PyPDF2

Mathieu Fenniak

Jun 06, 2022

USER GUIDE

PyPDF2 is a free and open source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. PyPDF2 can retrieve text and metadata from PDFs as well.

You can contribute to PyPDF2 on Github.

CHAPTER

ONE

INSTALLATION

There are several ways to install PyPDF2. The most common option is to use pip.

1.1 pip

PyPDF2 requires Python 3.6+ to run.

Typically Python comes with `pip`, a package installer. Using it you can install PyPDF2:

```
pip install PyPDF2
```

If you are not a super-user (a system adminstrator / root), you can also just install PyPDF2 for your current user:

```
pip install --user PyPDF2
```

1.2 Anaconda

Anaconda users can install PyPDF2 via conda-forge.

1.3 Development Version

In case you want to use the current version under development:

```
pip install git+https://github.com/py-pdf/PyPDF2.git
```

CHAPTER

TWO

ROBUSTNESS AND STRICT=FALSE

PDF is specified in various versions. The specification of PDF 1.7 has 978 pages. This length makes it hard to get everything right. As a consequence, a lot of PDF files are not strictly following the specification.

If a PDF file does not follow the specification, it is not always possible to be certain what the intended effect would be. Think of the following broken Python code as an example:

```
# Broken
function (foo, bar):

# Potentially intendet:
def function(foo, bar):
    ...

# Also possible:
function = (foo, bar)
```

Writing a parser you can go two paths: Either you try to be forgiving and try to figure out what the user intended, or you are strict and just tell the user that they should fix their stuff.

PyPDF2 gives you the option to be strict or not.

PyPDF2 has three core objects and all of them have a `strict` parameter:

- `PdfReader`
- `PdfWriter`
- `PdfMerger`

Choosing `strict=True` means that PyPDF2 will raise an exception if a PDF does not follow the specification.

Choosing `strict=False` means that PyPDF2 will try to be forgiving and do something reasonable, but it will log a warning message. It is a best-effort approach.

CHAPTER

THREE

METADATA

3.1 Reading metadata

```
from PyPDF2 import PdfReader

reader = PdfReader("example.pdf")

meta = reader.metadata

print(len(reader.pages))

# All of the following could be None!
print(meta.author)
print(meta.creator)
print(meta.producer)
print(meta.subject)
print(meta.title)
```

3.2 Writing metadata

```
from PyPDF2 import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Add the metadata
writer.add_metadata(
    {
        "/Author": "Martin",
        "/Producer": "Libre Writer",
    }
)

# Save the new PDF to a file
```

(continues on next page)

(continued from previous page)

```
with open("meta-pdf.pdf", "wb") as f:
    writer.write(f)
```

CHAPTER

FOUR

EXTRACT TEXT FROM A PDF

You can extract text from a PDF like this:

```
from PyPDF2 import PdfReader

reader = PdfReader("example.pdf")
page = reader.pages[0]
print(page.extract_text())
```

4.1 Why Text Extraction is hard

Extracting text from a PDF can be pretty tricky. In several cases there is no clear answer what the expected result should look like:

1. **Paragraphs**: Should the text of a paragraph have line breaks at the same places where the original PDF had them or should it rather be one block of text?
2. **Page numbers**: Should they be included in the extract?
3. **Headers and Footers**: Similar to page numbers - should they be extracted?
4. **Outlines**: Should outlines be extracted at all?
5. **Formatting**: If text is **bold** or *italic*, should it be included in the output?
6. **Tables**: Should the text extraction skip tables? Should it extract just the text? Should the borders be shown in some Markdown-like way or should the structure be present e.g. as an HTML table? How would you deal with merged cells?
7. **Captions**: Should image and table captions be included?
8. **Ligatures**: The Unicode symbol U+FB00 is a single symbol for two lowercase letters 'f'. Should that be parsed as the Unicode symbol '' or as two ASCII symbols 'ff'?
9. **SVG images**: Should the text parts be extracted?
10. **Mathematical Formulas**: Should they be extracted? Formulas have indices, and nested fractions.
11. **Whitespace characters**: How many new lines should be extracted for 3cm of vertical whitespace? How many spaces should be extracted if there is 3cm of horizontal whitespace? When would you extract tabs and when spaces?
12. **Footnotes**: When the text of multiple pages is extracted, where should footnotes be shown?
13. **Hyperlinks and Metadata**: Should it be extracted at all? Where should it be placed in which format?

14. **Linearization**: Assume you have a floating figure in between a paragraph. Do you first finish the paragraph or do you put the figure text in between?

Then there are issues where most people would agree on the correct output, but the way PDF stores information just makes it hard to achieve that:

1. **Tables**: Typically, tables are just absolutely positioned text. In the worst case, ever single letter could be absolutely positioned. That makes it hard to tell where columns / rows are.
2. **Images**: Sometimes PDFs do not contain the text as it's displayed, but instead an image. You notice that when you cannot copy the text. Then there are PDF files that contain an image and a text layer in the background. That typically happens when a document was scanned. Although the scanning software (OCR) is pretty good today, it still fails once in a while. PyPDF2 is no OCR software; it will not be able to detect those failures. PyPDF2 will also never be able to extract text from images.

And finally there are issues that PyPDF2 will deal with. If you find such a text extraction bug, please share the PDF with us so we can work on it!

CHAPTER

FIVE

ENCRYPTION AND DECRYPTION OF PDFS

5.1 Encrypt

Add a password to a PDF (encrypt it):

```
from PyPDF2 import PdfReader, PdfWriter

reader = PdfReader("example.pdf")
writer = PdfWriter()

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Add a password to the new PDF
writer.encrypt("my-secret-password")

# Save the new PDF to a file
with open("encrypted-pdf.pdf", "wb") as f:
    writer.write(f)
```

5.2 Decrypt

Remove the password from a PDF (decrypt it):

```
from PyPDF2 import PdfReader, PdfWriter

reader = PdfReader("encrypted-pdf.pdf")
writer = PdfWriter()

if reader.is_encrypted:
    reader.decrypt("my-secret-password")

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Save the new PDF to a file
```

(continues on next page)

(continued from previous page)

```
with open("decrypted-pdf.pdf", "wb") as f:
    writer.write(f)
```

CHAPTER

SIX

MERGING PDF FILES

6.1 Basic Example

```
from PyPDF2 import PdfMerger

merger = PdfMerger()

for pdf in ["file1.pdf", "file2.pdf", "file3.pdf"]:
    merger.append(pdf)

merger.write("merged-pdf.pdf")
merger.close()
```

For more details, see an excellent answer on StackOverflow by Paul Rooney.

6.2 Showing more merging options

```
from PyPDF2 import PdfMerger

merger = PdfMerger()

input1 = open("document1.pdf", "rb")
input2 = open("document2.pdf", "rb")
input3 = open("document3.pdf", "rb")

# add the first 3 pages of input1 document to output
merger.append(fileobj=input1, pages=(0, 3))

# insert the first page of input2 into the output beginning after the second page
merger.merge(position=2, fileobj=input2, pages=(0, 1))

# append entire input3 document to the end of the output document
merger.append(input3)

# Write to an output PDF document
output = open("document-output.pdf", "wb")
merger.write(output)
```

(continues on next page)

(continued from previous page)

```
# Close File Descriptors
merger.close()
output.close()
```

CHAPTER

SEVEN

CROPPING AND TRANSFORMING PDFS

```
from PyPDF2 import PdfWriter, PdfReader

reader = PdfReader("example.pdf")
writer = PdfWriter()

# add page 1 from reader to output document, unchanged:
writer.add_page(reader.pages[0])

# add page 2 from reader, but rotated clockwise 90 degrees:
writer.add_page(reader.pages[1].rotate(90))

# add page 3 from reader, but crop it to half size:
page3 = reader.pages[2]
page3.mediabox.upper_right = (
    page3.mediabox.right / 2,
    page3.mediabox.top / 2,
)
writer.add_page(page3)

# add some Javascript to launch the print window on opening this PDF.
# the password dialog may prevent the print dialog from being shown,
# comment the the encription lines, if that's the case, to try this out:
writer.add_js("this.print({bUI:true,bSilent:false,bShrinkToFit:true});")

# write to document-output.pdf
with open("PyPDF2-output.pdf", "wb") as fp:
    writer.write(fp)
```

7.1 Page rotation

The most typical rotation is a clockwise rotation of the page by multiples of 90 degrees. That is done when the orientation of the page is wrong. You can do that with the `rotate` method of the `PageObject` class:

```
from PyPDF2 import PdfWriter, PdfReader

reader = PdfReader("input.pdf")
writer = PdfWriter()
```

(continues on next page)

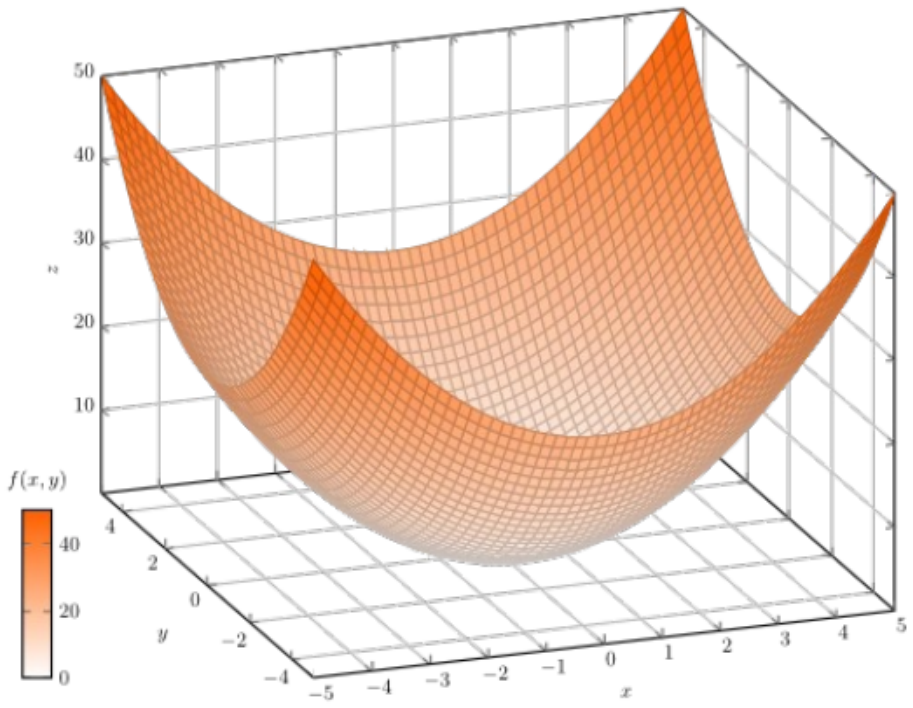
(continued from previous page)

```
writer.add_page(reader.pages[0])
writer.pages[0].rotate(90)

with open("output.pdf", "wb") as fp:
    writer.write(fp)
```

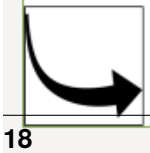
The rotate method is typically preferred over the `page.add_transformation(Transformation().rotate())` method, because `rotate` will ensure that the page is still in the mediabox / cropbox. The transformation object operates on the coordinates of the pages contents and does not change the mediabox or cropbox.

