Python-Fitz Documentation

version 1.7

Ruikai Liu

Jorj McKie

June 10, 2015

Contents

The Python-Fitz Documentation

Introduction

python-fitz is a Python binding for MuPDF - "a lightweight PDF and XPS viewer".

MuPDF can access files in PDF, XPS, OpenXPS and EPUB formats.

These are files with file extensions `*.pdf`, `*.xps`, `*.oxps` or `*.epub` (so in essence, this binding can also serve to use **Python scripts as e-book viewers** ...)

python-fitz provides access to all important functions of MuPDF from within a Python environment. We are continuously increasing the set of accessible MuPDF functions.

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

You can check this out yourself: Compare the various free PDF-viewers. In terms of speed and rendering quality SumatraPDF ranges at the top - and it is based on MuPDF!

While these bindings have been available since several years for an earlier version of MuPDF (1.2), it was until only recently (mid May 2015), that its creator and a few co-workers decided to elevate this repository to the current release of MuPDF, which now counts at version 1.7a.

And we are determined to keep python-fitz current with major MuPDF releases in the future!

This work is almost completed - final tests and bug fixes are underway.

If you know how to build MuPDF on your platform (or you could use our development binaries - just drop a note), then you can use this repository to **make PDF, XPS, OpenXPS and EPUB available** to your Python scripts already **today** - everything works!

python-fitz can be used today on LINUX, Windows 7, Python 2 and Python 3.

So, what do we have?

- We have a ready SWIG-generated wrapper that has been tested on LINUX and Windows installations.
- We have demo scripts for typical use cases that you can take as templates for your development.
- We have a detailed description of how to install under Windows and Python 2.

So, what is still missing then?

- New documentation is under construction, though we are making progress - you are looking at its result. Do have a look at the demos and examples provided to learn how things work in practice.
- Some tests are still outstanding, e.g. for the combination Win & Python 3.
- We plan to simplify the installation procedure for the majority of potential users. In detail this means that **binaries will be supplied** e.g. for Windows and popular other platforms to reduce the setup effort (e.g. for Windows the installation will just require to put `fitz.py` and `_fitz.pyd` at a place where your Python will find it - and you are all set!).

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

Installation

This describes how to install python-fitz.

Step 1: Download python-fitz

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

Step 2: Download MuPDF 1.7a

Download MuPDF version 1.7(a) source, and unzip it. Let us call the resulting folder `mupdf17`.

You can put it inside `PyFitz` as a subdirectory, if you want keep everything in one place.

Step 3: Generate MuPDF

For many platforms supported by MuPDF, generating scripts are provided. I.e. makefiles exist for Linux / Unix platforms and Visual Studio solutions / projects are provided for Windows. As MuPDF depends on a number of third party software components, you must generate these, too. Follow the instructions provided with the MuPDF download package.

For Windows, you only need to BUILD the solution in Visual Studio (set the configuration to "Release Win32"). This will generate MuPDF together with any third party software.

The result of this step will be libraries, which are needed for the setup of python-fitz.

On Windows, the following two libraries are required: `libmupdf.lib` and `libthirdparty.lib`.

On Unix / Linux platforms, the libraries are these: `mupdf.a`, `mujs.a`, `crypto.a`, `jbig2dec.a`, `openjp2.a`, `jpeg.a` and `freetype.a`.

Step 4: Build / Setup python-fitz

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

- the include directory is correctly set,
- the directory for the libraries created in Step 3 is correcly set.

Now perform a `python setup.py install`

Tutorial

This tutorial will show you the use of MuPDF in Python step by step. You should note that MuPDF supports not only PDF files, but also XPS, OpenXPS and EPUB files. The MuPDF bindings for Python also support all of these filetypes. Nevertheless we will only talk about PDF's for the sake of brevity.

