Belval / pdf2image

A python module that wraps the pdftoppm utility to convert PDF to PIL Image object

#pdf #pil #pil-image #convert #poppler

109 commits | **3** branches | **2** releases | **7** contributors | MIT

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find File** | **Clone or download**

Belval Update README.md — Latest commit 5875594 3 days ago

.github	Exempt enhancement-labelled issues from stale-bot	7 months ago
pdf2image	Handle first_page greater than last_page (#66)	2 months ago
tests	Add strict parameter	6 months ago
.codecov.yml	Ignore tests.py in code coverage	11 months ago
.gitignore	Add multithreading and update README.md	last year
.travis.yml	Drop Python 3.3 support	2 months ago
LICENSE	Initial commit	2 years ago
MANIFEST.in	Fix error when importing package	2 years ago
README.md	Update README.md	3 days ago
setup.cfg	Version 0.1	2 years ago
setup.py	Handle first_page greater than last_page (#66)	2 months ago
tests.py	Handle first_page greater than last_page (#66)	2 months ago

README.md

pdf2image

A python 2.7 and 3.4+ module that wraps pdftoppm and pdftocairo to convert PDF to a PIL Image object

How to install

First you need poppler-utils

pdftoppm and pdftocairo are the piece of software that do the actual magic. It is distributed as part of a greater package called poppler.

Using `pip`

Windows users will have to install poppler for Windows, then add the `bin/` folder to PATH.

Mac users will have to install poppler for Mac.

Linux users will have both tools pre-installed with Ubuntu 16.04+ and Archlinux. If it's not, run `sudo apt install poppler-utils`

Using `conda`

```
conda install -c conda-forge poppler
```

Then you can install the pip package!

```
pip install pdf2image
```

Install `Pillow` if you don't have it already with `pip install pillow`

How does it work?

```
from pdf2image import convert_from_path, convert_from_bytes
```

```
from pdf2image.exceptions import (
    PDFInfoNotInstalledError,
    PDFPageCountError,
    PDFSyntaxError
)
```

Then simply do:

```
images = convert_from_path('/home/kankroc/example.pdf')
```

OR

```
images = convert_from_bytes(open('/home/kankroc/example.pdf', 'rb').read())
```

OR better yet

```
import tempfile

with tempfile.TemporaryDirectory() as path:
     images_from_path = convert_from_path('/home/kankroc/example.pdf', output_folder=path)
     # Do something here
```

`images` will be a list of PIL Image representing each page of the PDF document.

Here are the definitions:

```
convert_from_path(pdf_path, dpi=200, output_folder=None, first_page=None, last_page=None, fmt='ppm', thread_count=1, userpw=None, use_cropbox=False, strict=False, transparent=False, poppler_path=None)
```

```
convert_from_bytes(pdf_file, dpi=200, output_folder=None, first_page=None, last_page=None, fmt='ppm', thread_count=1, userpw=None, use_cropbox=False, strict=False, transparent=False, poppler_path=None)
```

What's new?

- Allow the user to specify poppler's installation path with `poppler_path`
- Fixed a bug where PNGs buffer with a non-terminating I-E-N-D sequence would throw an exception
- Fixed a bug that left open file descriptors when using `convert_from_bytes()` (Thank you @FabianUken)
- `fmt='tiff'` parameter allows you to create .tiff files (You need pdftocairo for this)
- `transparent` parameter allows you to generate images with no background instead of the usual white one (You need pdftocairo for this)
- `strict` parameter allows you to catch pdftoppm syntax error with a custom type `PDFSyntaxError`
- `use_cropbox` parameter allows you to use the crop box instead of the media box when converting (`-cropbox` in pdftoppm's CLI)
- `userpw` parameter allows you to set a password to unlock the converted PDF (`-upw` in pdftoppm's CLI)

Performance tips

- Using an output folder is significantly faster if you are using an SSD. Otherwise i/o usually becomes the bottleneck.
- Using multiple threads can give you some gains but avoid more than 4 as this will cause i/o bottleneck (even on my NVMe SSD!).
- If i/o is your bottleneck, using the JPEG format can lead to significant gains.
- PNG format is pretty slow, this is because of the compression.

- If you want to know the best settings (most settings will be fine anyway) you can clone the project and run `python tests.py` to get timings.

Limitations / known issues

- A relatively big PDF will use up all your memory and cause the process to be killed (unless you use an output folder)