PYTHON-FITZ 1.7 Documentation

version 1.7

Ruikai Liu
Jorj McKie

June 09, 2015

Contents

Welcome to the documentation of PYTHON-FITZ 1.7!

Contents:

Welcome to PYTHON-FITZ!

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. The interface has changed: compared to 1.2, it has been simplified and looks a lot less cryptic. As for now, you would have to look at the demos to see how things work.
- 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!

Tutorial

Welcome to the documentation of PYTHON-FITZ 1.7!

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    # this is 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)              # create the Page object representing page n of the
```

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

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
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 (floating point 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*, *obj*)
    Constructor
  obj

An object representing on of *Pixmap*, *DisplayList*

Type: instance

__init__ (*self*, *ts*, *tp*)
Constructor

ts
A *TextSheet* object.

Type: instance

tp
A *TextPage* object.

Type: instance

DisplayList

DisplayList is a list containing drawing commands (text, images, etc.). The intent is two-fold: as a caching-mechanism to reduce parsing of a page, and to be used as a data structure in multi-threading 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 calling run().

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*)

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

`Document` class, constructor `fitz.Document(filename)`. This will also open the document specified as `filename`. Returns a `Document` object.

Methods and Attributes

authenticate()	Decrypts the document.
loadPage()	Reads a page.

save()	Saves a copy of the document.
ToC()	Creates a table of contents.
close()	Closes the document.
outline	First *Outline* item.
needsPass	Is document is encrypted?
pageCount	The document's number of pages.
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

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 `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 minute (`'mm'`, attention: enclosed in apostrophies!). For example, a Venezuelan value might look like this `D:20150415131602-04'30'`, which means 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

IRect

IRect is a bounding box similar to *Rect*, except that all corner coordinates are rounded to integer coordinates.

Seealso
Rect

Attributes

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

Class API

class IRect
__init__ (*self*, *x0=0*, *y0=0*, *x1=0*, *y1=0*)
Constructor. An empty `IRect` will be built if no arguments given.
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 chained to each other, starting from an initial link which is accessible by the `Page.loadLinks()` method.

Methods

rect()	Clickable area in untransformed coordinates.
get_kind()	Kind of link destination.
get_uri()	UTF-8 encoded URI to launch.
get_page()	Target page number of an internal link
get_page_flags()	Validity and meaning of an internal link.
get_page_lt()	Top left corner of target rectangle.
get_page_rb()	Bottom right corner of target rectangle.

Class API

class Link
rect (*self*)
Get the hot zone. The area that can be clicked in untransformed coordinates.

Return type: *Rect*

get_kind (*self*)
Get the kind of link destination.

Return type: fz_link_kind

get_uri (*self*)
Get a UTF-8 encoded URI to launch. Return `None` if no URI is avaliable.

Return type: string

get_page (*self*)
Get the target page number of a internal link

Return type: int

get_page_flags (*self*)
Get the validity and meaning of a internal link, see fz_link_flag. The flags is a bitfield consisting of fz_link_flag describing the validity and meaning of the different parts of lr and rb. Link destinations are constructed (as far as possible) so that lt and rb can be treated as a bounding box, though the validity flags indicate which of the values was actually specified in the file.

Seealso
get_page_lt(), get_page_rb()

Return type: int

get_page_lt (*self*)
Get the top left corner of the destination bounding box.

> ***Warning***
>
> For the returning point, (0, 0) is in the bottom left corner, which is different from *Rect* and *IRect*.

Return type: *Point*

get_page_rb (*self*)

Get the bottom right corner of the destination bounding box.

> ***Warning***
>