7.2 Plain Merge

Top Left Centered Top Right



Bottom Left Centered Bottom Right

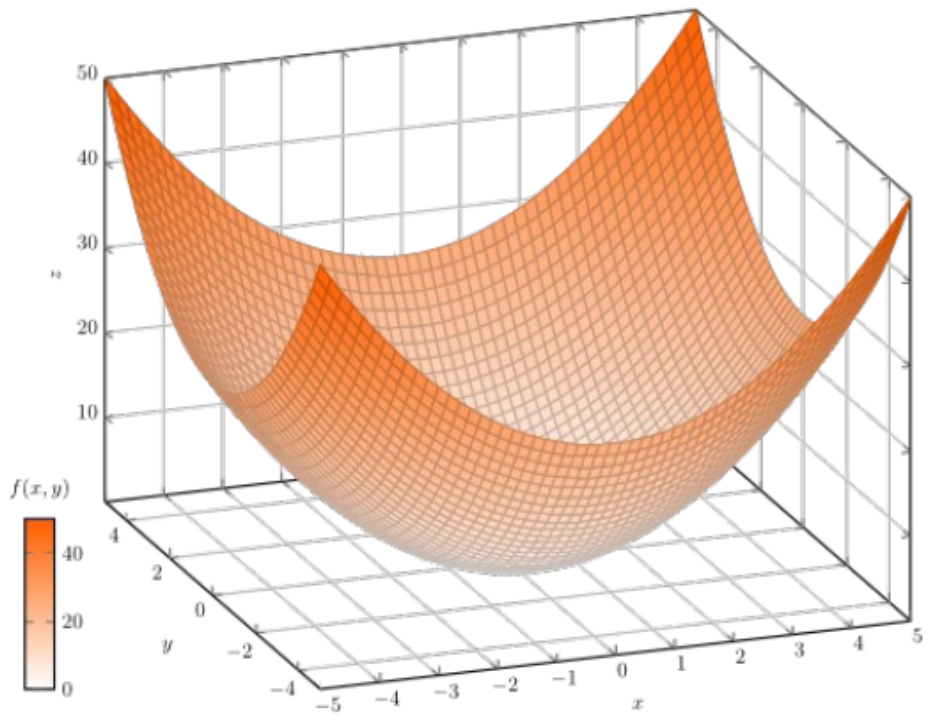


is the result of

```
from PyPDF2 import PdfReader, PdfWriter, Transformation

# Get the data
reader_base = PdfReader("labeled-edges-center-image.pdf")
page_base = reader_base.pages[0]

reader = PdfReader("box.pdf")
page_box = reader.pages[0]

page_base.merge_page(page_box)

# Write the result back
writer = PdfWriter()
writer.add_page(page_base)
with open("merged-foo.pdf", "wb") as fp:
    writer.write(fp)
```

7.3 Merge with Rotation

Top Left Centered Top Right



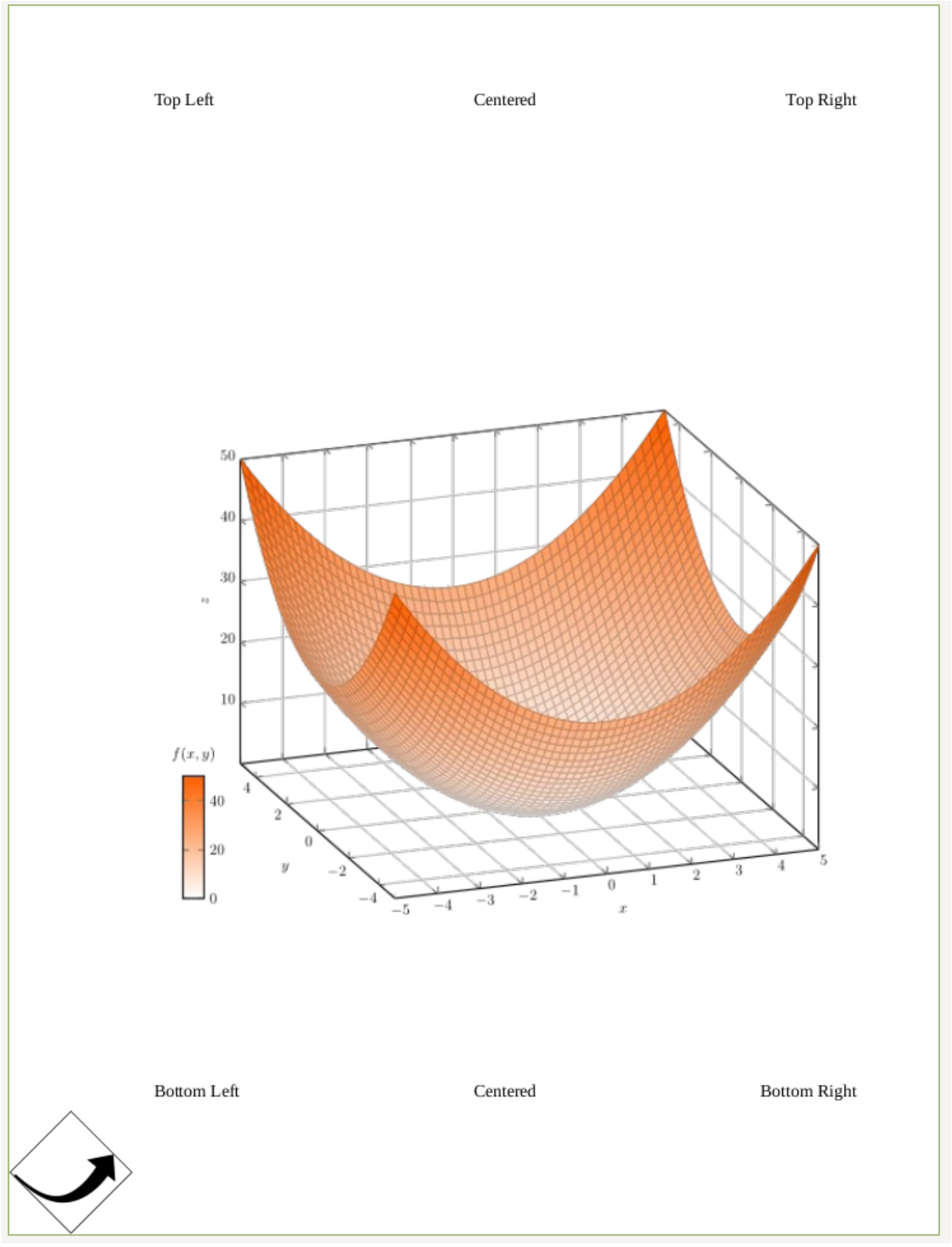
Bottom Left Centered Bottom Right

```
from PyPDF2 import PdfReader, PdfWriter, Transformation

# Get the data
reader_base = PdfReader("labeled-edges-center-image.pdf")
page_base = reader_base.pages[0]

reader = PdfReader("box.pdf")
page_box = reader.pages[0]

# Apply the transformation
transformation = Transformation().rotate(45)
page_box.add_transformation(transformation)
page_base.merge_page(page_box)

# Write the result back
writer = PdfWriter()
writer.add_page(page_base)
with open("merged-foo.pdf", "wb") as fp:
    writer.write(fp)
```

If you add the expand parameter:

```
transformation = Transformation().rotate(45)
page_box.add_transformation(transformation)
page_base.merge_page(page_box)
```

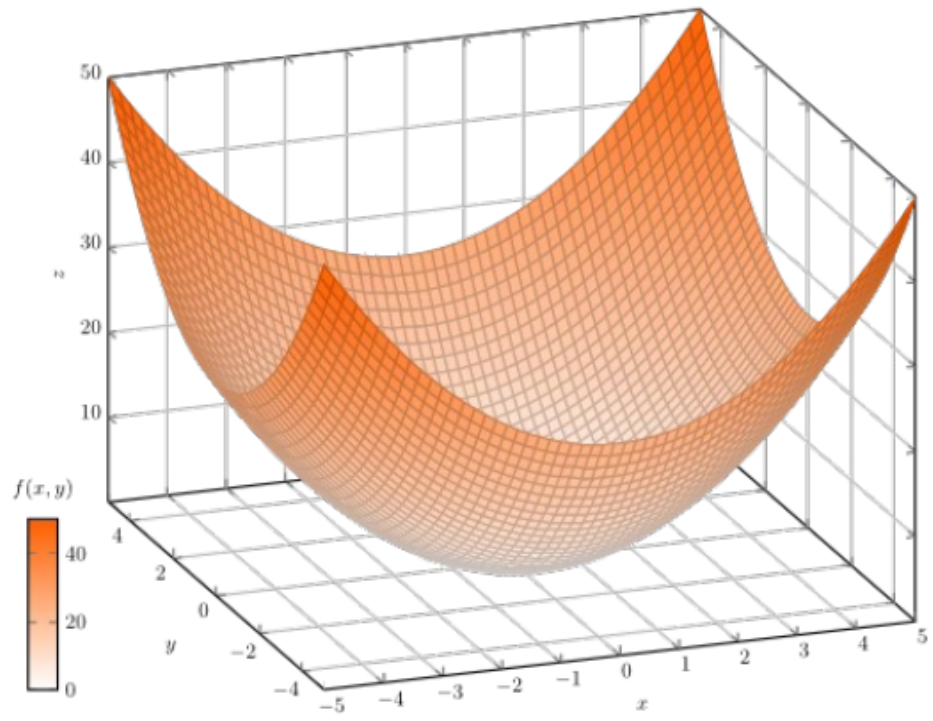
you get:



Alternatively, you can move the merged image a bit to the right by using

```
op = Transformation().rotate(45).translate(tx=50)
```

Top Left Centered Top Right



Bottom Left Centered Bottom Right

CHAPTER

EIGHT

ADDING A WATERMARK TO A PDF

```
from PyPDF2 import PdfWriter, PdfReader


# Read the watermark
watermark = PdfReader("watermark.pdf")

# Read the page without watermark
reader = PdfReader("example.pdf")
page = reader.pages[0]

# Add the watermark to the page
page.merge_page(watermark.pages[0])

# Add the page to the writer
writer = PdfWriter()
writer.add_page(page)

# finally, write the new document with a watermark
with open("PyPDF2-output.pdf", "wb") as fp:
    output.write(fp)
```

CHAPTER

NINE

READING PDF ANNOTATIONS

PDF 1.7 defines 25 different annotation types:

- Text
- Link
- FreeText
- Line, Square, Circle, Polygon, PolyLine, Highlight, Underline, Squiggly, StrikeOut
- Stamp, Caret, Ink
- Popup
- FileAttachment
- Sound, Movie
- Widget, Screen
- PrinterMark
- TrapNet
- Watermark
- 3D

Reading the most common ones is described here.

9.1 Text

```
from PyPDF2 import PdfReader

reader = PdfReader("example.pdf")

for page in reader.pages:
    if "/Annots" in page:
        for annot in page["/Annots"]:
            subtype = annot.get_object()["/Subtype"]
            if subtype == "/Text":
                print(annot.get_object()["/Contents"])
```

9.2 Highlights

```
from PyPDF2 import PdfReader

reader = PdfReader("commented.pdf")

for page in reader.pages:
    if "/Annots" in page:
        for annot in page["/Annots"]:
            subtype = annot.get_object()["/Subtype"]
            if subtype == "/Highlight":
                coords = annot.get_object()["/QuadPoints"]
                x1, y1, x2, y2, x3, y3, x4, y4 = coords
```

9.3 Attachments

```
from PyPDF2 import PdfReader

reader = PdfReader("example.pdf")

attachments = {}
for page in reader.pages:
    if "/Annots" in page:
        for annotation in page["/Annots"]:
            subtype = annot.get_object()["/Subtype"]
            if subtype == "/FileAttachment":
                fileobj = annotobj["/FS"]
                attachments[fileobj["/F"]] = fileobj["/EF"]["/F"].get_data()
```

CHAPTER

TEN

ADDING PDF ANNOTATIONS

10.1 Attachments

```
from PyPDF2 import PdfWriter

writer = PdfWriter()
writer.add_blank_page(width=200, height=200)

data = b"any bytes - typically read from a file"
writer.add_attachment("smile.png", data)

with open("output.pdf", "wb") as output_stream:
    writer.write(output_stream)
```

CHAPTER

ELEVEN

INTERACTIONS WITH PDF FORMS

11.1 Reading form fields

```
from PyPDF2 import PdfReader

reader = PdfReader("form.pdf")
fields = reader.get_form_text_fields()
fields == {"key": "value", "key2": "value2"}
```

11.2 Filling out forms

```
from PyPDF2 import PdfReader, PdfWriter

reader = PdfReader("form.pdf")
writer = PdfWriter()

page = reader.pages[0]
fields = reader.get_fields()

writer.add_page(page)

writer.update_page_form_field_values(
    writer.pages[0], {"fieldname": "some filled in text"}
)

# write "output" to PyPDF2-output.pdf
with open("filled-out.pdf", "wb") as output_stream:
    writer.write(output_stream)
```

CHAPTER

TWELVE

STREAMING DATA WITH PYPDF2

In some cases you might want to avoid saving things explicitly as a file to disk, e.g. when you want to store the PDF in a database or AWS S3.

PyPDF2 supports streaming data to a file-like object and here is how.

```
from io import BytesIO

# Prepare example
with open("example.pdf", "rb") as fh:
    bytes_stream = BytesIO(fh.read())

# Read from bytes_stream
reader = PdfReader(bytes_stream)

# Write to bytes_stream
writer = PdfWriter()
with BytesIO() as bytes_stream:
    writer.write(bytes_stream)
```

12.1 Writing a PDF directly to AWS S3

Suppose you want to manipulate a PDF and write it directly to AWS S3 without having to write the document to a file first. We have the original PDF in `raw_bytes_data` as `bytes` and want to set `my-secret-password`:

```
from io import BytesIO

import boto3
from PyPDF2 import PdfReader, PdfWriter


reader = PdfReader(BytesIO(raw_bytes_data))
writer = PdfWriter()

# Add all pages to the writer
for page in reader.pages:
    writer.add_page(page)

# Add a password to the new PDF
writer.encrypt("my-secret-password")
```

(continues on next page)

(continued from previous page)

```
# Save the new PDF to a file
with BytesIO() as bytes_stream:
    writer.write(bytes_stream)
    bytes_stream.seek(0)
    s3 = boto3.client("s3")
    s3.write_get_object_response(
        Body=bytes_stream, RequestRoute=request_route, RequestToken=request_token
    )
```

CHAPTER

THIRTEEN

REDUCE PDF SIZE

There are multiple ways to reduce the size of a given PDF file. The easiest one is to remove content (e.g. images) or pages.