Import the Bindings

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

```
import fitz
```

Open a Document

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

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

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

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

Access the Meta Data

Document.metadata is a Python dictionary with the following keys. For details of their meanings and formats consult the PDF manuals. The meta data fields are of type string if not otherwise indicated and may be missing, in which case they contain `None`.

producer	Producer (producing software)
format	PDF release, e.g. 'PDF 1.4'
encryption	Encryption method method used
author	Author
modDate	Date of last modification
keywords	Keywords (dictionary)
title	Title
creationDate	Date of creation
creator	Creating application
subject	Subject

Work with Outlines

Entering the documents outline tree works like this:

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

Some `Document.outline` functions and attributes

saveText()	Save table of contents text to a file.
saveXML()	Save table of contents quasi-XML to a file.
next	Next item of the same level
down	Next item one level down
title	Title of this item (UTF-8).
dest	Destination ('where does this entry point to?').

Some `Document.outline.dest` functions and attributes

page	Target page number.
lt	Top-left corner of target rectangle.
rb	Bottem-right corner of target rectangle.

MuPDF also supports outline destinations to other files and URIs into which we will not dive here.

In order to get a complete Python outline list ("table of contents") of a document, use the following function:

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

Work with Pages

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

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

Some typical uses of *Page* objects:

1. Inspect the links on a *Page*:

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

2. Render a *Page*:

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

2.1 Example: save the page image as a png file:

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

2.2 Example: convert the image to a Bitmap for use in the wxPython dialog manager:

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

3. Extract the text of a *Page*:

```
dl = fitz.DisplayList()                 # create a DisplayList
ts = fitz.TextSheet()                   # create a TextSheet
tp = fitz.TextPage()                    # create a TextPage
dev = fitz.Device(ts, tp)               # create a text Device
# now run the page through the created device
dl.run(dev, fitz.Identity, irect)
# Extract the complete text of the page now contained in the TextPage.
# Includes all whitespace (tabulation, end-of-line, etc.) characters, too.
text = tp.extractText()                 # remember: UTF-8 encoding!
```

Classes

The list of python-fitz classes, to be used as `fitz.class`.

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

Colorspace

Represents the color space of a *Pixmap*.

Class API

class **Colorspace**

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

colorspace
A number identifying the colorspace. Currently only RGBA is supported (`fitz.CS_RGB`).

Type: int

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

Type: instance

Device

The different format handlers (pdf, xps, etc.) interpret pages to a "device". These devices are the basis for everything that be done with a page, like rendering, text extraction and searching. What will actually be done with a page, depends on the argument type used in constructing a device.

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.

Methods

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

Class API

class **DisplayList**

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

Return type: *Device*

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

Parameters:

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

Document

This class represents a document and is constructed by `fitz.Document(filename)`. This will also **open** the document specified as `filename`. Returns a `Document` object.

Methods and Attributes

Method / Attribute	Short Description
`Document.authenticate()`	Decrypts the document.
`Document.loadPage()`	Reads a page.
`Document.save()`	Saves a copy of the document.
`Document.ToC()`	Creates a table of contents.
`Document.close()`	Closes the document.
`Document.outline`	First *Outline* item.
`Document.needsPass`	Is document is encrypted?
`Document.pageCount`	The document's number of pages.
`Document.metadata`	The document's meta data.

Class API

class **`Document`**

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

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

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

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

`save (filename)`
Saves a copy of the document under the `filename` (absolute or relative path specifications). Internally the document may have changed, i.e. if the document has been decrypted before, an unencrypted copy will be saved.

Parameters: **filename** (*string*) -- The filename to save to. Must be different from the original file name.

`ToC ()`
Creates a table of contents from the `outline` entries. This will be a Python list `[[level, title, page], [...], ...]` or `[]` if there are no outline entries. Note that the title entries are unicode strings.

Return type: list

`close ()`
Closes `filename` thus freeing it for other purposes.

`outline`
Contains either `None` or the first *Outline* entry of the document. Can be used as a starting point to walk through all outline items.

Return type: *Outline*

`needsPass`
Contains an indicator showing whether the document is encrypted (`True` (`1`)) or not (`False` (`0`)).

