PyPDF2 Documentation

Contents:

About PyPDF2

PyPDF2 is a pure-python PDF toolkit originating from the pyPdf project. It is currently maintained by Phaseit, Inc. PyPDF2 can extract data from PDF files, or manipulate existing PDFs to produce a new file. PyPDF2 is compatible with Python versions 2.6, 2.7, and 3.2 - 3.5.

The PdfFileReader Class

class `PyPDF2.PdfFileReader`(*stream*, *strict=True*, *warndest=None*, *overwriteWarnings=True*)

Initializes a PdfFileReader 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 `True`.
- **warndest** – Destination for logging warnings (defaults to `sys.stderr`).
- **overwriteWarnings** (*bool*) – Determines whether to override Python's `warnings.py` module with a custom implementation (defaults to `True`).

`decrypt`(*password*)

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.

`documentInfo`

Read-only property that accesses the `getDocumentInfo()` function.

`getDestinationPageNumber`(*destination*)

Retrieve page number of a given Destination object

Parameters: **destination** (*Destination*) – The destination to get page number. Should be an instance of `Destination`

Returns: the page number or -1 if page not found

Return type: int

`getDocumentInfo`()

Retrieves 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.

`getFields`(*tree=None*, *retval=None*, *fileobj=None*)

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.

`getFormTextFields`()

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

`getNamedDestinations`(*tree=None*, *retval=None*)

Retrieves the named destinations present in the document.

Returns: a dictionary which maps names to `Destinations`.

Return type: dict

`getNumPages`()

Calculates the number of pages in this PDF file.

Returns: number of pages

Return type: int

Raises PdfReadError:

if file is encrypted and restrictions prevent this action.

`getOutlines`(*node=None*, *outlines=None*)

Retrieves the document outline present in the document.

Returns: a nested list of `Destinations`.

`getPage`(*pageNumber*)

Retrieves a page by number from this PDF file.

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

Returns: a `PageObject` instance.

Return type: PageObject

getPageLayout()

Get the page layout. See setPageLayout() for a description of valid layouts.

Returns: Page layout currently being used.

Return type: str, None if not specified

getPageMode()

Get the page mode. See setPageMode() for a description of valid modes.

Returns: Page mode currently being used.

Return type: str, None if not specified

getPageNumber(*page*)

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

getXmpMetadata()

Retrieves XMP (Extensible Metadata Platform) data from the PDF document root.

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.

isEncrypted

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.

namedDestinations

Read-only property that accesses the getNamedDestinations() function.

numPages

Read-only property that accesses the getNumPages() function.

outlines

Read-only property that accesses the getOutlines() function.

pageLayout

Read-only property accessing the getPageLayout() method.

`pageMode`

Read-only property accessing the `getPageMode()` method.

`pages`

Read-only property that emulates a list based upon the `getNumPages()` and `getPage()` methods.

`xmpMetadata`

Read-only property that accesses the `getXmpMetadata()` function.

The PdfFileMerger Class

class `PyPDF2.PdfFileMerger`(*strict=True*)

Initializes a PdfFileMerger object. PdfFileMerger 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 `True`.

`addBookmark`(*title*, *pagenum*, *parent=None*)

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.

`addMetadata`(*infos*)

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'}`

`addNamedDestination`(*title*, *pagenum*)

Add a destination to the output.

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

`append`(*fileobj*, *bookmark=None*, *pages=None*, *import_bookmarks=True*)

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 Page Range 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`()

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

`merge`(*position*, *fileobj*, *bookmark=None*, *pages=None*, *import_bookmarks=True*)

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 Page Range 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*)

Set the page layout

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

Valid layouts are:

`/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
`/TwoColumnRight`	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

`setPageMode`(*mode*)

Set the page mode.

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

Valid modes are:

`/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*)

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.

The PageObject Class

class `PyPDF2.pdf.PageObject`(*pdf=None*, *indirectRef=None*)

This class represents a single page within a PDF file. Typically this object will be created by accessing the `getPage()` method of the `PdfFileReader` class, but it is also possible to create an empty page with the `createBlankPage()` static method.

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

`addTransformation`(*ctm*)

Applies a transformation matrix to the page.

Parameters: **ctm** (*tuple*) – A 6-element tuple containing the operands of the transformation matrix.

`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.

`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`()

Compresses 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=None*, *width=None*, *height=None*)

Returns 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

`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`()

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.

Returns: a unicode string object.

`getContents`()

Accesses 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

`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*)

Merges 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.

`mergeRotatedPage`(*page2*, *rotation*, *expand=False*)

This is similar to mergePage, 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.

`mergeRotatedScaledPage`(*page2*, *rotation*, *scale*, *expand=False*)

This is similar to mergePage, 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.

`mergeRotatedScaledTranslatedPage`(*page2*, *rotation*, *scale*, *tx*, *ty*, *expand=False*)

This is similar to mergePage, 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.

