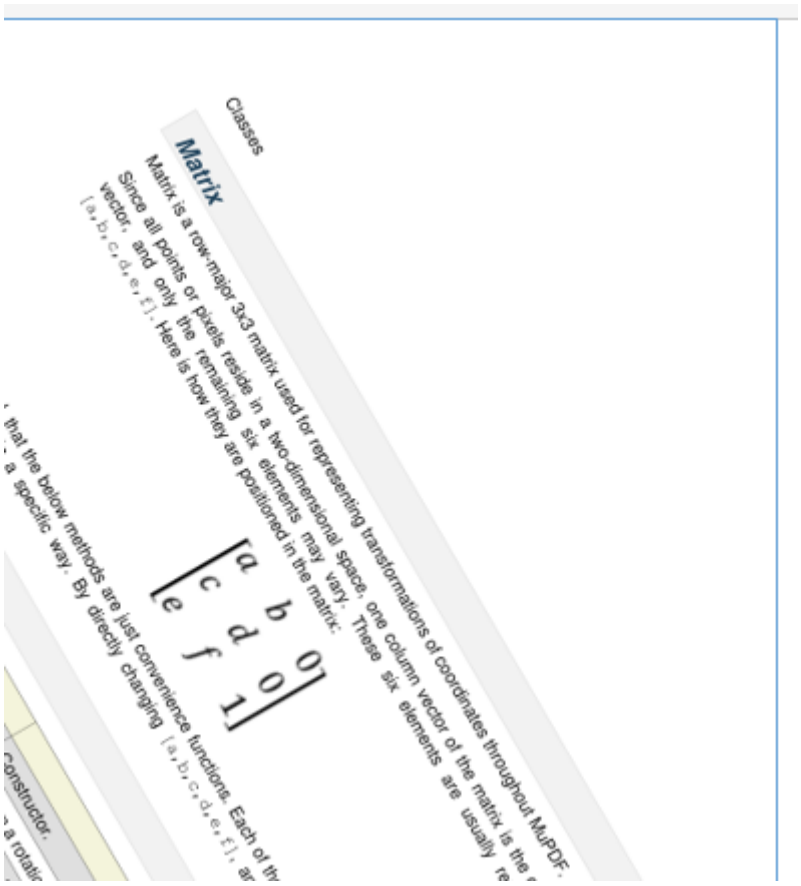


Second a shear in X direction (`c = 0.5`).



Rotating

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



Outline

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

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

Class API

class **`Outline`**

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

Return type: Outline

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

Return type: Outline

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

Return type: linkDest

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

Return type: string

`saveText (``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, while `page# = 0` if and only if the outline entry points to a place outside this document. If no title text was specified for an outline entry, it appears as a tab character in this file.

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.

Page

Class representing one document page. Can only be created by **`Document.loadPage()`**, there is no separate constructor defined.

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

Class API

class **Page**

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

Return type: Rect

`loadLinks ()`
Get the first link of a page.

Return type: Link

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

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

Return type: list

Returns: A list of dictionaries or `[]`. The entries are in the order as specified during PDF generation.

`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, the text is returned in the order as specified during PDF creation (which is not necessarily the normal reading order). As this may not always look like expected, consider using the example program `PDF2TextJS.py`. It is based on `extractJSON()` 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"`**`)`**
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).

Return type: Pixmap

Returns: Pixmap of the page.

`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.

`number`
The page number

Return type: int

`parent`
The owning document object.

Return type: Document

Pages

This is an iterator class over a document's set of pages.

Class API

class **Pages**

Pages (doc)
This creates an iterator over the pages of document doc.

Parameters: **doc** (Document) -- An opened document
Return type: iterator

Usage

The iterator object is constructed as follows:

```
doc = fitz.open(...)
pages = fitz.Pages(doc)

# this will loop through all the pages
for page in pages:
    # do something with the page. page.number contains current page number

# a single page can now also be accessed by its index
assert pages[20] == doc.loadPage(20)

# the len() function returns the number of pages
assert len(fitz.Pages(doc)) == doc.pageCount
```

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 alpha (transparency) byte.

In PyMuPDF, there exist several ways to create a pixmap:

- create an empty pixmap based on Colorspace and IRect information,
- create a pixmap from an in-memory image,
- create a pixmap from a memory area that contains plain pixels,
- create a pixmap from a document page (via methods **`Page.getPixmap()`** or **`Document.getPagePixmap()`**),
- create a pixmap from a file.