13.1 Remove images

```
import PyPDF2

reader = PyPDF2.PdfReader("example.pdf")
writer = PyPDF2.PdfWriter()

for page in reader.pages:
    writer.add_page(page)

writer.remove_images()

with open("out.pdf", "wb") as f:
    writer.write(f)
```

13.2 Compression

```
import PyPDF2

reader = PyPDF2.PdfReader("example.pdf")
writer = PyPDF2.PdfWriter()

for page in reader.pages:
    page.compress_content_streams()
    writer.add_page(page)

with open("out.pdf", "wb") as f:
    writer.write(f)
```

CHAPTER

FOURTEEN

PDF VERSION SUPPORT

PDF comes in the following versions:

- 1993: 1.0
- 1994: 1.1
- 1996: 1.2
- 1999: 1.3
- 2001: 1.4
- 2003: 1.5
- 2004: 1.6
- 2006 - 2012: 1.7, ISO 32000-1:2008
- 2017: 2.0

The general format didn't change, but new features got added. It can be that PyPDF2 can do the operations you want on PDF 2.0 files without fully supporting all features of PDF 2.0.

14.1 PDF Feature Support by PyPDF2

Feature	PDF-Version	PyPDF2 Support
Transparent Graphics	1.4	?
CMaps	1.4	#201, #464, #805
Object Streams	1.5	?
Cross-reference Streams	1.5	?
Optional Content Groups (OCGs) - Layers	1.5	?
Content Stream Compression	1.5	?
AES Encryption	1.6	#749

See History of PDF for more features.

CHAPTER

FIFTEEN

THE PDFREADER CLASS

class PyPDF2.**PdfReader**(*stream: Union[str, _io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*, *strict: bool = False*, *password: Union[None, str, bytes] = None*)

Bases: `object`

Initialize a PdfReader object.

This operation can take some time, as the PDF stream's cross-reference tables are read into memory.

Parameters

- **stream** – A File object or an object that supports the standard read and seek methods similar to a File object. Could also be a string representing a path to a PDF file.
- **strict** (*bool*) – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to `False`.
- **password** (*None/str/bytes*) – Decrypt PDF file at initialization. If the password is None, the file will not be decrypted. Defaults to `None`

cacheGetIndirectObject(*generation: int*, *idnum: int*) → Optional[PyPDF2.generic.PdfObject]

Deprecated since version 1.28.0: Use *cache_get_indirect_object()* instead.

cacheIndirectObject(*generation: int*, *idnum: int*, *obj: Optional[PyPDF2.generic.PdfObject]*) → Optional[PyPDF2.generic.PdfObject]

Deprecated since version 1.28.0: Use *cache_indirect_object()* instead.

cache_get_indirect_object(*generation: int*, *idnum: int*) → Optional[PyPDF2.generic.PdfObject]

cache_indirect_object(*generation: int*, *idnum: int*, *obj: Optional[PyPDF2.generic.PdfObject]*) → Optional[PyPDF2.generic.PdfObject]

decode_permissions(*permissions_code: int*) → Dict[str, bool]

decrypt(*password: Union[str, bytes]*) → int

When using an encrypted / secured PDF file with the PDF Standard encryption handler, this function will allow the file to be decrypted. It checks the given password against the document's user password and owner password, and then stores the resulting decryption key if either password is correct.

It does not matter which password was matched. Both passwords provide the correct decryption key that will allow the document to be used with this library.

Parameters **password** (*str*) – The password to match.

Returns `0` if the password failed, `1` if the password matched the user password, and `2` if the password matched the owner password.

Return type int

Raises `NotImplementedError` – if document uses an unsupported encryption method.

property documentInfo: Optional[*PyPDF2._reader.DocumentInformation*]

Deprecated since version 1.28.0.

Use the attribute *metadata* instead.

getDestinationPageNumber(*destination:* PyPDF2.generic.Destination) → int

Deprecated since version 1.28.0: Use *get_destination_page_number()* instead.

getDocumentInfo() → Optional[*PyPDF2._reader.DocumentInformation*]

Deprecated since version 1.28.0: Use the attribute *metadata* instead.

getFields(*tree: Optional[PyPDF2.generic.TreeObject] = None*, *retval: Optional[Dict[Any, Any]] = None*, *fileobj: Optional[Any] = None*) → Optional[Dict[str, Any]]

Deprecated since version 1.28.0: Use *get_fields()* instead.

getFormTextFields() → Dict[str, Any]

Deprecated since version 1.28.0: Use *get_form_text_fields()* instead.

getIsEncrypted() → bool

Deprecated since version 1.28.0: Use *is_encrypted* instead.

getNamedDestinations(*tree: Optional[PyPDF2.generic.TreeObject] = None*, *retval: Optional[Any] = None*) → Dict[str, Any]

Deprecated since version 1.28.0: Use *named_destinations* instead.

getNumPages() → int

Deprecated since version 1.28.0: Use `len(reader.pages)` instead.

getObject(*indirectReference: PyPDF2.generic.IndirectObject*) → Optional[PyPDF2.generic.PdfObject]

Deprecated since version 1.28.0: Use *get_object()* instead.

getOutlines(*node: Optional[PyPDF2.generic.DictionaryObject] = None*, *outlines: Optional[Any] = None*) → List[Union[*PyPDF2.generic.Destination*, List[Union[*PyPDF2.generic.Destination*, List[*PyPDF2.generic.Destination*]]]]]

Deprecated since version 1.28.0: Use *outlines* instead.

getPage(*pageNumber: int*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use `reader.pages[pageNumber]` instead.

getPageLayout() → Optional[str]

Deprecated since version 1.28.0: Use *page_layout* instead.

getPageMode() → Optional[typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]]

Deprecated since version 1.28.0: Use *page_mode* instead.

getPageNumber(*page:* PyPDF2._page.PageObject) → int

Deprecated since version 1.28.0: Use *get_page_number()* instead.

getXmpMetadata() → Optional[*PyPDF2.xmp.XmpInformation*]

Deprecated since version 1.28.0: Use the attribute *xmp_metadata* instead.

get_destination_page_number(*destination:* PyPDF2.generic.Destination) → int

Retrieve page number of a given Destination object.

Parameters `destination` (`Destination`) – The destination to get page number.

Returns the page number or -1 if page not found

Return type int

get_fields(*tree: Optional[PyPDF2.generic.TreeObject] = None*, *retval: Optional[Dict[Any, Any]] = None*, *fileobj: Optional[Any] = None*) → Optional[Dict[str, Any]]

Extracts field data if this PDF contains interactive form fields. The *tree* and *retval* parameters are for recursive use.

Parameters **fileobj** – A file object (usually a text file) to write a report to on all interactive form fields found.

Returns A dictionary where each key is a field name, and each value is a *Field* object. By default, the mapping name is used for keys.

Return type dict, or None if form data could not be located.

get_form_text_fields() → Dict[str, Any]

Retrieves form fields from the document with textual data (inputs, dropdowns)

get_object(*indirect_reference: PyPDF2.generic.IndirectObject*) → Optional[PyPDF2.generic.PdfObject]

get_page_number(*page:* PyPDF2._page.PageObject) → int

Retrieve page number of a given PageObject

Parameters **page** (PageObject) – The page to get page number. Should be an instance of *PageObject*

Returns the page number or -1 if page not found

Return type int

property isEncrypted: bool

Deprecated since version 1.28.0.

Use *is_encrypted* instead.

property is_encrypted: bool

Read-only boolean property showing whether this PDF file is encrypted. Note that this property, if true, will remain true even after the *decrypt()* method is called.

property metadata: Optional[*PyPDF2._reader.DocumentInformation*]

Retrieve the PDF file's document information dictionary, if it exists. Note that some PDF files use metadata streams instead of docinfo dictionaries, and these metadata streams will not be accessed by this function.

Returns the document information of this PDF file

Return type DocumentInformation or None if none exists.

property namedDestinations: Dict[str, Any]

Deprecated since version 1.28.0.

Use *named_destinations* instead.

property named_destinations: Dict[str, Any]

A read-only dictionary which maps names to *Destinations*

property numPages: int

Deprecated since version 1.28.0.

Use len(reader.pages) instead.

property outlines: List[Union[*PyPDF2.generic.Destination*, List[Union[*PyPDF2.generic.Destination*, List[*PyPDF2.generic.Destination*]]]]]

Read-only property for outlines present in the document.

Returns a nested list of *Destinations*.

property pageLayout: Optional[str]

Deprecated since version 1.28.0.

Use *page_layout* instead.

property pageMode: Optional[typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]]

Deprecated since version 1.28.0.

Use *page_mode* instead.

property page_layout: Optional[str]

Get the page layout.

Returns Page layout currently being used.

Return type `str`, `None` if not specified

Table 1: Valid `layout` values

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

property page_mode: Optional[typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]]

Get the page mode.

Returns Page mode currently being used.

Return type `str`, `None` if not specified

Table 2: Valid `mode` values

/UseNone	Do not show outlines or thumbnails panels
/UseOutlines	Show outlines (aka bookmarks) panel
/UseThumbs	Show page thumbnails panel
/FullScreen	Fullscreen view
/UseOC	Show Optional Content Group (OCG) panel
/UseAttachments	Show attachments panel

property pages: PyPDF2._page._VirtualList

Read-only property that emulates a list of `Page` objects.

read(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*) → None

readNextEndLine(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*, *limit_offset: int = 0*) → bytes

Deprecated since version 1.28.0: Use *read_next_end_line()* instead.

readObjectHeader(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*) → Tuple[int, int]

Deprecated since version 1.28.0: Use *read_object_header()* instead.

read_next_end_line(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*, *limit_offset: int = 0*) → bytes

read_object_header(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*) → Tuple[int, int]

property xmpMetadata: Optional[*PyPDF2.xmp.XmpInformation*]

Deprecated since version 1.28.0.

Use the attribute *xmp_metadata* instead.

property xmp_metadata: Optional[*PyPDF2.xmp.XmpInformation*]

XMP (Extensible Metadata Platform) data

Returns a *XmpInformation* instance that can be used to access XMP metadata from the document.

Return type *XmpInformation* or `None` if no metadata was found on the document root.

CHAPTER

SIXTEEN

THE PDFWRITER CLASS

class PyPDF2.**PdfWriter**

Bases: `object`

This class supports writing PDF files out, given pages produced by another class (typically *PdfReader*).

addAttachment(*fname: str*, *fdata: Union[str, bytes]*) → None

Deprecated since version 1.28.0: Use *add_attachment()* instead.

addBlankPage(*width: Optional[float] = None*, *height: Optional[float] = None*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use *add_blank_page()* instead.

addBookmark(*title: str, pagenum: int, parent: typing.Union[None, PyPDF2.generic.TreeObject, PyPDF2.generic.IndirectObject] = None, color: typing.Optional[typing.Tuple[float, float, float]] = None, bold: bool = False, italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: typing.List[typing.Union[PyPDF2.generic.NumberObject, PyPDF2.generic.NullObject]]*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *add_bookmark()* instead.

addBookmarkDestination(*dest:* PyPDF2._page.PageObject, *parent: Optional[PyPDF2.generic.TreeObject] = None*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *add_bookmark_destination()* instead.

addBookmarkDict(*bookmark: Union[PyPDF2.generic.Bookmark,* PyPDF2.generic.Destination*]*, *parent: Optional[PyPDF2.generic.TreeObject] = None*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *add_bookmark_dict()* instead.

addJS(*javascript: str*) → None

Deprecated since version 1.28.0: Use *add_js()* instead.

addLink(*pagenum: int, pagedest: int, rect: PyPDF2.generic.RectangleObject, border: typing.Optional[PyPDF2.generic.ArrayObject] = None, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: typing.Union[PyPDF2.generic.NumberObject, PyPDF2.generic.NullObject]*) → None

Deprecated since version 1.28.0: Use *add_link()* instead.

addMetadata(*infos: Dict[str, Any]*) → None

Deprecated since version 1.28.0: Use *add_metadata()* instead.

addNamedDestination(*title: str*, *pagenum: int*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *add_named_destination()* instead.

addNamedDestinationObject(*dest: PyPDF2.generic.PdfObject*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *add_named_destination_object()* instead.

addPage(*page:* PyPDF2._page.PageObject) → None

Deprecated since version 1.28.0: Use *add_page()* instead.

addURI(*pagenum: int*, *uri: int*, *rect:* PyPDF2.generic.RectangleObject, *border: Optional[PyPDF2.generic.ArrayObject] = None*) → None

Deprecated since version 1.28.0: Use *add_uri()* instead.

add_attachment(*filename: str*, *data: Union[str, bytes]*) → None

Embed a file inside the PDF.

Parameters

- **filename** (*str*) – The filename to display.
- **data** (*str*) – The data in the file.