`mergeRotatedTranslatedPage`(*page2*, *rotation*, *tx*, *ty*, *expand=False*)

This is similar to mergePage, 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.

`mergeScaledPage`(*page2*, *scale*, *expand=False*)

This is similar to mergePage, 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.

`mergeScaledTranslatedPage`(*page2*, *scale*, *tx*, *ty*, *expand=False*)

This is similar to mergePage, 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.

`mergeTransformedPage`(*page2*, *ctm*, *expand=False*)

This is similar to mergePage, 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.

`mergeTranslatedPage`(*page2*, *tx*, *ty*, *expand=False*)

This is similar to mergePage, 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.

`rotateClockwise`(*angle*)

Rotates a page clockwise by increments of 90 degrees.

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

`rotateCounterClockwise`(*angle*)

Rotates a page counter-clockwise by increments of 90 degrees.

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

`scale`(*sx*, *sy*)

Scales 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*)

Scales 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).

`scaleTo`(*width*, *height*)

Scales 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.

`trimBox`

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

The PdfFileWriter Class

class `PyPDF2.PdfFileWriter`

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

`addAttachment`(*fname*, *fdata*)

Embed a file inside the PDF.

Parameters:
- **fname** (*str*) – The filename to display.
- **fdata** (*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

`addBlankPage`(*width=None*, *height=None*)

Appends 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.

`addBookmark`(*title*, *pagenum*, *parent=None*, *color=None*, *bold=False*, *italic=False*, *fit='/Fit'*, **args*)

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.

addJS(*javascript*)

Add Javascript which will launch upon opening this PDF.

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

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

addLink(*pagenum*, *pagedest*, *rect*, *border=None*, *fit='/Fit'*, **args*)

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.

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]

addMetadata(*infos*)

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.

addPage(*page*)

Adds a page to this PDF file. The page is usually acquired from a PdfFileReader instance.

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

`appendPagesFromReader`(*reader*, *after_page_append=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 PdfFileReader object from which to copy page annotations to this writer object. The writer's annots

will then be updated :callback after_page_append (function): Callback function that is invoked after

each page is appended to the writer. Callback signature:

param writer_pageref (PDF page reference):

Reference to the page appended to the writer.

`cloneDocumentFromReader`(*reader*, *after_page_append=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.

`cloneReaderDocumentRoot`(*reader*)

Copy the reader document root to the writer.

Parameters: **reader** – PdfFileReader from the document root should be copied. :callback after_page_append

`encrypt`(*user_pwd*, *owner_pwd=None*, *use_128bit=True*)

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.

`getNumPages`()

Returns: the number of pages.

Return type: int

`getPage`(*pageNumber*)

Retrieves 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`

`getPageLayout`()

Get the page layout. See `setPageLayout()` for a description of valid layouts.

Returns: Page layout currently being used.

Return type: str, None if not specified

`getPageMode`()

Get the page mode. See `setPageMode()` for a description of valid modes.

Returns: Page mode currently being used.

Return type: str, None if not specified

`insertBlankPage`(*width=None*, *height=None*, *index=0*)

Inserts 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.

`insertPage`(*page*, *index=0*)

Insert a page in this PDF file. The page is usually acquired from a `PdfFileReader` 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.

`pageLayout`

Read and write property accessing the `getPageLayout()` and `setPageLayout()` methods.

`pageMode`

Read and write property accessing the `getPageMode()` and `setPageMode()` methods.

`removeImages`(*ignoreByteStringObject=False*)

Removes images from this output.

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

`removeLinks`()

Removes links and annotations from this output.

`removeText`(*ignoreByteStringObject=False*)

Removes images from this output.

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

`setPageLayout`(*layout*)

Set the page layout

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

Valid layouts are:

`/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
`/TwoColumnRight`	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

`setPageMode`(*mode*)

Set the page mode.

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

Valid modes are:

`/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

`updatePageFormFieldValues`(*page*, *fields*)

Update the form field values for a given page from a fields dictionary. Copy field texts and values from fields to page.

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)

`write`(*stream*)

Writes 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.

The DocumentInformation Class

class `PyPDF2.pdf.DocumentInformation`

A class representing the basic document metadata provided in a PDF File. This class is accessible through `getDocumentInfo()`

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.

`author`

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

`author_raw`

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

`creator`

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.

`creator_raw`

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

`producer`

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.

`producer_raw`

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

`subject`

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

`subject_raw`

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

`title`

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

`title_raw`

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

Example Usage:

```
>>> from PyPDF2 import PdfFileReader
>>> inputPdf = PdfFileReader(open("test.pdf", "rb"))
>>> docInfo = inputPdf.getDocumentInfo()
>>> docInfo.author
Anonymous
>>> docInfo.creator
Hewlett Packard MFP
>>> docInfo.producer
Acrobat Distiller 10.0.0 (Windows)
>>> docInfo.title
A Test
>>> docInfo.subject
testing
```

The XmpInformation Class

class `PyPDF2.xmp.XmpInformation`(*stream*)

An object that represents Adobe XMP metadata. Usually accessed by `getXmpMetadata()`

`custom_properties`

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

`dc_contributor`

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

`dc_coverage`

Text describing the extent or scope of the resource.