Return type: bool

`metadata`
Contains the document's meta data as a Python dictionary. Its keys are `format`, `encryption`, `title`, `author`, `subject`, `keywords`, `creator`, `producer`, `creationDate`, `modDate`. These key names correspond to the PDF's "official" meta data fields `/Creator`, `/Producer`, `/CreationDate`, `/ModDate`, `/Title`, `/Author`, `/Subject`, `/Keywords` respectively where applicable. `format` contains the PDF format version of the file (e.g. 'PDF 1.4'), `encryption` contains either `None` when not encrypted, or a string naming the encryption method used (e.g. `'Standard V4 R4 128-bit RC4'`). Note that all other metadata values are encrypted if the value for 'encoding' is not `None`. All item values are UTF-8 encoded strings (or `None`), except `keywords`. If `keywords` is not `None`, it contains a Python dictionary specifying the document's keywords (again, as UTF-8 encoded strings). The date fields are strings with the internal timestamp format "D:<DateTime><TZ>", where <DateTime> is the 12 character ISO date `YYYMMDDhhmmss` (`YYYY` - year, `MM` - month, `DD` - day, `hh` - hour, `mm` - minute, `ss` - second), and <TZ> is a time zone value (time intervall relative to GMT) containing a sign ('+' or '-'), the hour (hh), and minute (`'mm'`, attention: enclose in apostrophies!). For example, a Venezuelan value might look like `D:20150415131602-04'30'`, which corresponds to the timestamp April 15, 2015, at 1:16:02 pm local time Venezuela.

Return type: dict

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

Return type: int

Identity

Identity is just a *Matrix* that performs no action. The default constructor of *Matrix* creates an identity matrix.

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.

Seealso

Rect

Attributes

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

Class API

class **`IRect`**

`__init__` **(*`self`*, *`x0=0`*, *`y0=0`*, *`x1=0`*, *`y1=0`*)**
Constructor. The default values will create an empty rectangle. Function **`Rect.round()`** creates the smallest `IRect` containing `Rect`.

`width`
Contains the width of the bounding box.

Type: int

`height`
Contains the height of the bounding box.

Type: int

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

Type: int

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

Type: int

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

Type: int

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

Type: int

Link

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

Attributes

Attribute	Short Description
`Link.rect`	Clickable area in untransformed coordinates.

Link.dest	Kind of link destination.
Link.next	Link to next link
Link.refs	Target page number of an internal link

Class API

class **Link**

rect
The area that can be clicked in untransformed coordinates.

Return type: *Rect*

dest
Get the kind of link destination.

Return type: **fz_link_kind**

next
The next `Link` or `None`

Return type: *Link*

refs
The ???

Return type: *Link*

linkDest

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

Attributes

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`
something

Return type: *Link*

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

Return type: string

`flags`
The flags is a 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 was actually specified in the file. 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 on / `True`, so do not test for the value of the integer.

Return type: int

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

Return type: bool

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

Return type: bool

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

Return type: int

`lt`
The top left *Point* of the destination.

Return type: *Point*

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

Return type: int

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

Return type: bool

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

Return type: int

`rb`
The bottom right *Point* of this destination.

Return type: *Point*

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

Return type: string

Matrix

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

Since all points reside in a two-dimensional space, one vector is always a constant unit vector; hence only some elements may vary in a matrix. Below is how the elements map between different representations:

```
/ a b 0 \
| c d 0 |
\ e f 1 /
```

normally represented as [a b c d e f].