Reference: https://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/PDF32000_2008.pdf Section 7.11.3

add_blank_page(*width: Optional[float] = None*, *height: Optional[float] = None*) → *PyPDF2._page.PageObject*

Append a blank page to this PDF file and returns it. If no page size is specified, use the size of the last page.

Parameters

- **width** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – The height of the new page expressed in default user space units.

Returns the newly appended page

Return type *PageObject*

Raises **PageSizeNotDefinedError** – if width and height are not defined and previous page does not exist.

add_bookmark(*title: str, pagenum: int, parent: typing.Union[None, PyPDF2.generic.TreeObject, PyPDF2.generic.IndirectObject] = None, color: typing.Optional[typing.Tuple[float, float, float]] = None, bold: bool = False, italic: bool = False, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: typing.List[typing.Union[PyPDF2.generic.NumberObject, PyPDF2.generic.NullObject]]*) → PyPDF2.generic.IndirectObject

Add a bookmark to this PDF file.

Parameters

- **title** (*str*) – Title to use for this bookmark.
- **pagenum** (*int*) – Page number this bookmark will point to.
- **parent** – A reference to a parent bookmark to create nested bookmarks.
- **color** (*tuple*) – Color of the bookmark as a red, green, blue tuple from 0.0 to 1.0
- **bold** (*bool*) – Bookmark is bold
- **italic** (*bool*) – Bookmark is italic
- **fit** (*str*) – The fit of the destination page. See *addLink()* for details.

add_bookmark_destination(*dest:* PyPDF2._page.PageObject, *parent: Optional[PyPDF2.generic.TreeObject] = None*) → PyPDF2.generic.IndirectObject

add_bookmark_dict(*bookmark: Union[PyPDF2.generic.Bookmark,* PyPDF2.generic.Destination*], parent: Optional[PyPDF2.generic.TreeObject] = None*) → PyPDF2.generic.IndirectObject

add_js(*javascript: str*) → None

Add Javascript which will launch upon opening this PDF.

Parameters **javascript** (*str*) – Your Javascript.

```
>>> output.add_js("this.print({bUI:true,bSilent:false,bShrinkToFit:true});")
# Example: This will launch the print window when the PDF is opened.
```

add_link(*pagenum: int, pagedest: int, rect: PyPDF2.generic.RectangleObject, border: typing.Optional[PyPDF2.generic.ArrayObject] = None, fit: typing_extensions.Literal[/Fit, /XYZ, /FitH, /FitV, /FitR, /FitB, /FitBH, /FitBV] = '/Fit', *args: typing.Union[PyPDF2.generic.NumberObject, PyPDF2.generic.NullObject]*) → None

Add an internal link from a rectangular area to the specified page.

Parameters

- **pagenum** (*int*) – index of the page on which to place the link.
- **pagedest** (*int*) – index of the page to which the link should go.
- **rect** – *RectangleObject* or array of four integers specifying the clickable rectangular area `[xLL, yLL, xUR, yUR]`, or string in the form `"[ xLL yLL xUR yUR ]"`.
- **border** – if provided, an array describing border-drawing properties. See the PDF spec for details. No border will be drawn if this argument is omitted.
- **fit** (*str*) – Page fit or 'zoom' option (see below). Additional arguments may need to be supplied. Passing `None` will be read as a null value for that coordinate.

Table 1: Valid `zoom` arguments (see Table 8.2 of the PDF 1.7 reference for details)

/Fit	No additional arguments
/XYZ	[left] [top] [zoomFactor]
/FitH	[top]
/FitV	[left]
/FitR	[left] [bottom] [right] [top]
/FitB	No additional arguments
/FitBH	[top]
/FitBV	[left]

add_metadata(*infos: Dict[str, Any]*) → None

Add custom metadata to the output.

Parameters **infos** (*dict*) – a Python dictionary where each key is a field and each value is your new metadata.

add_named_destination(*title: str*, *pagenum: int*) → PyPDF2.generic.IndirectObject

add_named_destination_object(*dest: PyPDF2.generic.PdfObject*) → PyPDF2.generic.IndirectObject

add_page(*page:* PyPDF2._page.PageObject) → None

Add a page to this PDF file. The page is usually acquired from a *PdfReader* instance.

Parameters **page** (PageObject) – The page to add to the document. Should be an instance of *PageObject*

add_uri(*pagenum: int*, *uri: int*, *rect:* PyPDF2.generic.RectangleObject, *border: Optional[PyPDF2.generic.ArrayObject] = None*) → None

Add an URI from a rectangular area to the specified page. This uses the basic structure of AddLink

Parameters

- **pagenum** (*int*) – index of the page on which to place the URI action.
- **uri** (*int*) – string – uri of resource to link to.
- **rect** – *RectangleObject* or array of four integers specifying the clickable rectangular area `[xLL, yLL, xUR, yUR]`, or string in the form `"[ xLL yLL xUR yUR ]"`.
- **border** – if provided, an array describing border-drawing properties. See the PDF spec for details. No border will be drawn if this argument is omitted.

REMOVED FIT/ZOOM ARG -John Mulligan

appendPagesFromReader(*reader:* PyPDF2._reader.PdfReader, *after_page_append: Optional[Callable[[PyPDF2._page.PageObject], None]] = None*) → None

Deprecated since version 1.28.0: Use *append_pages_from_reader()* instead.

append_pages_from_reader(*reader:* PyPDF2._reader.PdfReader, *after_page_append: Optional[Callable[[PyPDF2._page.PageObject], None]] = None*) → None

Copy pages from reader to writer. Includes an optional callback parameter which is invoked after pages are appended to the writer.

Parameters

- **reader** – a PdfReader object from which to copy page annotations to this writer object. The writer's annots will then be updated
- **reference)** (*writer_pageref (PDF page*) – Reference to the page appended to the writer.

Callback after_page_append (function) Callback function that is invoked after each page is appended to the writer. Callback signature:

cloneDocumentFromReader(*reader:* PyPDF2._reader.PdfReader, *after_page_append: Optional[Callable[[PyPDF2._page.PageObject], None]] = None*) → None

Deprecated since version 1.28.0: Use *clone_document_from_reader()* instead.

cloneReaderDocumentRoot(*reader:* PyPDF2._reader.PdfReader) → None

Deprecated since version 1.28.0: Use *clone_reader_document_root()* instead.

clone_document_from_reader(*reader:* PyPDF2._reader.PdfReader, *after_page_append: Optional[Callable[[PyPDF2._page.PageObject], None]] = None*) → None

Create a copy (clone) of a document from a PDF file reader

Parameters **reader** – PDF file reader instance from which the clone should be created.

Callback after_page_append (function) Callback function that is invoked after each page is appended to the writer. Signature includes a reference to the appended page (delegates to appendPagesFromReader). Callback signature:

param writer_pageref (PDF page reference) Reference to the page just appended to the document.

clone_reader_document_root(*reader:* PyPDF2._reader.PdfReader) → None

Copy the reader document root to the writer.

Parameters **reader** – PdfReader from the document root should be copied.

Callback after_page_append

encrypt(*user_pwd: str*, *owner_pwd: Optional[str] = None*, *use_128bit: bool = True*, *permissions_flag: int = -1*) → None

Encrypt this PDF file with the PDF Standard encryption handler.

Parameters

- **user_pwd** (*str*) – The "user password", which allows for opening and reading the PDF file with the restrictions provided.
- **owner_pwd** (*str*) – The "owner password", which allows for opening the PDF files without any restrictions. By default, the owner password is the same as the user password.
- **use_128bit** (*bool*) – flag as to whether to use 128bit encryption. When false, 40bit encryption will be used. By default, this flag is on.
- **permissions_flag** (*unsigned int*) – permissions as described in TABLE 3.20 of the PDF 1.7 specification. A bit value of 1 means the permission is grantend. Hence an integer value of -1 will set all flags. Bit position 3 is for printing, 4 is for modifying content, 5 and 6 control annotations, 9 for form fields, 10 for extraction of text and graphics.

getNamedDestRoot() → PyPDF2.generic.ArrayObject

Deprecated since version 1.28.0: Use *get_named_dest_root()* instead.

getNumPages() → int

Deprecated since version 1.28.0: Use `len(writer.pages)` instead.

getObject(*ido: PyPDF2.generic.IndirectObject*) → PyPDF2.generic.PdfObject

Deprecated since version 1.28.0: Use *get_object()* instead.

getOutlineRoot() → PyPDF2.generic.TreeObject

Deprecated since version 1.28.0: Use *get_outline_root()* instead.

getPage(*pageNumber: int*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use `writer.pages[page_number]` instead.

getPageLayout() → Optional[typing_extensions.Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]]

Deprecated since version 1.28.0: Use *page_layout* instead.

getPageMode() → Optional[typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]]

Deprecated since version 1.28.0: Use *page_mode* instead.

getReference(*obj: PyPDF2.generic.PdfObject*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *get_reference()* instead.

get_named_dest_root() → PyPDF2.generic.ArrayObject

get_object(*ido: PyPDF2.generic.IndirectObject*) → PyPDF2.generic.PdfObject

get_outline_root() → PyPDF2.generic.TreeObject

get_page(*pageNumber: int*) → *PyPDF2._page.PageObject*

Retrieve a page by number from this PDF file.

Parameters **pageNumber** (*int*) – The page number to retrieve (pages begin at zero)

Returns the page at the index given by *pageNumber*

Return type *PageObject*

get_reference(*obj: PyPDF2.generic.PdfObject*) → PyPDF2.generic.IndirectObject

insertBlankPage(*width: Optional[decimal.Decimal] = None*, *height: Optional[decimal.Decimal] = None*, *index: int = 0*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use *insertBlankPage()* instead.

insertPage(*page:* PyPDF2._page.PageObject, *index: int = 0*) → None

Deprecated since version 1.28.0: Use *insert_page()* instead.

insert_blank_page(*width: Optional[decimal.Decimal] = None*, *height: Optional[decimal.Decimal] = None*, *index: int = 0*) → *PyPDF2._page.PageObject*

Insert a blank page to this PDF file and returns it. If no page size is specified, use the size of the last page.

Parameters

- **width** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – The height of the new page expressed in default user space units.
- **index** (*int*) – Position to add the page.

Returns the newly appended page

Return type *PageObject*

Raises **PageSizeNotDefinedError** – if width and height are not defined and previous page does not exist.

insert_page(*page:* PyPDF2._page.PageObject, *index: int = 0*) → None

Insert a page in this PDF file. The page is usually acquired from a *PdfReader* instance.

Parameters

- **page** (PageObject) – The page to add to the document. This argument should be an instance of *PageObject*.
- **index** (*int*) – Position at which the page will be inserted.

property pageLayout: Optional[typing_extensions.Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]]

Deprecated since version 1.28.0.

Use *page_layout* instead.

property pageMode: Optional[typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]]

Deprecated since version 1.28.0.

Use *page_mode* instead.

property page_layout: Optional[typing_extensions.Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]]

Page layout property.

Table 2: Valid `layout` values

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

property page_mode: Optional[typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]]

Page mode property.

Table 3: Valid `mode` values

/UseNone	Do not show outlines or thumbnails panels
/UseOutlines	Show outlines (aka bookmarks) panel
/UseThumbs	Show page thumbnails panel
/FullScreen	Fullscreen view
/UseOC	Show Optional Content Group (OCG) panel
/UseAttachments	Show attachments panel

property pages: List[*PyPDF2._page.PageObject*]

Property that emulates a list of *PageObject*

removeImages(*ignoreByteStringObject: bool = False*) → None

Deprecated since version 1.28.0: Use *remove_images()* instead.

removeLinks() → None

Deprecated since version 1.28.0: Use *remove_links()* instead.

removeText(*ignoreByteStringObject: bool = False*) → None

Deprecated since version 1.28.0: Use *remove_text()* instead.

remove_images(*ignore_byte_string_object: bool = False*) → None

Remove images from this output.

Parameters **ignore_byte_string_object** (*bool*) – optional parameter to ignore ByteString Objects.

remove_links() → None

Remove links and annotations from this output.

remove_text(*ignore_byte_string_object: bool = False*) → None

Remove text from this output.

Parameters **ignore_byte_string_object** (*bool*) – optional parameter to ignore ByteString Objects.

setPageLayout(*layout: typing_extensions.Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]*) → None

Deprecated since version 1.28.0: Use `page_layout` instead.

setPageMode(*mode: typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]*) → None

Deprecated since version 1.28.0: Use `page_mode` instead.

set_need_appearances_writer() → None

set_page_mode(*mode: typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]*) → None

Deprecated since version 1.28.0: Use `page_mode` instead.

updatePageFormFieldValues(*page:* PyPDF2._page.PageObject, *fields: Dict[str, Any]*, *flags: int = 0*) → None

Deprecated since version 1.28.0: Use `update_page_form_field_values()` instead.

update_page_form_field_values(*page:* PyPDF2._page.PageObject, *fields: Dict[str, Any]*, *flags: int = 0*) → None

Update the form field values for a given page from a fields dictionary. Copy field texts and values from fields to page. If the field links to a parent object, add the information to the parent.

