pdf2image Documentation

Release latest

Edouard Belval

Apr 17, 2023

CONTENTS

pdf2image is a python module that wraps the pdftoppm and pdftocairo utilities to convert PDF into images.

If you are new to the project, start with the installation section!

CHAPTER

ONE

INSTALLATION

1.1 Official package

pdf2image has a pip package with a matching name.

```
pip install pdf2image
```

1.2 From source

If you want to add a new language The easiest way to use the tool is by cloning the official repo.

```
git clone https://github.com/Belval/pdf2image
```

Then install the package with `python3 setup.py install`

1.3 Installing poppler

Poppler is the underlying project that does the magic in pdf2image. You can check if you already have it installed by calling `pdftoppm -h` in your terminal/cmd.

1.3.1 Ubuntu

```
sudo apt-get install poppler-utils
```

1.3.2 Archlinux

```
sudo pacman -S poppler
```

1.3.3 MacOS

```
brew install poppler
```

1.3.4 Windows

1. Download the latest poppler package from @oschwartz10612 version which is the most up-to-date.
2. Move the extracted directory to the desired place on your system
3. Add the `bin/` directory to your PATH
4. Test that all went well by opening `cmd` and making sure that you can call `pdftoppm -h`

CHAPTER

TWO

OVERVIEW

pdf2image subscribes to the Unix philosophy of "Do one thing and do it well", and is only used to convert PDF into images.

You can convert from a path or from bytes with aptly named `convert_from_path` and `convert_from_bytes`.

```
from pdf2image import convert_from_path, convert_from_bytes

images = convert_from_path("/home/user/example.pdf")

# OR

with open("/home/user/example.pdf") as pdf:
    images = convert_from_bytes(pdf.read())
```

This is the most basic usage, but the converted images will exist in memory and that may not be what you want since you can exhaust resources quickly with big PDF.

Instead, use an `output_folder` to avoid using the memory directly. The images will stil be readable and Pillow takes care of loading them on demand.

```
import tempfile

from pdf2image import convert_from_path


with tempfile.TemporaryDirectory() as path:
    images_from_path = convert_from_path("/home/user/example.pdf", output_folder=path)
```

Got it? Now by default `pdf2image` uses PPM as its file format. While the logic if abstracted by Pillow, this is still a raw file format that has no compression and is therefore quite big. Why not use good old JPEG?

```
images_from_path = convert_from_path("/home/user/example.pdf", fmt="jpeg")
```

Supported file formats are jpeg, png, tiff and ppm.

For a more in depth description of every parameters, see the *reference page*.

CHAPTER

THREE

LIMITATIONS / KNOWN ISSUES

3.1 DocuSign PDFs

If you have this error:

```
pdf2image.exceptions.PDFPageCountError: Unable to get page count.
Syntax Error: Gen inside xref table too large (bigger than INT_MAX)
Syntax Error: Invalid XRef entry 3
Syntax Error: Top-level pages object is wrong type (null)
Command Line Error: Wrong page range given: the first page (1) can not be after the last␣
↪page (0).
```

You are possibly using an old version of poppler. The solution is to update to the latest version. Similarly, if you are working with Docker (Debian 11 Image), maybe you can not update poppler because is not available. So, you have to use an image in ubuntu, install Python and then what you need.

More details here.

CHAPTER

FOUR

REFERENCE

4.1 Main functions

pdf2image is a light wrapper for the poppler-utils tools that can convert your PDFs into Pillow images.

pdf2image.pdf2image.**convert_from_bytes**(*pdf_file: bytes*, *dpi: int = 200*, *output_folder: ~typing.Union[str, ~pathlib.PurePath] = None*, *first_page: int = None*, *last_page: int = None*, *fmt: str = 'ppm'*, *jpegopt: ~typing.Dict = None*, *thread_count: int = 1*, *userpw: str = None*, *ownerpw: str = None*, *use_cropbox: bool = False*, *strict: bool = False*, *transparent: bool = False*, *single_file: bool = False*, *output_file: ~typing.Union[str, ~pathlib.PurePath] = <pdf2image.generators.ThreadSafeGenerator object>*, *poppler_path: ~typing.Union[str, ~pathlib.PurePath] = None*, *grayscale: bool = False*, *size: ~typing.Union[~typing.Tuple, int] = None*, *paths_only: bool = False*, *use_pdftocairo: bool = False*, *timeout: int = None*, *hide_annotations: bool = False*) → List[Image]

Function wrapping pdftoppm and pdftocairo.