`dc_creator`

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

`dc_date`

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

`dc_description`

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

`dc_format`

The mime-type of the resource.

`dc_identifier`

Unique identifier of the resource.

`dc_language`

An unordered array specifying the languages used in the resource.

`dc_publisher`

An unordered array of publisher names.

`dc_relation`

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

`dc_rights`

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

`dc_source`

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

`dc_subject`

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

`dc_title`

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

`dc_type`

An unordered array of textual descriptions of the document type.

`pdf_keywords`

An unformatted text string representing document keywords.

`pdf_pdfversion`

The PDF file version, for example 1.0, 1.3.

`pdf_producer`

The name of the tool that created the PDF document.

`xmp_createDate`

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

`xmp_creatorTool`

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

`xmp_metadataDate`

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.

`xmp_modifyDate`

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

`xmpmm_documentId`

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

`xmpmm_instanceId`¶

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

The Destination Class

class PyPDF2.generic.Destination(*title*, *page*, *typ*, **args*)

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** (*int*) – Page number of this destination.
- **typ** (*str*) – How the destination is displayed.
- **args** – Additional arguments may be necessary depending on the type.

Raises PdfReadError:
If destination type is invalid.

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]

bottom

Read-only property accessing the bottom vertical coordinate.

Return type: int, or None if not available.

left

Read-only property accessing the left horizontal coordinate.

Return type: int, or None if not available.

page

Read-only property accessing the destination page number.

Return type: int

right

Read-only property accessing the right horizontal coordinate.

Return type: int, or None if not available.

title

Read-only property accessing the destination title.

Return type: str

`top`

Read-only property accessing the top vertical coordinate.

Return type: int, or `None` if not available.

`typ`

Read-only property accessing the destination type.

Return type: str

`zoom`

Read-only property accessing the zoom factor.

Return type: int, or `None` if not available.

The RectangleObject Class

class `PyPDF2.generic.RectangleObject`(*arr*)

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

- `artBox`
- `bleedBox`
- `cropBox`
- `mediaBox`
- `trimBox`

`lowerLeft`

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

`lowerRight`

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

`upperLeft`

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

`upperRight`¶

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

The Field Class

class `PyPDF2.generic.Field`(*data*)

A class representing a field dictionary. This class is accessed through `getFields()`

`additionalActions`

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.

`altName`

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

`defaultValue`

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

`fieldType`

Read-only property accessing the type of this field.

`flags`

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

`kids`

Read-only property accessing the kids of this field.

`mappingName`

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

`name`

Read-only property accessing the name of this field.

`parent`

Read-only property accessing the parent of this field.

`value`

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

Easy Concatenation with `pdfcat`

PyPDF2 contains a growing variety of sample programs meant to demonstrate its features. It also contains useful scripts such as `pdfcat`, located within the `Scripts` folder. This script makes it easy to concatenate PDF files by using Python slicing syntax. Because we are slicing PDF pages, we refer to the slices as *page ranges*.

Page range expression examples:

`:`	all pages	`-1`	last page
`22`	just the 23rd page	`:-1`	all but the last page
`0:3`	the first three pages	`-2`	second-to-last page
`:3`	the first three pages	`-2:`	last two pages
`5:`	from the sixth page onward	`-3:-1`	third & second to last

The third *stride* or *step* number is also recognized:

`::2`	0 2 4 ... to the end
`1:10:2`	1 3 5 7 9
`::-1`	all pages in reverse order
`3:0:-1`	3 2 1 but not 0
`2::-1`	2 1 0

Usage for `pdfcat` is as follows:

```
>>> pdfcat [-h] [-o output.pdf] [-v] input.pdf [page_range...] ...
```

You can add as many input files as you like. You may also specify as many *page ranges* as needed for each file.

Optional arguments:

`-h, --help`	Show the help message and exit
`-o, --output`	Follow this argument with the output PDF file. Will be created if it doesn't exist.
`-v, --verbose`	Show *page ranges* as they are being read

Examples:

```
>>> pdfcat -o output.pdf head.pdf content.pdf :6 7: tail.pdf -1
```

Concatenates all of `head.pdf`, all but page seven of `content.pdf`, and the last page of `tail.pdf`, producing `output.pdf`.

```
>>> pdfcat chapter*.pdf >book.pdf
```

You can specify the output file by redirection.

```
>>> pdfcat chapter?.pdf chapter10.pdf >book.pdf
```

In case you don’t want chapter 10 before chapter 2.

Thanks to **Steve Witham** for this script!