Parameters

- **page** – Page reference from PDF writer where the annotations and field data will be updated.
- **fields** – a Python dictionary of field names (/T) and text values (/V)
- **flags** – An integer (0 to 7). The first bit sets ReadOnly, the second bit sets Required, the third bit sets NoExport. See PDF Reference Table 8.70 for details.

write(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*) → None

Write the collection of pages added to this object out as a PDF file.

Parameters **stream** – An object to write the file to. The object must support the write method and the tell method, similar to a file object.

CHAPTER

SEVENTEEN

THE PDFMERGER CLASS

class PyPDF2.**PdfMerger**(*strict: bool = False*)

Bases: `object`

Initializes a `PdfMerger` object. `PdfMerger` merges multiple PDFs into a single PDF. It can concatenate, slice, insert, or any combination of the above.

See the functions *merge()* (or *append()*) and *write()* for usage information.

Parameters **strict** (*bool*) – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to `False`.

addBookmark(*title: str*, *pagenum: int*, *parent: Union[None, PyPDF2.generic.TreeObject, PyPDF2.generic.IndirectObject] = None*, *color: Optional[Tuple[float, float, float]] = None*, *bold: bool = False*, *italic: bool = False*, *fit: str = '/Fit'*, **args: Union[PyPDF2.generic.NumberObject, PyPDF2.generic.NullObject]*) → PyPDF2.generic.IndirectObject

Deprecated since version 1.28.0: Use *add_bookmark()* instead.

addMetadata(*infos: Dict[str, Any]*) → None

Deprecated since version 1.28.0: Use *add_metadata()* instead.

addNamedDestination(*title: str*, *pagenum: int*) → None

Deprecated since version 1.28.0: Use *add_named_destination()* instead.

add_bookmark(*title: str*, *pagenum: int*, *parent: Union[None, PyPDF2.generic.TreeObject, PyPDF2.generic.IndirectObject] = None*, *color: Optional[Tuple[float, float, float]] = None*, *bold: bool = False*, *italic: bool = False*, *fit: str = '/Fit'*, **args: Union[PyPDF2.generic.NumberObject, PyPDF2.generic.NullObject]*) → PyPDF2.generic.IndirectObject

Add a bookmark to this PDF file.

Parameters

- **title** (*str*) – Title to use for this bookmark.
- **pagenum** (*int*) – Page number this bookmark will point to.
- **parent** – A reference to a parent bookmark to create nested bookmarks.
- **color** (*tuple*) – Color of the bookmark as a red, green, blue tuple from 0.0 to 1.0
- **bold** (*bool*) – Bookmark is bold
- **italic** (*bool*) – Bookmark is italic
- **fit** (*str*) – The fit of the destination page. See `addLink()` for details.

add_metadata(*infos: Dict[str, Any]*) → None

Add custom metadata to the output.

Parameters **infos** (*dict*) – a Python dictionary where each key is a field and each value is your new metadata. Example: `{u'/Title': u'My title'}`

add_named_destination(*title: str*, *pagenum: int*) → None

Add a destination to the output.

Parameters

- **title** (*str*) – Title to use
- **pagenum** (*int*) – Page number this destination points at.

append(*fileobj: Union[str, _io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO, PyPDF2._reader.PdfReader]*, *bookmark: Optional[str] = None*, *pages: Union[None, PyPDF2.pagerange.PageRange, Tuple[int, int], Tuple[int, int, int]] = None*, *import_bookmarks: bool = True*) → None

Identical to the *merge()* method, but assumes you want to concatenate all pages onto the end of the file instead of specifying a position.

Parameters

- **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.
- **bookmark** (*str*) – Optionally, you may specify a bookmark to be applied at the beginning of the included file by supplying the text of the bookmark.
- **pages** – can be a *PageRange* or a `(start, stop[, step])` tuple to merge only the specified range of pages from the source document into the output document.
- **import_bookmarks** (*bool*) – You may prevent the source document's bookmarks from being imported by specifying this as `False`.

close() → None

Shuts all file descriptors (input and output) and clears all memory usage.

find_bookmark(*bookmark: Dict[str, Any]*, *root: Optional[List[Union[PyPDF2.generic.Destination, List[Union[PyPDF2.generic.Destination, List[PyPDF2.generic.Destination]]]]]] = None*) → Optional[List[int]]

merge(*position: int*, *fileobj: Union[str, _io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO, PyPDF2._reader.PdfReader]*, *bookmark: Optional[str] = None*, *pages: Optional[Union[str, PyPDF2.pagerange.PageRange, Tuple[int, int], Tuple[int, int, int]]] = None*, *import_bookmarks: bool = True*) → None

Merges the pages from the given file into the output file at the specified page number.

Parameters

- **position** (*int*) – The *page number* to insert this file. File will be inserted after the given number.
- **fileobj** – A File Object or an object that supports the standard read and seek methods similar to a File Object. Could also be a string representing a path to a PDF file.
- **bookmark** (*str*) – Optionally, you may specify a bookmark to be applied at the beginning of the included file by supplying the text of the bookmark.
- **pages** – can be a *PageRange* or a `(start, stop[, step])` tuple to merge only the specified range of pages from the source document into the output document.

- **import_bookmarks** (*bool*) – You may prevent the source document's bookmarks from being imported by specifying this as `False`.

setPageLayout(*layout: typing_extensions.Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]*) → None

Deprecated since version 1.28.0: Use `set_page_layout()` instead.

setPageMode(*mode: typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]*) → None

Deprecated since version 1.28.0: Use `set_page_mode()` instead.

set_page_layout(*layout: typing_extensions.Literal[/NoLayout, /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight]*) → None

Set the page layout

Parameters **layout** (*str*) – The page layout to be used

Table 1: Valid `layout` arguments

/NoLayout	Layout explicitly not specified
/SinglePage	Show one page at a time
/OneColumn	Show one column at a time
/TwoColumnLeft	Show pages in two columns, odd-numbered pages on the left
/TwoColumn-Right	Show pages in two columns, odd-numbered pages on the right
/TwoPageLeft	Show two pages at a time, odd-numbered pages on the left
/TwoPageRight	Show two pages at a time, odd-numbered pages on the right

set_page_mode(*mode: typing_extensions.Literal[/UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments]*) → None

Set the page mode.

Parameters **mode** (*str*) – The page mode to use.

Table 2: Valid `mode` arguments

/UseNone	Do not show outlines or thumbnails panels
/UseOutlines	Show outlines (aka bookmarks) panel
/UseThumbs	Show page thumbnails panel
/FullScreen	Fullscreen view
/UseOC	Show Optional Content Group (OCG) panel
/UseAttachments	Show attachments panel

write(*fileobj: Union[str, _io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*) → None

Writes all data that has been merged to the given output file.

Parameters **fileobj** – Output file. Can be a filename or any kind of file-like object.

CHAPTER

EIGHTEEN

THE PAGEOBJECT CLASS

class PyPDF2._page.**PageObject**(*pdf: Optional[Any] = None*, *indirect_ref: Optional[PyPDF2.generic.IndirectObject] = None*)

Bases: PyPDF2.generic.DictionaryObject

PageObject represents a single page within a PDF file.

Typically this object will be created by accessing the get_page() method of the *PdfReader* class, but it is also possible to create an empty page with the *create_blank_page()* static method.

Parameters

- **pdf** – PDF file the page belongs to.
- **indirect_ref** – Stores the original indirect reference to this object in its source PDF

addTransformation(*ctm: Tuple[float, float, float, float, float, float]*) → None

Deprecated since version 1.28.0: Use *add_transformation()* instead.

add_transformation(*ctm: Union[*PyPDF2._page.Transformation*, Tuple[float, float, float, float, float, float]]*, *expand: bool = False*) → None

Apply a transformation matrix to the page.

Parameters **ctm** (*tuple*) – A 6-element tuple containing the operands of the transformation matrix. Alternatively, a *Transformation* object can be passed.

See *Cropping and Transforming PDFs*.

property artBox: *PyPDF2.generic.RectangleObject*

Deprecated since version 1.28.0.

Use *artbox* instead.

property artbox

A *RectangleObject*, expressed in default user space units, defining the extent of the page's meaningful content as intended by the page's creator.

property bleedBox: *PyPDF2.generic.RectangleObject*

Deprecated since version 1.28.0.

Use *bleedbox* instead.

property bleedbox

A *RectangleObject*, expressed in default user space units, defining the region to which the contents of the page should be clipped when output in a production enviroment.

compressContentStreams() → None

Deprecated since version 1.28.0: Use *compress_content_streams()* instead.

compress_content_streams() → None

Compress the size of this page by joining all content streams and applying a FlateDecode filter.

However, it is possible that this function will perform no action if content stream compression becomes "automatic" for some reason.

static createBlankPage(*pdf: Optional[Any] = None*, *width: Optional[Union[float, decimal.Decimal]] = None*, *height: Optional[Union[float, decimal.Decimal]] = None*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use *create_blank_page()* instead.

static create_blank_page(*pdf: Optional[Any] = None*, *width: Optional[Union[float, decimal.Decimal]] = None*, *height: Optional[Union[float, decimal.Decimal]] = None*) → *PyPDF2._page.PageObject*

Return a new blank page.

If `width` or `height` is `None`, try to get the page size from the last page of *pdf*.

Parameters

- **pdf** – PDF file the page belongs to
- **width** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – The height of the new page expressed in default user space units.

Returns the new blank page:

Return type *PageObject*

Raises **PageSizeNotDefinedError** – if `pdf` is `None` or contains no page

property cropBox: *PyPDF2.generic.RectangleObject*

Deprecated since version 1.28.0.

Use *cropbox* instead.

property cropbox

A *RectangleObject*, expressed in default user space units, defining the visible region of default user space. When the page is displayed or printed, its contents are to be clipped (cropped) to this rectangle and then imposed on the output medium in some implementation-defined manner. Default value: same as *mediabox*.

extractText(*Tj_sep: str = ''*, *TJ_sep: str = ''*) → str

Deprecated since version 1.28.0: Use *extract_text()* instead.

extract_text(*Tj_sep: str = ''*, *TJ_sep: str = ''*, *space_width: float = 200.0*) → str

Locate all text drawing commands, in the order they are provided in the content stream, and extract the text. This works well for some PDF files, but poorly for others, depending on the generator used. This will be refined in the future. Do not rely on the order of text coming out of this function, as it will change if this function is made more sophisticated. space_width : float = force default space width (if not extracted from font (default 200)

Returns a string object.

extract_xform_text(*xform: PyPDF2.generic.EncodedStreamObject*, *space_width: float = 200.0*) → str

Extraction tet from an XObject. space_width : float = force default space width (if not extracted from font (default 200)

Returns a string object.

getContents() → Optional[PyPDF2.generic.ContentStream]

Deprecated since version 1.28.0: Use *get_contents()* instead.

get_contents() → Optional[PyPDF2.generic.ContentStream]

Access the page contents.

Returns the /Contents object, or None if it doesn't exist. /Contents is optional, as described in PDF Reference 7.7.3.3

property mediaBox: *PyPDF2.generic.RectangleObject*

Deprecated since version 1.28.0.

Use *mediabox* instead.

property mediabox

A *RectangleObject*, expressed in default user space units, defining the boundaries of the physical medium on which the page is intended to be displayed or printed.

mergePage(*page2:* PyPDF2._page.PageObject) → None

Deprecated since version 1.28.0: Use *merge_page()* instead.

mergeRotatedPage(*page2:* PyPDF2._page.PageObject, *rotation: float*, *expand: bool = False*) → None

mergeRotatedPage is similar to merge_page, but the stream to be merged is rotated by appling a transformation matrix.

Parameters

- **page2** (PageObject) – the page to be merged into this one. Should be an instance of *PageObject*.
- **rotation** (*float*) – The angle of the rotation, in degrees
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeRotatedScaledPage(*page2:* PyPDF2._page.PageObject, *rotation: float*, *scale: float*, *expand: bool = False*) → None

mergeRotatedScaledPage is similar to merge_page, but the stream to be merged is rotated and scaled by appling a transformation matrix.

Parameters

- **page2** (PageObject) – the page to be merged into this one. Should be an instance of *PageObject*.
- **rotation** (*float*) – The angle of the rotation, in degrees
- **scale** (*float*) – The scaling factor
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeRotatedScaledTranslatedPage(*page2:* PyPDF2._page.PageObject, *rotation: float*, *scale: float*, *tx: float*, *ty: float*, *expand: bool = False*) → None

mergeRotatedScaledTranslatedPage is similar to merge_page, but the stream to be merged is translated, rotated and scaled by appling a transformation matrix.