For supported image types using the file or in-memory constructors, see section below. Please 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.tintWith()`	tints a pixmap with a color
`Pixmap.gammaWith()`	applies a gamma factor to the pixmap
`Pixmap.writePNG()`	saves a pixmap as a PNG file
`Pixmap.getPNGData()`	returns a PNG as a memory area
`Pixmap.writeImage()`	saves a pixmap in a variety of image formats
`Pixmap.copyPixmap()`	copy parts of another pixmap
`Pixmap.getSize()`	returns the pixmap's total length
`Pixmap.getColorspace()`	returns the Colorspace used
`Pixmap.getIRect()`	returns the IRect used
`Pixmap.invertIRect()`	invert the pixels of a given area
`Pixmap.samplesRGB()`	RGB pixel data without alpha bytes
`Pixmap.samplesAlpha()`	returns the alpha bytes
`Pixmap.samples`	the components data for all pixels
`Pixmap.height`	height of the region in pixels
`Pixmap.width`	width of the region in pixels
`Pixmap.x`	X-coordinate of top-left corner of pixmap
`Pixmap.y`	Y-coordinate of top-left corner of pixmap
`Pixmap.n`	number of bytes per pixel
`Pixmap.xres`	resolution in X-direction
`Pixmap.yres`	resolution in Y-direction
`Pixmap.interpolate`	interpolation method indicator

Class API

class **`Pixmap`**

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

Parameters:

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

__init__ (self, filename)

This constructor creates a (non-empty) pixmap from file `filename`, which is assumed to contain a supported image.

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

__init__ (self, data, len)

This constructor creates a (non-empty) pixmap from `data`, which is assumed to contain a supported image of `len` bytes.

Parameters:

- **data** (*string*) -- Data containing a complete, valid image in the specified format. E.g. this may have been obtained from a statement like `data = open('somepic.png', 'rb').read()`. The origin of the resulting pixmap is (0,0).
- **len** (*int*) -- An integer specifying the length of `data`.

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

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 may be usefull to create a full copy of a given pixmap, or when raw image data are being provided by some other program - see examples below.

Parameters:

- **colorspace** (Colorspace) -- Colorspace of the image. This crucial parameter controls the interpretation of the `samples` area: for **CS_GRAY**, **CS_RGB** and **CS_CMYK**, 2, 4 or 5 bytes in samples will be assumed to define one pixel, respectively.
- **width** (*int*) -- Width of the image
- **height** (*int*) -- Height of the image
- **samples** (*string*) -- A string containing consecutive bytes describing all pixels of the image.

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). 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, omit this parameter.

Parameters: **irect** (IRect) -- The IRect 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`**). The copy mechanism automatically adjusts to any 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 first being calculated. 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, modify its x and y attributes accordingly 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.

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

Return type: int

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

Return type: Colorspace

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

Return type: IRect

writePNG (filename**,** savealpha=False**)**
Saves a pixmap as a PNG file. Please note that only grayscale and RGB colorspaces can be saved in PNG format (this is not a PyMuPDF restriction).

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

getPNGData (savealpha=False**)**
Returns the pixmap data as an image area in PNG format.

Parameters: **savealpha** (*bool*) -- Save the alpha channel (`True`) or not (`False` - the default).

Return type: string

writeImage (filename, output="png", savealpha=False**)**
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.

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`.
- **savealpha** (*bool*) -- Save the alpha channel (`True`) or not (`False` - the default).

samplesRGB ()
Returns the pixmap samples (see below) without alpha bytes (currently RGB only). This is a technical function: occasionally dialog managers cannot deal with the RGBA format and either expect RGB data only, or eventually a separate alpha channel alongside.

Return type: bytearray

samplesAlpha ()
Returns the alpha channel of the pixmap's samples area (see below). This is a technical function: occasionally dialog managers cannot deal with the RGBA format and either expect RGB data only, or eventually a separate alpha channel alongside.

Return type: bytearray

samples
The color and transparency values for all pixels. `samples` is a memory area of size `width * height * n` bytes. Each n bytes define one pixel. Each successive n bytes yield another pixel in scanline order. Subsequent scanlines follow each other with no padding. E.g. for an RGBA colorspace (i.e. `n = 4`) this means, `samples` is a bytearray like `..., R, G, B, A, ...`, and the four byte values R, G, B, A 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 additional image formats. See example 3.

Return type: bytearray

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 (and identifies) the chosen colorspace: `CS_GRAY` = 2, `CS_RGB` = 4, `CS_CMYK` = 5.

Return type: int

xres
Horizontal resolution in dpi (dots per inch).

Return type: int

yres
Vertical resolution in dpi.

Return type: int

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

Supported Pixmap Construction Image Types

Support includes the following file types: BMP, JPEG, GIF, TIFF, JXR, and PNG.