Parameters

- **pdf_bytes** (*bytes*) – Bytes of the PDF that you want to convert
- **dpi** (*int, optional*) – Image quality in DPI (default 200), defaults to 200
- **output_folder** (*Union[str, PurePath], optional*) – Write the resulting images to a folder (instead of directly in memory), defaults to None
- **first_page** (*int, optional*) – First page to process, defaults to None
- **last_page** (*int, optional*) – Last page to process before stopping, defaults to None
- **fmt** (*str, optional*) – Output image format, defaults to "ppm"
- **jpegopt** (*Dict, optional*) – jpeg options *quality*, *progressive*, and *optimize* (only for jpeg format), defaults to None
- **thread_count** (*int, optional*) – How many threads we are allowed to spawn for processing, defaults to 1
- **userpw** (*str, optional*) – PDF's password, defaults to None
- **ownerpw** (*str, optional*) – PDF's owner password, defaults to None
- **use_cropbox** (*bool, optional*) – Use cropbox instead of mediabox, defaults to False

- **strict** (*bool, optional*) – When a Syntax Error is thrown, it will be raised as an Exception, defaults to False
- **transparent** (*bool, optional*) – Output with a transparent background instead of a white one, defaults to False
- **single_file** (*bool, optional*) – Uses the -singlefile option from pdftoppm/pdftocairo, defaults to False
- **output_file** (*Any, optional*) – What is the output filename or generator, defaults to uuid_generator()
- **poppler_path** (*Union[str, PurePath], optional*) – Path to look for poppler binaries, defaults to None
- **grayscale** (*bool, optional*) – Output grayscale image(s), defaults to False
- **size** (*Union[Tuple, int], optional*) – Size of the resulting image(s), uses the Pillow (width, height) standard, defaults to None
- **paths_only** (*bool, optional*) – Don't load image(s), return paths instead (requires output_folder), defaults to False
- **use_pdftocairo** (*bool, optional*) – Use pdftocairo instead of pdftoppm, may help performance, defaults to False
- **timeout** (*int, optional*) – Raise PDFPopplerTimeoutError after the given time, defaults to None
- **hide_annotations** (*bool, optional*) – Hide PDF annotations in the output, defaults to False

Raises

- **NotImplementedError** – Raised when conflicting parameters are given (hide_annotations for pdftocairo)
- ***PDFPopplerTimeoutError*** – Raised after the timeout for the image processing is exceeded
- ***PDFSyntaxError*** – Raised if there is a syntax error in the PDF and strict=True

Returns
A list of Pillow images, one for each page between first_page and last_page

Return type
List[Image.Image]

pdf2image.pdf2image.**convert_from_path**(*pdf_path: ~typing.Union[str, ~pathlib.PurePath], dpi: int = 200, output_folder: ~typing.Union[str, ~pathlib.PurePath] = None, first_page: int = None, last_page: int = None, fmt: str = 'ppm', jpegopt: ~typing.Dict = None, thread_count: int = 1, userpw: str = None, ownerpw: str = None, use_cropbox: bool = False, strict: bool = False, transparent: bool = False, single_file: bool = False, output_file: ~typing.Any = <pdf2image.generators.ThreadSafeGenerator object>, poppler_path: ~typing.Union[str, ~pathlib.PurePath] = None, grayscale: bool = False, size: ~typing.Union[~typing.Tuple, int] = None, paths_only: bool = False, use_pdftocairo: bool = False, timeout: int = None, hide_annotations: bool = False*) → List[Image]

Function wrapping pdftoppm and pdftocairo

Parameters

- **pdf_path** (*Union[str, PurePath]*) – Path to the PDF that you want to convert
- **dpi** (*int, optional*) – Image quality in DPI (default 200), defaults to 200
- **output_folder** (*Union[str, PurePath], optional*) – Write the resulting images to a folder (instead of directly in memory), defaults to None
- **first_page** (*int, optional*) – First page to process, defaults to None
- **last_page** (*int, optional*) – Last page to process before stopping, defaults to None
- **fmt** (*str, optional*) – Output image format, defaults to "ppm"
- **jpegopt** (*Dict, optional*) – jpeg options *quality*, *progressive*, and *optimize* (only for jpeg format), defaults to None
- **thread_count** (*int, optional*) – How many threads we are allowed to spawn for processing, defaults to 1
- **userpw** (*str, optional*) – PDF's password, defaults to None
- **ownerpw** (*str, optional*) – PDF's owner password, defaults to None
- **use_cropbox** (*bool, optional*) – Use cropbox instead of mediabox, defaults to False
- **strict** (*bool, optional*) – When a Syntax Error is thrown, it will be raised as an Exception, defaults to False
- **transparent** (*bool, optional*) – Output with a transparent background instead of a white one, defaults to False
- **single_file** (*bool, optional*) – Uses the -singlefile option from pdftoppm/pdftocairo, defaults to False
- **output_file** (*Any, optional*) – What is the output filename or generator, defaults to uuid_generator()
- **poppler_path** (*Union[str, PurePath], optional*) – Path to look for poppler binaries, defaults to None
- **grayscale** (*bool, optional*) – Output grayscale image(s), defaults to False
- **size** (*Union[Tuple, int], optional*) – Size of the resulting image(s), uses the Pillow (width, height) standard, defaults to None
- **paths_only** (*bool, optional*) – Don't load image(s), return paths instead (requires output_folder), defaults to False
- **use_pdftocairo** (*bool, optional*) – Use pdftocairo instead of pdftoppm, may help performance, defaults to False
- **timeout** (*int, optional*) – Raise PDFPopplerTimeoutError after the given time, defaults to None
- **hide_annotations** (*bool, optional*) – Hide PDF annotations in the output, defaults to False