Parameters

- **page2** (PageObject) – the page to be merged into this one. Should be an instance of *PageObject*.
- **tx** (*float*) – The translation on X axis
- **ty** (*float*) – The translation on Y axis
- **rotation** (*float*) – The angle of the rotation, in degrees
- **scale** (*float*) – The scaling factor
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeRotatedTranslatedPage(*page2: PyPDF2._page.PageObject*, *rotation: float*, *tx: float*, *ty: float*, *expand: bool = False*) → None

mergeRotatedTranslatedPage is similar to merge_page, but the stream to be merged is rotated and translated by appling a transformation matrix.

Parameters

- **page2** (PageObject) – the page to be merged into this one. Should be an instance of *PageObject*.
- **tx** (*float*) – The translation on X axis
- **ty** (*float*) – The translation on Y axis
- **rotation** (*float*) – The angle of the rotation, in degrees
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeScaledPage(*page2: PyPDF2._page.PageObject*, *scale: float*, *expand: bool = False*) → None

mergeScaledPage is similar to merge_page, but the stream to be merged is scaled by appling a transformation matrix.

Parameters

- **page2** (PageObject) – The page to be merged into this one. Should be an instance of *PageObject*.
- **scale** (*float*) – The scaling factor
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeScaledTranslatedPage(*page2: PyPDF2._page.PageObject*, *scale: float*, *tx: float*, *ty: float*, *expand: bool = False*) → None

mergeScaledTranslatedPage is similar to merge_page, but the stream to be merged is translated and scaled by appling a transformation matrix.

Parameters

- **page2** (PageObject) – the page to be merged into this one. Should be an instance of *PageObject*.
- **scale** (*float*) – The scaling factor

- **tx** (*float*) – The translation on X axis
- **ty** (*float*) – The translation on Y axis
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeTransformedPage(*page2:* PyPDF2._page.PageObject, *ctm: Union[Tuple[float, float, float, float, float, float],* PyPDF2._page.Transformation*]*, *expand: bool = False*) → None

mergeTransformedPage is similar to merge_page, but a transformation matrix is applied to the merged stream.

Parameters

- **page2** (PageObject) – The page to be merged into this one. Should be an instance of *PageObject*.
- **ctm** (*tuple*) – a 6-element tuple containing the operands of the transformation matrix
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

mergeTranslatedPage(*page2:* PyPDF2._page.PageObject, *tx: float*, *ty: float*, *expand: bool = False*) → None

mergeTranslatedPage is similar to merge_page, but the stream to be merged is translated by appling a transformation matrix.

Parameters

- **page2** (PageObject) – the page to be merged into this one. Should be an instance of *PageObject*.
- **tx** (*float*) – The translation on X axis
- **ty** (*float*) – The translation on Y axis
- **expand** (*bool*) – Whether the page should be expanded to fit the dimensions of the page to be merged.

Deprecated since version 1.28.0: Use *add_transformation()* and *merge_page()* instead.

merge_page(*page2:* PyPDF2._page.PageObject, *expand: bool = False*) → None

Merge the content streams of two pages into one.

Resource references (i.e. fonts) are maintained from both pages. The mediabox/cropbox/etc of this page are not altered. The parameter page's content stream will be added to the end of this page's content stream, meaning that it will be drawn after, or "on top" of this page.

Parameters

- **page2** (PageObject) – The page to be merged into this one. Should be an instance of *PageObject*.
- **expand** (*bool*) – If true, the current page dimensions will be expanded to accommodate the dimensions of the page to be merged.

rotate(*angle: float*) → *PyPDF2._page.PageObject*

Rotate a page clockwise by increments of 90 degrees.

Parameters **angle** (*int*) – Angle to rotate the page. Must be an increment of 90 deg.

rotateClockwise(*angle: float*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use `rotate_clockwise()` instead.

rotateCounterClockwise(*angle: float*) → *PyPDF2._page.PageObject*

Deprecated since version 1.28.0: Use `rotate_clockwise()` with a negative argument instead.

rotate_clockwise(*angle: float*) → *PyPDF2._page.PageObject*

scale(*sx: float*, *sy: float*) → None

Scale a page by the given factors by appling a transformation matrix to its content and updating the page size.

Parameters

- **sx** (*float*) – The scaling factor on horizontal axis.
- **sy** (*float*) – The scaling factor on vertical axis.

scaleBy(*factor: float*) → None

Deprecated since version 1.28.0: Use `scale_by()` instead.

scaleTo(*width: float*, *height: float*) → None

Deprecated since version 1.28.0: Use `scale_to()` instead.

scale_by(*factor: float*) → None

Scale a page by the given factor by appling a transformation matrix to its content and updating the page size.

Parameters **factor** (*float*) – The scaling factor (for both X and Y axis).

scale_to(*width: float*, *height: float*) → None

Scale a page to the specified dimentions by appling a transformation matrix to its content and updating the page size.

Parameters

- **width** (*float*) – The new width.
- **height** (*float*) – The new heigth.

property trimBox: *PyPDF2.generic.RectangleObject*

Deprecated since version 1.28.0.

Use `trimbox` instead.

property trimbox

A `RectangleObject`, expressed in default user space units, defining the intended dimensions of the finished page after trimming.

CHAPTER

NINETEEN

THE TRANSFORMATION CLASS

class PyPDF2.**Transformation**(*ctm: Tuple[float, float, float, float, float, float] = (1, 0, 0, 1, 0, 0)*)

Bases: `object`

Specify a 2D transformation.

The transformation between two coordinate systems is represented by a 3-by-3 transformation matrix written as follows:

```
a b 0
c d 0
e f 1
```

Because a transformation matrix has only six elements that can be changed, it is usually specified in PDF as the six-element array [a b c d e f].

Coordinate transformations are expressed as matrix multiplications:

```
                          a b 0
[ x y 1 ] = [ x y 1 ] ×   c d 0
                          e f 1
```

```
>>> from PyPDF2 import Transformation
>>> op = Transformation().scale(sx=2, sy=3).translate(tx=10, ty=20)
>>> page.add_transformation(op)
```

static compress(*matrix: Tuple[Tuple[float, float, float], Tuple[float, float, float], Tuple[float, float, float]]*) → Tuple[float, float, float, float, float, float]

property matrix: Tuple[Tuple[float, float, float], Tuple[float, float, float], Tuple[float, float, float]]

rotate(*rotation: float*) → *PyPDF2._page.Transformation*

scale(*sx: Optional[float] = None*, *sy: Optional[float] = None*) → *PyPDF2._page.Transformation*

translate(*tx: float = 0*, *ty: float = 0*) → *PyPDF2._page.Transformation*

CHAPTER

TWENTY

THE DOCUMENTINFORMATION CLASS

class PyPDF2.**DocumentInformation**

Bases: PyPDF2.generic.DictionaryObject

A class representing the basic document metadata provided in a PDF File. This class is accessible through *PdfReader.metadata*.

All text properties of the document metadata have *two* properties, eg. author and author_raw. The non-raw property will always return a TextStringObject, making it ideal for a case where the metadata is being displayed. The raw property can sometimes return a ByteStringObject, if PyPDF2 was unable to decode the string's text encoding; this requires additional safety in the caller and therefore is not as commonly accessed.

property author: Optional[str]

Read-only property accessing the document's **author**. Returns a unicode string (TextStringObject) or None if the author is not specified.

property author_raw: Optional[str]

The "raw" version of author; can return a ByteStringObject.

property creator: Optional[str]

Read-only property accessing the document's **creator**. If the document was converted to PDF from another format, this is the name of the application (e.g. OpenOffice) that created the original document from which it was converted. Returns a unicode string (TextStringObject) or None if the creator is not specified.

property creator_raw: Optional[str]

The "raw" version of creator; can return a ByteStringObject.

getText(*key: str*) → Optional[str]

The text value of the specified key or None.

Deprecated since version 1.28.0: Use the attributes (e.g. *title* / *author*).

property producer: Optional[str]

Read-only property accessing the document's **producer**. If the document was converted to PDF from another format, this is the name of the application (for example, OSX Quartz) that converted it to PDF. Returns a unicode string (TextStringObject) or None if the producer is not specified.

property producer_raw: Optional[str]

The "raw" version of producer; can return a ByteStringObject.

property subject: Optional[str]

Read-only property accessing the document's **subject**. Returns a unicode string (TextStringObject) or None if the subject is not specified.

property subject_raw: Optional[str]

The "raw" version of subject; can return a `ByteStringObject`.

property title: Optional[str]

Read-only property accessing the document's **title**. Returns a unicode string (`TextStringObject`) or `None` if the title is not specified.

property title_raw: Optional[str]

The "raw" version of title; can return a `ByteStringObject`.

CHAPTER

TWENTYONE

THE XMPINFORMATION CLASS

class PyPDF2.xmp.**XmpInformation**(*stream: PyPDF2.generic.ContentStream*)

Bases: PyPDF2.generic.PdfObject

An object that represents Adobe XMP metadata. Usually accessed by *xmp_metadata()*

property custom_properties: Dict[Any, Any]

Retrieves custom metadata properties defined in the undocumented pdfx metadata schema.

Returns a dictionary of key/value items for custom metadata properties.

Return type dict

property dc_contributor: Optional[List[str]]

Contributors to the resource (other than the authors). An unsorted array of names.

property dc_coverage: Optional[Any]

Text describing the extent or scope of the resource.

property dc_creator: Optional[List[Any]]

A sorted array of names of the authors of the resource, listed in order of precedence.

property dc_date: Optional[List[Any]]

A sorted array of dates (datetime.datetime instances) of significance to the resource. The dates and times are in UTC.

property dc_description: Optional[Dict[Any, Any]]

A language-keyed dictionary of textual descriptions of the content of the resource.

property dc_format: Optional[Any]

The mime-type of the resource.

property dc_identifier: Optional[Any]

Unique identifier of the resource.

property dc_language: Optional[List[str]]

An unordered array specifying the languages used in the resource.

property dc_publisher: Optional[List[str]]

An unordered array of publisher names.

property dc_relation: Optional[List[str]]

An unordered array of text descriptions of relationships to other documents.

property dc_rights: Optional[Dict[Any, Any]]

A language-keyed dictionary of textual descriptions of the rights the user has to this resource.

property dc_source: Optional[Any]

Unique identifier of the work from which this resource was derived.

property dc_subject: Optional[List[str]]

An unordered array of descriptive phrases or keywrods that specify the topic of the content of the resource.

property dc_title: Optional[Dict[Any, Any]]

A language-keyed dictionary of the title of the resource.

property dc_type: Optional[List[str]]

An unordered array of textual descriptions of the document type.

getElement(*aboutUri: str*, *namespace: str*, *name: str*) → Iterator[Any]

Deprecated since version 1.28.0: Use *get_element()* instead.

getNodesInNamespace(*aboutUri: str*, *namespace: str*) → Iterator[Any]

Deprecated since version 1.28.0: Use *get_nodes_in_namespace()* instead.

get_element(*about_uri: str*, *namespace: str*, *name: str*) → Iterator[Any]

get_nodes_in_namespace(*about_uri: str*, *namespace: str*) → Iterator[Any]

property pdf_keywords: Optional[Any]

An unformatted text string representing document keywords.

property pdf_pdfversion: Optional[Any]

The PDF file version, for example 1.0, 1.3.

property pdf_producer: Optional[Any]

The name of the tool that created the PDF document.

writeToStream(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*, *encryption_key: Union[None, str, bytes]*) → None

Deprecated since version 1.28.0: Use *write_to_stream()* instead.

write_to_stream(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*, *encryption_key: Union[None, str, bytes]*) → None

property xmp_createDate: Optional[Any]

The date and time the resource was originally created. The date and time are returned as a UTC datetime.datetime object.

property xmp_creatorTool: Optional[Any]

The name of the first known tool used to create the resource.

property xmp_metadataDate: Optional[Any]

The date and time that any metadata for this resource was last changed. The date and time are returned as a UTC datetime.datetime object.

property xmp_modifyDate: Optional[Any]

The date and time the resource was last modified. The date and time are returned as a UTC datetime.datetime object.

property xmpmm_documentId: Optional[Any]

The common identifier for all versions and renditions of this resource.

property xmpmm_instanceId: Optional[Any]

An identifier for a specific incarnation of a document, updated each time a file is saved.

CHAPTER

TWENTYTWO

THE DESTINATION CLASS

class PyPDF2.generic.**Destination**(*title: str*, *page: Union[PyPDF2.generic.NumberObject, PyPDF2.generic.IndirectObject, PyPDF2.generic.NullObject, PyPDF2.generic.DictionaryObject]*, *typ: Union[str, PyPDF2.generic.NumberObject]*, **args: Any*)

Bases: PyPDF2.generic.TreeObject

A class representing a destination within a PDF file. See section 8.2.1 of the PDF 1.6 reference.