Methods

Matrix.__init__()	Constructor.
Matrix.preRotate()	Perform a rotation
Matrix.preScale()	Perform a scaling
Matrix.preShear()	Perform a shearing

Attributes

Matrix.a	Matrix entry at (1, 1)
Matrix.b	Matrix entry at (1, 2)
Matrix.c	Matrix entry at (2, 1)
Matrix.d	Matrix entry at (2, 2)
Matrix.e	Matrix entry at (3, 1)
Matrix.f	Matrix entry at (3, 2)

Class API

class **Matrix**

__init__ (*self*, *a=1*, *b=0*, *c=0*, *d=1*, *e=0*, *f=0*)
Constructor. The default values will construct an identity matrix.

preRotate (*deg*)
Perform a rotation for deg degrees

Parameters: **degree** -- The extent of the rotation in degrees.
Return type: *Matrix*

preScale ()
Scale

Return type: *Matrix*

preShear ()
Shear

Return type: int

a

Matrix entry at (1, 1), default value 1.

Type: float

b
Matrix entry at (1, 2), default value 0.

Type: float

c
Matrix entry at (2, 1), default value 0.

Type: float

d
Matrix entry at (2, 2), default value 1.

Type: float

e
Matrix entry at (3, 1), default value 0.

Type: float

f
Matrix entry at (3, 2), default value 0.

Type: float

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 either "horizontal" direction by property `.next` to the next item of same level, or downwards with property `.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".

Methods and Attributes

Method / Attribute	Short Description
`Outline.down`	Next item downwards
`Outline.next`	Next item same level
`Outline.dest`	Link destination
`Outline.title`	Title (UTF-8 string)
`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 UTF-8 string.

Return type: string

`saveText ()`
The chain of outline items is being processed and printed to a file `filename` as a conventional table of contents.

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

`saveXML ()`
The chain of outline items is being processed and printed to a file `filename` as an XML-like table of contents.

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

Page

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

Methods

`Page.bound()`	Page size before transformation.
`Page.loadLinks()`	Get all the links in a page.
`Page.run()`	Run a page through a device.

Class API

class **`Page`**

`bound ()`
Determine the size of a page before transformation.

Return type: *Rect*

`loadLinks ()`
Get all the links in a page.

Return type: list

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

`run (dev, transform)`
Run a page through a device.

Parameters:

- **dev** (*Device*) -- Device obtained from **`new_*_device()`**.
- **transform** (*Matrix*) -- Transform to apply to page. May include for example scaling and rotation, see **`Matrix.preScale()`** and **`Matrix.preRotate()`**. Set it to *Identity* if no transformation is desired.

Pixmap

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

Methods and Attributes

Method / Attribute	Short Description
`Pixmap.clearWith()`	Clears a pixmap (with given value)
`Pixmap.writePNG()`	Saves a pixmap as a png file
`Pixmap.invertIRect()`	Invert the pixels of a given bounding box
`Pixmap.interpolate()`	unknown
`Pixmap.samples`	The components data for all pixels
`Pixmap.h`	Height of the region in pixels
`Pixmap.w`	Width of the region in pixels
`Pixmap.x`	X-coordinate of top-left corner of pixmap
`Pixmap.y`	Y-coordinate of top-left corner of pixmap
`Pixmap.n`	Number of components per pixel
`Pixmap.xres`	Resolution in X-direction
`Pixmap.yres`	Resolution in Y-direction

Class API

class **`Pixmap`**

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

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

`samples`
The color and transparency values for all pixels. Samples is a memory area of size `width * height * n` bytes. The first n bytes are components 0 to n-1 for the pixel at point (x,y). Each successive n bytes gives another pixel in scanline order. Subsequent scanlines follow each other with no padding. E.g. for an RGBA colorspace this means, `samples` is a bytearray like `..., R, G, B, A, ...`, and the four byte values R, G, B, A describe one pixel (RGBA is the only supported colorspace at this time).

Return type: bytearray

`w`
The width of the region in pixels.

Return type: int

`h`
The height of the region in pixels.

Return type: int

`x`
X-coordinate of top-left corner

Return type: int

y
Y-coordinate of top-left corner

Return type: int

n
Number of components per pixel

Return type: int

xres
Horizontal resolution

Return type: int

yres
Vertical resolution

Return type: int

invertIRect (*self*, *irect*)
Invert all pixels in *IRect*. All components except alpha are inverted.