Raises

- **NotImplementedError** – Raised when conflicting parameters are given (hide_annotations for pdftocairo)
- *PDFPopplerTimeoutError* – Raised after the timeout for the image processing is exceeded
- *PDFSyntaxError* – Raised if there is a syntax error in the PDF and strict=True

Returns
A list of Pillow images, one for each page between first_page and last_page

Return type
List[Image.Image]

pdf2image.pdf2image.**pdfinfo_from_bytes**(*pdf_bytes: bytes*, *userpw: str = None*, *ownerpw: str = None*, *poppler_path: str = None*, *rawdates: bool = False*, *timeout: int = None*) → Dict

Function wrapping poppler's pdfinfo utility and returns the result as a dictionary.

Parameters

- **pdf_bytes** (*bytes*) – Bytes of the PDF that you want to convert
- **userpw** (*str, optional*) – PDF's password, defaults to None
- **ownerpw** (*str, optional*) – PDF's owner password, defaults to None
- **poppler_path** (*Union[str, PurePath], optional*) – Path to look for poppler binaries, defaults to None
- **rawdates** (*bool, optional*) – Return the undecoded data strings, defaults to False
- **timeout** (*int, optional*) – Raise PDFPopplerTimeoutError after the given time, defaults to None

Returns
Dictionary containing various information on the PDF

Return type
Dict

pdf2image.pdf2image.**pdfinfo_from_path**(*pdf_path: str*, *userpw: str = None*, *ownerpw: str = None*, *poppler_path: str = None*, *rawdates: bool = False*, *timeout: int = None*) → Dict

Function wrapping poppler's pdfinfo utility and returns the result as a dictionary.

Parameters

- **pdf_path** (*str*) – Path to the PDF that you want to convert
- **userpw** (*str, optional*) – PDF's password, defaults to None
- **ownerpw** (*str, optional*) – PDF's owner password, defaults to None
- **poppler_path** (*Union[str, PurePath], optional*) – Path to look for poppler binaries, defaults to None
- **rawdates** (*bool, optional*) – Return the undecoded data strings, defaults to False
- **timeout** (*int, optional*) – Raise PDFPopplerTimeoutError after the given time, defaults to None

Raises

- *PDFPopplerTimeoutError* – Raised after the timeout for the image processing is exceeded
- *PDFInfoNotInstalledError* – Raised if pdfinfo is not installed
- *PDFPageCountError* – Raised if the output could not be parsed

Returns
Dictionary containing various information on the PDF

Return type
Dict

4.2 Exceptions

Define exceptions specific to pdf2image

exception pdf2image.exceptions.**PDFInfoNotInstalledError**
Raised when pdfinfo is not installed

exception pdf2image.exceptions.**PDFPageCountError**
Raised when the pdfinfo was unable to retrieve the page count

exception pdf2image.exceptions.**PDFPopplerTimeoutError**
Raised when the timeout is exceeded while converting a PDF

exception pdf2image.exceptions.**PDFSyntaxError**
Raised when a syntax error was thrown during rendering

exception pdf2image.exceptions.**PopplerNotInstalledError**
Raised when poppler is not installed

4.3 Parsers

pdf2image custom buffer parsers

pdf2image.parsers.**parse_buffer_to_jpeg**(*data: bytes*) → List[Image]
Parse JPEG file bytes to Pillow Image

Parameters
data (*bytes*) – pdftoppm/pdftocairo output bytes

Returns
List of JPEG images parsed from the output

Return type
List[Image.Image]

pdf2image.parsers.**parse_buffer_to_pgm**(*data: bytes*) → List[Image]
Parse PGM file bytes to Pillow Image

Parameters
data (*bytes*) – pdftoppm/pdftocairo output bytes

Returns
List of PGM images parsed from the output

Return type
List[Image.Image]

pdf2image.parsers.**parse_buffer_to_png**(*data: bytes*) → List[Image]
Parse PNG file bytes to Pillow Image

Parameters
data (*bytes*) – pdftoppm/pdftocairo output bytes

Returns
List of PNG images parsed from the output

Return type
List[Image.Image]

pdf2image.parsers.**parse_buffer_to_ppm**(*data: bytes*) → List[Image]

Parse PPM file bytes to Pillow Image

Parameters
data (*bytes*) – pdftoppm/pdftocairo output bytes

Returns
List of PPM images parsed from the output

Return type
List[Image.Image]

PYTHON MODULE INDEX

p

INDEX

C

M

P