Parameters

- **title** (*str*) – Title of this destination.
- **page** (*IndirectObject*) – Reference to the page of this destination. Should be an instance of IndirectObject.
- **typ** (*str*) – How the destination is displayed.
- **args** – Additional arguments may be necessary depending on the type.

Raises **PdfReadError** – If destination type is invalid.

Table 1: Valid typ arguments (see PDF spec for details)

/Fit	No additional arguments
/XYZ	[left] [top] [zoomFactor]
/FitH	[top]
/FitV	[left]
/FitR	[left] [bottom] [right] [top]
/FitB	No additional arguments
/FitBH	[top]
/FitBV	[left]

property bottom: Optional[PyPDF2.generic.FloatObject]

Read-only property accessing the bottom vertical coordinate.

Return type float, or None if not available.

property dest_array: PyPDF2.generic.ArrayObject

getDestArray() → PyPDF2.generic.ArrayObject

Deprecated since version 1.28.3: Use *dest_array* instead.

property left: Optional[PyPDF2.generic.FloatObject]

Read-only property accessing the left horizontal coordinate.

Return type float, or None if not available.

property page: Optional[int]

Read-only property accessing the destination page number.

Return type int

property right: Optional[PyPDF2.generic.FloatObject]

Read-only property accessing the right horizontal coordinate.

Return type float, or None if not available.

property title: Optional[str]

Read-only property accessing the destination title.

Return type str

property top: Optional[PyPDF2.generic.FloatObject]

Read-only property accessing the top vertical coordinate.

Return type float, or None if not available.

property typ: Optional[str]

Read-only property accessing the destination type.

Return type str

write_to_stream(*stream: Union[_io.BytesIO, _io.BufferedReader, _io.BufferedWriter, _io.FileIO]*, *encryption_key: Union[None, str, bytes]*) → None

property zoom: Optional[int]

Read-only property accessing the zoom factor.

Return type int, or None if not available.

CHAPTER

TWENTYTHREE

THE RECTANGLEOBJECT CLASS

class PyPDF2.generic.**RectangleObject**(*arr: Tuple[float, float, float, float]*)

Bases: PyPDF2.generic.ArrayObject

This class is used to represent *page boxes* in PyPDF2. These boxes include:

- *artbox*
- *bleedbox*
- *cropbox*
- *mediabox*
- *trimbox*

property bottom: PyPDF2.generic.FloatObject

ensureIsNumber(*value: Any*) → Union[PyPDF2.generic.FloatObject, PyPDF2.generic.NumberObject]

getHeight() → decimal.Decimal

getLowerLeft() → Tuple[decimal.Decimal, decimal.Decimal]

getLowerLeft_x() → PyPDF2.generic.FloatObject

getLowerLeft_y() → PyPDF2.generic.FloatObject

getLowerRight() → Tuple[decimal.Decimal, decimal.Decimal]

getLowerRight_x() → PyPDF2.generic.FloatObject

getLowerRight_y() → PyPDF2.generic.FloatObject

getUpperLeft() → Tuple[decimal.Decimal, decimal.Decimal]

getUpperLeft_x() → PyPDF2.generic.FloatObject

getUpperLeft_y() → PyPDF2.generic.FloatObject

getUpperRight() → Tuple[decimal.Decimal, decimal.Decimal]

getUpperRight_x() → PyPDF2.generic.FloatObject

getUpperRight_y() → PyPDF2.generic.FloatObject

getWidth() → decimal.Decimal

property height: decimal.Decimal

property left: PyPDF2.generic.FloatObject

property lowerLeft: Tuple[decimal.Decimal, decimal.Decimal]

property lowerRight: Tuple[decimal.Decimal, decimal.Decimal]

property lower_left: Tuple[decimal.Decimal, decimal.Decimal]

Property to read and modify the lower left coordinate of this box in (x,y) form.

property lower_right: Tuple[decimal.Decimal, decimal.Decimal]

Property to read and modify the lower right coordinate of this box in (x,y) form.

property right: PyPDF2.generic.FloatObject

setLowerLeft(*value: Tuple[float, float]*) → None

setLowerRight(*value: Tuple[float, float]*) → None

setUpperLeft(*value: Tuple[float, float]*) → None

setUpperRight(*value: Tuple[float, float]*) → None

property top: PyPDF2.generic.FloatObject

property upperLeft: Tuple[decimal.Decimal, decimal.Decimal]

property upperRight: Tuple[decimal.Decimal, decimal.Decimal]

property upper_left: Tuple[decimal.Decimal, decimal.Decimal]

Property to read and modify the upper left coordinate of this box in (x,y) form.

property upper_right: Tuple[decimal.Decimal, decimal.Decimal]

Property to read and modify the upper right coordinate of this box in (x,y) form.

property width: decimal.Decimal

CHAPTER

TWENTYFOUR

THE FIELD CLASS

class PyPDF2.generic.**Field**(*data: Dict[str, Any]*)

Bases: PyPDF2.generic.TreeObject

A class representing a field dictionary. This class is accessed through *get_fields()*

property additionalActions: Optional[PyPDF2.generic.DictionaryObject]

Deprecated since version 1.28.3.

Use *additional_actions* instead.

property additional_actions: Optional[PyPDF2.generic.DictionaryObject]

Read-only property accessing the additional actions dictionary. This dictionary defines the field's behavior in response to trigger events. See Section 8.5.2 of the PDF 1.7 reference.

property altName: Optional[str]

Deprecated since version 1.28.3.

Use *alternate_name* instead.

property alternate_name: Optional[str]

Read-only property accessing the alternate name of this field.

property defaultValue: Optional[Any]

Deprecated since version 1.28.3.

Use *default_value* instead.

property default_value: Optional[Any]

Read-only property accessing the default value of this field.

property fieldType: Optional[PyPDF2.generic.NameObject]

Deprecated since version 1.28.3.

Use *field_type* instead.

property field_type: Optional[PyPDF2.generic.NameObject]

Read-only property accessing the type of this field.

property flags: Optional[int]

Read-only property accessing the field flags, specifying various characteristics of the field (see Table 8.70 of the PDF 1.7 reference).

property kids: Optional[PyPDF2.generic.ArrayObject]

Read-only property accessing the kids of this field.

property mappingName: Optional[str]

Deprecated since version 1.28.3.

Use *mapping_name* instead.

property mapping_name: Optional[str]

Read-only property accessing the mapping name of this field. This name is used by PyPDF2 as a key in the dictionary returned by *get_fields()*

property name: Optional[str]

Read-only property accessing the name of this field.

property parent: Optional[PyPDF2.generic.DictionaryObject]

Read-only property accessing the parent of this field.

property value: Optional[Any]

Read-only property accessing the value of this field. Format varies based on field type.

CHAPTER

TWENTYFIVE

THE PAGERANGE CLASS

class PyPDF2.**PageRange**(*arg: Union[slice,* PyPDF2.pagerange.PageRange*, str]*)

Bases: `object`

A slice-like representation of a range of page indices.

For example, page numbers, only starting at zero.

The syntax is like what you would put between brackets [ ]. The slice is one of the few Python types that can't be subclassed, but this class converts to and from slices, and allows similar use.

- PageRange(str) parses a string representing a page range.
- PageRange(slice) directly "imports" a slice.
- to_slice() gives the equivalent slice.
- str() and repr() allow printing.
- indices(n) is like slice.indices(n).

indices(*n: int*) → Tuple[int, int, int]

n is the length of the list of pages to choose from. Returns arguments for range(). See help(slice.indices).

to_slice() → slice

Return the slice equivalent of this page range.

static valid(*input: Any*) → bool

True if input is a valid initializer for a PageRange.

CHAPTER

TWENTYSIX

DEVELOPER INTRO

PyPDF2 is a library and hence its users are developers. This document is not for the users, but for people who want to work on PyPDF2 itself.

26.1 Installing Requirements

```
pip install -r requirements/dev.txt
```

26.2 Running Tests

```
pytest .
```

We have the following pytest markers defined:

- `external`: Tests which use files from the `sample-files` git submodule

You can locally choose not to run those via `pytest -m "not external"`.

26.3 The sample-files git submodule

The reason for having the submodule `sample-files` is that we want to keep the size of the PyPDF2 repository small while we also want to have an extensive test suite. Those two goals contradict each other.

The `resources` folder should contain a select set of core examples that cover most cases we typically want to test for. The `sample-files` might cover a lot more edge cases, the behavior we get when file sizes get bigger, different PDF producers.

26.4 Tools: git and pre-commit

Git is a command line application for version control. If you don't know it, you can play ohmygit to learn it.

Github is the service where the PyPDF2 project is hosted. While git is free and open source, Github is a paid service by Microsoft - but for free in lot of cases.

pre-commit is a command line application that uses git hooks to automatically execute code. This allows you to avoid style issues and other code quality issues. After you entered `pre-commit install` once in your local copy of PyPDF2, it will automatically be executed when you `git commit`.

26.5 Commit Messages

Having a clean commit message helps people to quickly understand what the commit was about, witout actually looking at the changes. The first line of the commit message is used to auto-generate the CHANGELOG. For this reason, the format should be:

```
PREFIX: DESCRIPTION

BODY
```

The `PREFIX` can be:

- `BUG`: A bug was fixed. Likely there is one or multiple issues. Then write in the `BODY`: `Closes #123` where 123 is the issue number on Github. It would be absolutely amazing if you could write a regression test in those cases. That is a test that would fail without the fix.
- `ENH`: A new feature! Describe in the body what it can be used for.
- `DEP`: A deprecation - either marking something as "this is going to be removed" or actually removing it.
- `ROB`: A robustness change. Dealing better with broken PDF files.
- `DOC`: A documentation change.
- `TST`: Adding / adjusting tests.
- `DEV`: Developer experience improvements - e.g. pre-commit or setting up CI
- `MAINT`: Quite a lot of different stuff. Performance improvements are for sure the most interesting changes in here. Refactorings as well.
- `STY`: A style change. Something that makes PyPDF2 code more consistent. Typically a small change.

26.6 Benchmarks

We need to keep an eye on performance and thus we have a few benchmarks.

See py-pdf.github.io/PyPDF2/dev/bench

CHAPTER

TWENTYSEVEN

THE PDF FORMAT

It's recommended to look in the PDF specification for details and clarifications. This is only intended to give a very rough overview of the format.

27.1 Overall Structure

A PDF consists of:

1. Header: Contains the version of the PDF, e.g. `%PDF-1.7`
2. Body: Contains a sequence of indirect objects
3. Cross-reference table (xref): Contains a list of the indirect objects in the body
4. Trailer

27.2 The xref table

A cross-reference table (xref) is a table of the indirect objects in the body. It allows quick access to those objects by pointing to their location in the file.

It looks like this:

```
xref 42 5
0000001000 65535 f
0000001234 00000 n
0000001987 00000 n
0000011987 00000 n
0000031987 00000 n
```

Let's go through it step-by-step:

- `xref` is justa keyword that specifies the start of the xref table.
- `42` is TODO; `6` is the number of entries in the xref table.
- Now every object has 3 entries `nnnnnnnnnn ggggg n`: The 10-digit byte offset, a 5-digit generation number, and a literal keyword which is either `n` or `f`.
  - `nnnnnnnnnn` is the byte offset of the object. It tells the reader where the object is in the file.
  - `ggggg` is the generation number. It tells the reader how old the object is.
  - `n` means that the object is a normal in-use object, `f` means that the object is a free object.

* The first free object always has a generation number of 65535. It forms the head of a linked-list of all free objects.
* The generation number of a normal object is always 0. The generation number allows the PDF format to contain multiple versions of the same object. This is a version history mechanism.

27.3 The body

The body is a sequence of indirect objects:

`counter generationnumber << the_object >> endobj`

- `counter` (integer) is a unique identifier for the object.
- `generationnumber` (integer) is the generation number of the object.
- `the_object` is the object itself. It can be empty. Starts with `/Keyword` to specify which kind of object it is.
- `endobj` marks the end of the object.

A concrete example can be found in `test_reader.py::test_get_images_raw`:

```
1 0 obj << /Count 1 /Kids [4 0 R] /Type /Pages >> endobj
2 0 obj << >> endobj
3 0 obj << >> endobj
4 0 obj << /Contents 3 0 R /CropBox [0.0 0.0 2550.0 3508.0]
 /MediaBox [0.0 0.0 2550.0 3508.0] /Parent 1 0 R
 /Resources << /Font << >> >>
 /Rotate 0 /Type /Page >> endobj
5 0 obj << /Pages 1 0 R /Type /Catalog >> endobj
```

27.4 The trailer

The trailer looks like this:

```
trailer << /Root 5 0 R
           /Size 6
        >>
startxref 1234
%%EOF
```

Let's go through it:

- `trailer` << indicates that the *trailer dictionary* starts. It ends with >>.
- `startxref` is a keyword followed by the byte-location of the `xref` keyword. As the trailer is always at the bottom of the file, this allows readers to quickly find the xref table.
- `%%EOF` is the end-of-file marker.