Parameters: **irect** -- Invert all the pixels in the irect. If not given, the whole pixmap will be inverted.

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

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

interpolate ()
Unknown ...

Parameters: **unknown** -- Unknown.

Point

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

Methods

Point.__init__()	Constructor.

Attributes

Point.x	The X- coordinate.
Point.y	The Y- coordinate.

Class API

class **Point**

__init__ (*self*, *x=0*, *y=0*)
Constructor, defaulting to "top left".

x

Type: float

y

Type: float

Rect

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

Rectangles are always aligned with the respective X- and Y-axes. If x0 <= x1 and y0 <= y1 is true, the rectangle is called "finite", else "infinite".

Methods

`Rect.round()`	creates the smallest *IRect* containing `Rect`
`Rect.transform()`	transform `Rect` with a *Matrix*

Attributes

`Rect.height`	`Rect` height.
`Rect.width`	`Rect` width.
`Rect.x0`	Top left corner's X-coordinate.
`Rect.y0`	Top left corner's Y-coordinate.
`Rect.x1`	Bottom right corner's X-coordinate.
`Rect.y1`	Bottom right corner's Y-coordinate.

Class API

class **`Rect`**

`__init__` (***`self`***, ***`x0=0`***, ***`y0=0`***, ***`x1=0`***, ***`y1=0`***)
Constructor. The default values will create an empty rectangle.

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

Return type: *IRect*

`transform`
Transforms `Rect` with a *Matrix*.

Return type: *Rect*

`width`
Contains the width of the rectangle.

Return type: float

`height`
Contains the height of the rectangle.

Return type: float

x0
X-coordinate of the top left corner.

Type: float

y0
Y-coordinate of the top left corner.

Type: float

x1
X-coordinate of the bottom right corner.

Type: float

y1
Y-coordinate of the bottom right corner.

Type: float

TextPage

`TextPage` contains the text of a page.

Methods

`TextPage.extractText()`	Extract the page's text.
`TextPage.search()`	Search for a string in the page.

Class API

class **`TextPage`**

`extractText (self)`
Extract the text from a `TextPage` object. Returns a UTF-8 encoded string of the page's complete text.

Return type: string

`search (self, string, maxhit)`
Search for the string `string`.

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

Return type: list

Returns: A python list. Each element of the list is an *IRect* (without transformation) surrounding a found `string` occurrence (or an empty list).

TextSheet

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

Constants and Enumerations

Here are the constants and enumerations of MuPDF as implemented by python-fitz.

Constants

`fitz.`**`CS_RGB`**	Type of *Colorspace* is RGBA

Enumerations

Possible values of **`linkDest.kind`** (link destination type).

Value	Description
`FZ_LINK_NONE`	0 - No destination.
`FZ_LINK_GOTO`	1 - Points to a place in this document.
`FZ_LINK_URI`	2 - Points to an URI.
`FZ_LINK_LAUNCH`	3 - Launches (opens) a file.
`FZ_LINK_NAMED`	4 - Performs some action.
`FZ_LINK_GOTOR`	5 - Points to a place in another document.

Possible values of **`linkDest.flags`** (link destination flags). **Attention**: these values represent boolean indicators, of which several can be `True` or `False`, and not numerical values. I.e. **`linkDest.flags`** is the sum of these values.

Value	Description
`LINK_FLAG_L_VALID`	1 - Top left x value is valid.
`LINK_FLAG_T_VALID`	2 - Top left y value is valid.
`LINK_FLAG_R_VALID`	4 - Bottom right x value is valid.
`LINK_FLAG_B_VALID`	8 - Bottom right y value is valid.
`LINK_FLAG_FIT_H`	16 - Horizontal fit.
`LINK_FLAG_FIT_V`	32 - Vertical fit.
`LINK_FLAG_R_IS_ZOOM`	64 - Bottom right x is a zoom figure.

Indices and tables

- *genindex*
- *search*

Index