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

Pixmap Example Code Snippets

Example 1

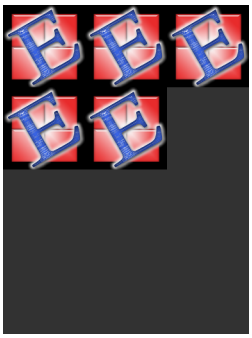
This shows how pixmaps can be used for purely graphical, non-PDF purposes. The script reads a PNG picture and creates a new PNG file which consist of 3 * 4 tiles of the original one:

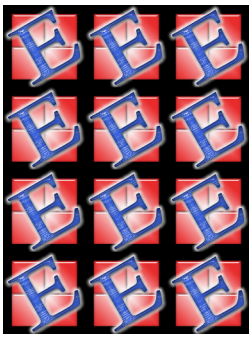
```
import fitz
# read in picture image and create a pixmap of it
try:
    pix0 = fitz.Pixmap("editra.png")
except:
    raise ValueError("file does not exist or has invalid format")

# calculate target colorspace and pixmap dimensions and create it
tar_cs     = pix0.getColorspace()        # use colorspace of input
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)
tar_pix    = fitz.Pixmap(tar_cs, tar_irect)
tar_pix.clearWith(90)        # clear target with a lively gray :-)
```

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

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

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, 4), 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, 255]

samples = bild.tostring()       # get plain pixel data from numpy array
pix=fitz.Pixmap(fitz.Colorspace(fitz.CS_RGB), width, height, samples)
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("RGBA", [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("RGBA")
samples = img.tobytes()
pix = fitz.Pixmap(fitz.csRGB, img.size[0], img.size[1], samples)
```

Point

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

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

Class API

class **`Point`**

`__init__ (self[, x, y])`

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

Parameters:

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

`__init__ (self, p)`

Constructor. Creates a **new copy** of the point.

Parameters: **p** (Point) -- The point to copy from.

Example:

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

Rect

`Rect` represents a rectangle defined by its top left and its bottom right Point objects, in coordinates: ((x0, y0), (x1, y1)). Respectively, a rectangle can be defined in one of the 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".

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

Class API

class **Rect**

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

`__init__ (self, p1, p2)`

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

`__init__ (self, x0, y0, p2)`
Overloaded constructors: `p1`, `p2` stand for Point objects, while the other parameters mean float coordinates.

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

Return type: IRect

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

Parameters: **m** (Matrix) -- The matrix to be used for the transformation.
Return type: Rect

`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

Example 1:

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import fitz
>>> p1 = fitz.Point(10, 10)
>>> p2 = fitz.Point(300, 450)
>>> 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:

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

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

TextPage

TextPage represents the text of a page.

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

Class API

class **TextPage**

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

Return type: string

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

Return type: string

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

Return type: string

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

Return type: string

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

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

Return type: list

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

TextSheet

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

Functions

The following are miscellaneous functions directly available under the binding name, i.e. can be invoked as `fitz.function`.

Function	Short Description
getPointDistance()	calculates the distance between to points

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

Constants and Enumerations

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

Constants

Constant	Description
CS_RGB	1 - Type of Colorspace is RGBA
CS_GRAY	2 - Type of Colorspace is GRAY
CS_CMYK	3 - Type of Colorspace is CMYK
csRGB	= fitz.Colorspace(fitz.CS_RGB)
csGRAY	= fitz.Colorspace(fitz.CS_GRAY)
csCMYK	= fitz.Colorspace(fitz.CS_CMYK)
VersionBind	'1.9.1' - string: version of PyMuPDF (this binding)
VersionFitz	'1.9a' - string: version of MuPDF / fitz

Enumerations

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

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

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

Value	Description
LINK_FLAG_L_VALID	1 (bit 0) Top left x value is valid
LINK_FLAG_T_VALID	2 (bit 1) Top left y value is valid
LINK_FLAG_R_VALID	4 (bit 2) Bottom right x value is valid
LINK_FLAG_B_VALID	8 (bit 3) Bottom right y value is valid
LINK_FLAG_FIT_H	16 (bit 4) Horizontal fit
LINK_FLAG_FIT_V	32 (bit 5) Vertical fit
LINK_FLAG_R_IS_ZOOM	64 (bit 6) Bottom right x is a zoom figure

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

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

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.

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.

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.

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)?

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.

Output of `getText(output="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.
...
```

Output of `getText(output="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
</div></p></div>
<div class="block"><p>
<div class="line"><div class="cell" style="width:0%;align:left"><span class="s1">This tutori
</div></p></div>
<div class="block"><p>
<div class="line"><div class="cell" style="width:0%;align:left"><span class="s1">Because MuP
<div class="line"><div class="cell" style="width:0%;align:left"><span class="s1">Neverthele
</div></p></div>
...
```

Output of `getText(output="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."
         }
       ]
      }
   ]
  },
...
```

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.

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

(Rect attribute)

yres (Pixmap attribute)