The trailer dictionary is a key-value list. The keys are specified in Table 3.13 of the PDF Reference 1.7, e.g. `/Root` and `/Size` (both are required).

- `/Root` (dictionary) contains the document catalog.
  - The `5` is the object number of the catalog dictionary

  - `0` is the generation number of the catalog dictionary
  - `R` is the keyword that indicates that the object is a reference to the catalog dictionary.
- `/Size` (integer) contains the total number of entries in the files xref table.

27.5 Reading PDF files

Most PDF files are compressed. If you want to read them, first uncompress them:

```
pdftk crazyones.pdf output crazyones-uncomp.pdf uncompress
```

Then rename `crazyones-uncomp.pdf` to `crazyones-uncomp.txt` and open it in our favorite IDE / text editor.

CHAPTER

TWENTYEIGHT

CMAPS

Looking at the cmap of "crazyones":

```
pdftk crazyones.pdf output crazyones-uncomp.pdf uncompress
```

You can see this:

```
begincmap
/CMapName /T1Encoding-UTF16 def
/CMapType 2 def
/CIDSystemInfo <<
  /Registry (Adobe)
  /Ordering (UCS)
  /Supplement 0
>> def
1 begincodespacerange
<00> <FF>
endcodespacerange
1 beginbfchar
<1B> <FB00>
endbfchar
endcmap
CMapName currentdict /CMap defineresource pop
```

28.1 codespacerange

A codespacerange maps a complete sequence of bytes to a range of unicode glyphs. It defines a starting point:

```
1 beginbfchar
<1B> <FB00>
```

That means that `1B` (Hex for 27) maps to the unicode character `FB00` - the ligature (two lowercase f's).

The two numbers in `begincodespacerange` mean that it starts with an offset of 0 (hence from `1B` `FB00`) upt to an offset of FF (dec: 255), hence 1B+FF = 282 FBFF.

Within the text stream, there is

```
(The)-342(mis\034ts.)
```

`\034` is octal for 28 decimal.

CHAPTER

TWENTYNINE

THE DEPRECATION PROCESS

PyPDF2 strives to be an excellent library for its current users and for new ones. We are careful with introducing potentially breaking changes, but we will do them if they provide value for the community on the long run.

We hope and think that deprecations will not happen soon again. If they do, users can rely on the following procedure.

29.1 Semantic Versioning

PyPDF2 uses semantic versioning. If you want to avoid breaking changes, please use dependency pinning (also known as version pinning). In Python, this is done by specifying the exact version you want to use in a `requirements.txt` file. A tool that can support you is `pip-compile` from `pip-tools`.

If you are using Poetry it is done with the `poetry.lock` file.

29.2 How PyPDF2 deprecates features

Assume the current version of PyPDF2 is `x.y.z`. After a discussion (e.g. via Github issues) we decided to remove a class / function / method. This is how we do it:

1. `x.y.(z+1)`: Add a PendingDeprecationWarning. If there is a replacement, the replacement is also introduced and the warning informs about the change and when it will happen. The docs let users know about the deprecation and when it will happen and the new function. The CHANGELOG informs about it.
2. `(x+1).0.0`: The PendingDeprecationWarning is changed to a DeprecationWarning. The CHANGELOG informs about it.
3. `(x+2).0.0`: The code and the DeprecationWarnings are removed. The CHANGELOG informs about it.

This means the users have 3 warnings in the CHANGELOG, a PendingDeprecationWarning until the next major release and a DeprecationWarning until the major release after that.

Please note that adding warnings can be a breaking change for some users; most likely just in the CI. This means it needs to be properly documented.

CHAPTER

THIRTY

PROJECT GOVERNANCE

This document describes how the PyPDF2 project is managed. It describes the different actors, their roles, and the responsibilities they have.

30.1 Terminology

- The **project** is PyPDF2 - a free and open-source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It includes the code, issues, and disscussions on GitHub, and the documentation on ReadTheDocs, the package on PyPI, and the website on GitHub.
- A **maintainer** is a person who has technical permissions to change one or more part of the projects. It is a person who is driven to keep the project running and improving.
- A **contributor** is a person who contributes to the project. That could be through writing code - in the best case through forking and creating a pull request, but that is up to the maintainer. Other contributors describe issues, help to ask questions on existing issues to make them easier to answer, participate in discussions, and help to improve the documentation. Contributors are similar to maintainers, but without technial permissions.
- A **user** is a person who imports PyPDF2 into their code. All PyPDF2 users are developers, but not developers who know the internals of PyPDF2. They only use the public interface of PyPDF2. They will likely have less knowledge about PDF than contributors.
- The **community** is all of that - the users, the contributors, and the maintainers.

30.2 Governance, Leadership, and Steering PyPDF2 forward

PyPDF2 is a free and open source project with over 100 contributors and likely (way) more than 1000 users.

As PyPDF2 does not have any formal relationship with any company and no funding, all the work done by the community are voluntary contributions. People don't get paid, but choose to spend their free time to create software of which many more are profiting. This has to be honored and respected.

Despite such a big community, the project was dormant from 2016 to 2022. There were still questions asked, issues reported, and pull requests created. But the maintainer didn't have the time to move PyPDF2 forward. During that time, nobody else stepped up to become the new maintainer.

For this reason, PyPDF2 has the **Benevolent Dictator** governance model. The benevolent dictator is a maintainer with all technical permissions - most importantly the permission to push new PyPDF2 versions on PyPI.

Being benevolent, the benevolent dictator listens for decisions to the community and tries their best to make decisions from which the overall community profits - the current one and the potential future one. Being a dictator, the benevolent dictator always has the power and the right to make decisions on their own - also against some members of the community.

As PyPDF2 is free software, parts of the community can split off (fork the code) and create a new community. This should limit the harm a bad benevolent dictator can do.

30.3 Project Language

The project language is (american) English. All documentation and issues must be written in English to ensure that the community can understand it.

We appreciate the fact that large parts of the community don't have English as their mother tongue. We try our best to understand others - automatic translators might help.

30.4 Expectations

The community can expect the following:

- The **benevolent dictator** tries their best to make decisions from which the overall community profits. The benevolent dictator is aware that his/her decisons can shape the overall community. Once the benevolent dictator notices that she/he doesn't have the time to advance PyPDF2, he/she looks for a new benevolent dictator. As it is expected that the benevolent dictator will step down at some point of their choice (hopefully before their death), it is NOT a benevolent dictator for life (BDFL).
- Every **maintainer** (including the benevolent dictator) is aware of their permissions and the harm they could do. They value security and ensure that the project is not harmed. They give their technical permissions back if they don't need them any longer. Any long-time contributor can become a maintainer. Maintainers can - and should! - step down from their role when they realize that they can no longer commit that time. Their contribution will be honored in the *History of PyPDF2*.
- Every **contributor** is aware that the time of maintainers and the benevolent dictator is limited. Short pull requests that briefly describe the solved issue and have a unit test have a higher chance to get merged soon - simply because it's easier for maintainers to see that the contribution will not harm the overall project. Their contributions are documented in the git history and in the public issues. Let us know if you would appriciate something else!
- Every **community member** uses a respectful language. We are all human, we get upset about things we care and other things than what's visible on the internet go on in our live. PyPDF2 does not pay its contributors - keep all of that in mind when you interact with others. We are here because we want to help others.

30.4.1 Issues and Discussions

An issue is any technical description that aims at bringing PyPDF2 forward:

- Bugs tickets: Something went wrong because PyPDF2 developers made a mistake.
- Feature requests: PyPDF2 does not support all features of the PDF specifications. There are certainly also convenience methods that would help users a lot.
- Robustness requests: There are many broken PDFs around. In some cases, we can deal with that. It's kind of a mixture between a bug ticket and a feature request.
- Performance tickets: PyPDF2 could be faster - let us know about your specific scenario.

Any comment that is in those technial descriptions which is not helping the discussion can be deleted. This is especially true for "me too" comments on bugs or "bump" comments for desired features. People can express this with / reactions.

Discussions are open. No comments will be deleted there - except if they are clearly unrelated spam or only try to insult people (luckily, the community was very respectful so far)

30.4.2 Releases

The maintainers follow semantic versioning. Most importantly, that means that breaking changes will have a major version bump.

Be aware that unintentional breaking changes might still happen. The PyPDF2 maintainers do their best to fix that in a timely manner - please report such issues!

30.5 People

- Martin Thoma is benevolent dictator since April 2022.
- Maintainers:
  - Matthew Stamy (mstamy2) was the benevolent dictator for a long time. He still is around on Github once in a while and has permissions on PyPI and Github.
  - Matthew Peveler (MasterOdin) is a maintainer on Github.

CHAPTER

THIRTYONE

HISTORY OF PYPDF2

31.1 The Origins: pyPdf (2005-2010)

In 2005, Mathieu Fenniak launched pyPdf "as a PDF toolkit…" focused on

- document manipulation: by-page splitting, concatenation, and merging;
- document introspection;
- page cropping; and
- document encryption and decryption.

The last release of PyPI was pyPdf 1.13 in 2010.

31.2 PyPDF2 is born (2011-2016)

At the end of 2011, after consultation with Mathieu and others, Phaseit sponsored PyPDF2 as a fork of pyPdf on GitHub. The initial impetus was to handle a wider range of input PDF instances; Phaseit's commercial work often encounters PDF instances "in the wild" that it needs to manage (mostly concatenate and paginate), but that deviate so much from PDF standards that pyPdf can't read them. PyPDF2 reads a considerably wider range of real-world PDF instances.

Neither pyPdf nor PyPDF2 aims to be universal, that is, to provide all possible PDF-related functionality. Note that the similar-appearing pyfpdf of Mariano Reingart is most comparable to ReportLab, in that both ReportLab and pyfpdf emphasize document generation. Interestingly enough, pyfpdf builds in a basic HTML→PDF converter while PyPDF2 has no knowledge of HTML.

So what is PyPDF2 truly about? Think about popular pdftk for a moment. PyPDF2 does what pdftk does, and it does so within your current Python process, and it handles a wider range of variant PDF formats [explain]. PyPDF2 has its own FAQ to answer other questions that have arisen.

The Reddit /r/python crowd chatted obliquely and briefly about PyPDF2 in March 2012.

31.3 PyPDF3 and PyPDF4 (2018 - 2022)

Two approaches were made to get PyPDF2 active again: PyPDF3 and PyPDF4.

PyPDF3 had it's first release in 2018 and its last one in February 2022. It never got the user base from PyPDF2.

PyPDF4 only had one release in 2018.

31.4 PyPDF2: Reborn (2022-Today)

Martin Thoma took over maintenance of PyPDF2 in April 2022.

CHAPTER

THIRTYTWO

PYPDF2 VS X

PyPDF2 is a free and open source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. PyPDF2 can retrieve text and metadata from PDFs as well.

32.1 PyMuPDF and PikePDF

PyMuPDF is a Python binding to MuPDF and PikePDF is the Python binding to QPDF.

While both are excellent libraries for various use-cases, using them is not always possible even when they support the use-case. Both of them are powered by C libraries which makes installation harder and might cause security concerns. For MuPDF you might also need to buy a commercial license.

A core feature of PyPDF2 is that it's pure Python. That means there is no C dependency. It has been used for over 10 years and for this reason a lot of support via StackOverflow and examples on the internet.

32.2 pyPDF

PyPDF2 was forked from pyPDF. pyPDF has been unmaintained for a long time.

32.3 PyPDF3 and PyPDF4

Developing and maintaining open source software is extremely time-intensive and in the case of PyPDF2 not paid at all. Having a continuous support is hard.

PyPDF2 was initially released in 2012 on PyPI and received releases until 2016. From 2016 to 2022 there was no update - but people were still using it.

As PyPDF2 is free software, there were attempts to fork it and continue the development. PyPDF3 was first released in 2018 and still receives updates. PyPDF4 has only one release from 2018.

I, Martin Thoma, the current maintainer of PyPDF2, hope that we can bring the community back to one path of development. Let's see.

32.4 pdfrw and pdfminer

I don't have experience with either of those libraries. Please add a comparison if you know PyPDF2 and `pdfrw` or `pdfminer.six`!

Please be aware that there is also `pdfminer` which is not maintained. Then there is `pdfrw2` which doesn't have a large community behind it.

And there are more:

- `pdfplumber`

32.5 Document Generation

There are (Python) tools to generate PDF documents. PyPDF2 is not one of them.

CHAPTER

THIRTYTHREE

FREQUENTLY-ASKED QUESTIONS

33.1 How is PyPDF2 related to pyPdf?

PyPDF2 is a fork from the no-longer-maintained pyPdf approved by the latter's founder.

33.2 Which Python versions are supported?

PyPDF2 2.0+ supports Python 3.6 and later. PyPDF2 1.27.10 supported Python 2.7 to 3.10.

CHAPTER

THIRTYFOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

V

W

X

Z