> For the returning point, (0, 0) is in the bottom left corner, which is different from *Rect* and *IRect*.

Return type: *Point*

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

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

Attributes

a	Matrix entry at (1, 1)
b	Matrix entry at (1, 2)
c	Matrix entry at (2, 1)
d	Matrix entry at (2, 2)
e	Matrix entry at (3, 1)
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. An identity matrix will be built if no arguments given.

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)

Type: float

b

Matrix entry at (1, 2)

Type: float

c

Matrix entry at (2, 1)

Type: float

d

Matrix entry at (2, 2)

Type: float

e

Matrix entry at (3, 1)

Type: float

f

Matrix entry at (3, 2)

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

down	Next item downwards
next	Next item same level
dest	Destination, contains sub-properties.

title	Title (UTF-8 string)
saveText()	Prints a conventional table of contents to a file.
saveXML()	Prints an XML-like table of contents to a file.

Properties / Methods API

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*

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.

Return type: *Outline*

get_page (*self*)
Get the page number of the destination.

Return type: int

get_page_flags (*self*)
Get the validity and meaning of the destination link, see fz_link_flag.

The flags is a bitfield consisting of fz_link_flag describing the validity and meaning of the different parts of lr and rb. Link destinations are constructed (as far as possible) so that lt and rb can be treated as a bounding box, though the validity flags indicate which of the values was actually specified in the file.

Seealso
Link.get_page_flags()

Return type: int

get_page_lt (*self*)
Get the top left corner of the destination bounding box.

> ***Warning***
>
> For the returning point, (0, 0) is in the bottom left corner, which is different from *Rect* and *IRect*.

Return type: *Point*

get_page_rb (*self*)
Get the bottom right corner of the destination bounding box.

> ***Warning***
>
> For the returning point, (0, 0) is in the bottom left corner, which is different from *Rect* and *IRect*.

Return type: *Point*

get_title (*self*)
Get title of outline item using UTF-8 encoding.

Return type: string

Page

Page interface, created by Document.loadPage().

Methods

bound()	Page size before transformation.
loadLinks()	Get all the links in a 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 scale_matrix() and rotate_matrix(). Set to fz_identity if no transformation is desired.
- **cookie** (*Cookie*) -- Communication mechanism between caller and library rendering the page. Intended for multi-threaded applications, while single-threaded applications set cookie to `None`. The caller may abort an ongoing rendering of a page. Cookie also communicates progress information back to the caller. The fields inside cookie are continually updated while the page is rendering.

Pixmap

Pixmaps represent a set of pixels for a 2 dimensional region of a plane. Each pixel has n components per pixel, the last of which is always alpha. The data is in premultiplied alpha when rendering, but non-premultiplied for colorspace conversions and rescaling.

Methods

clearWith()	Clears a pixmap (with given value).
writePNG()	Saves a pixmap as a png file.
samples	The components data for all pixels.
invertIRect()	Invert the pixels of a given bounding box.
interpolate()	unknown
h()	Height of the region in pixels.
w()	Width of the region in pixels
x()	X-coordinate of top-left corner of pixmap
y()	Y-coordinate of top-left corner of pixmap
n()	Number of components per pixel
xres()	Resolution in X-direction
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 component sample for each pixel in the pixmap will be set to this value, while alpha will always be set to 255 (non-transparent). If not given, all components of all pixels in a pixmap will be set to 0.

samples

The components data for all pixels. Samples is a block of memory width * height * n bytes of memory in which the components are stored. The first n bytes are components 0 to n-1 for the pixel at (x,y). Each successive n bytes gives another pixel in scanline order. Subsequent scanlines follow on with no padding. E.g. for RGBA this means, that each pixel ist described by the four bytes r, g, b, alpha.

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 the pixels in *IRect*. All components are inverted(except alpha, which is unchanged).

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

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

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

interpolate ()
Save a pixmap as a png.

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

Point

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

Methods

__init__()	Constructor.

Attributes

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

Class API

class Point
__init__ (*self*, *x=0*, *y=0*)
Constructor.
x

Type: float

y

Type: float

Rect

`Rect` is a rectangle represented by its top left and its bottom right corner coordinates: ((x0, y0), (x1, y1)).

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

Methods

round()	creates an *IRect* for `Rect` (smallest pixel area containing it)
transform()	transform rectangle with a *Matrix*

Attributes

height	rectangle height
width	rectangle width
x0	X-coordinate of the top left corner.
y0	Y-coordinate of the top left corner.
x1	X-coordinate of the bottom right corner.
y1	Y-coordinate of the bottom right corner.

Class API

class Rect

__init__ (*self*, *x0=0*, *y0=0*, *x1=0*, *y1=0*)

Constructor. An empty rectangle will be built if no arguments given.

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 all the text in a page. Create a new TextPage by Context.new_text_page().

Methods

extractText()	Extract the text from the `TextPage` object.
search()	Search for a string in the current 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*, *s*, *ic*)

Search string `s`.

Parameters:

- **s** (*string*) --
- **ic** (*int*) -- Ignore case or not.

Return type: list

Returns: A python list. Each element of the list is a list of *IRect* (without transformation) where `s` is found in the page. An empty list is returned if `s` not found.

TextSheet

TextSheet contains a list of distinct text styles used on a page (or a series of pages). Create a new TextSheet by fitz.TextSheet().

Functions

All the module level functions are listed here.

Function Summary

R | S | T

R

- rotate_matrix()

S

- scale_matrix()
- shear_matrix()

T

- translate_matrix()

Function Description

rotate_matrix (*degrees*)
Create a rotation matrix.
The returned matrix is of the form [cos(deg) sin(deg) -sin(deg) cos(deg) 0 0].

Parameters: **degrees** (*float*) -- Degrees of counter clockwise rotation. Values less than zero and greater than 360 are handled as expected.

Return type: *Matrix*

---

scale_matrix (*sx*, *sy*)
Create a scaling matrix.
The returned matrix is of the form [sx 0 0 sy 0 0].

Parameters:
- **sx** (*float*) -- Scaling factors along the X-axes.
- **sy** (*float*) -- Scaling factors along the Y-axes.

Return type: *Matrix*

---

shear_matrix (*sx*, *sy*)
Create a shearing matrix.
The returned matrix is of the form [1 sy sx 1 0 0].

Parameters:
- **sx** (*float*) -- Shearing factors.
- **sy** (*float*) -- Shearing factors.

Return type: *Matrix*

---

translate_matrix (*tx*, *ty*)
Create a translation matrix.
The returned matrix is of the form [1 0 0 1 tx ty].

Parameters:
- **tx** (*float*) -- Translation distances along the X-axes.
- **ty** (*float*) -- Translation distances along the Y-axes.

Return type: *Matrix*

Constants and enums

Constants and enums are listed here.

Constants

fz_device_bgr	obsolete
fz_device_cmyk	obsolete
fz_device_gray	obsolete
fz_device_rgb	obsolete
fz_expand_images	obsolete
fz_expand_fonts	obsolete

fz_expand_all	obsolete
fz_empty_bbox	obsolete
fz_empty_rect	obsolete
fz_identity	obsolete
fz_infinite_bbox	obsolete
fz_infinite_rect	obsolete
fz_unit_bbox	obsolete
fz_unit_rect	obsolete

Enums

fz_link_kind	
FZ_LINK_GOTO	Points to some place in this document.
FZ_LINK_GOTOR	Points to another document.
FZ_LINK_LAUNCH	Launches a file.
FZ_LINK_NAMED	Performs some action.
FZ_LINK_NONE	No destination.
FZ_LINK_URI	Points to an URI.

---

fz_link_flag	
fz_link_flag_l_valid	lt.x is valid
fz_link_flag_t_valid	lt.y is valid
fz_link_flag_r_valid	rb.x is valid
fz_link_flag_b_valid	rb.y is valid
fz_link_flag_fit_h	Fit horizontally
fz_link_flag_fit_v	Fit vertically
fz_link_flag_r_is_zoom	rb.x is actually a zoom figure

Indices and tables

- *genindex*
- *modindex*
- *search*

Index

(Rect attribute)

x1 (IRect attribute)

(Rect attribute)

xres (Pixmap attribute)

Y

y (Pixmap attribute)

(Point attribute)

y0 (IRect attribute)

(Rect attribute)

y1 (IRect attribute)

(Rect attribute)

yres (Pixmap attribute)