ocrmypdf Documentation

Release 12.7.0

James R. Barlow

2021-10-12

CONTENTS

1 Introduction 3

2 Release notes 7

3 Installing OCRmyPDF 53

4 PDF optimization 67

5 Installing additional language packs 69

6 Installing the JBIG2 encoder 71

7 Cookbook 73

8 OCRmyPDF Docker image 79

9 Advanced features 83

10 Batch processing 89

11 Performance 97

12 PDF security issues 99

13 Common error messages 103

14 Using the OCRmyPDF API 105

15 Plugins 109

16 API Reference 119

17 Contributing guidelines 127

18 Indices and tables 129

Python Module Index 131

Index 133

OCRmyPDF adds an optical character recognition (OCR) text layer to scanned PDF files, allowing them to be searched.

PDF is the best format for storing and exchanging scanned documents. Unfortunately, PDFs can be difficult to modify. OCRmyPDF makes it easy to apply image processing and OCR to existing PDFs.

CHAPTER

ONE

INTRODUCTION

OCRmyPDF is an application and library that adds text "layers" to images in PDFs, making scanned image PDFs searchable. It uses OCR to guess what text is contained in images. It is written in Python. OCRmyPDF supports plugins that allow customization of its processing steps, and is very tolerant of PDFs that contain scanned images and "born digital" content that needs no text recognition.

1.1 About OCR

Optical character recognition is technology that converts images of typed or handwritten text, such as in a scanned document, to computer text that can be selected, searched and copied.

OCRmyPDF uses Tesseract, the best available open source OCR engine, to perform OCR.

1.2 About PDFs

PDFs are page description files that attempts to preserve a layout exactly. They contain vector graphics that can contain raster objects such as scanned images. Because PDFs can contain multiple pages (unlike many image formats) and can contain fonts and text, it is a good format for exchanging scanned documents.

A PDF page might contain multiple images, even if it only appears to have one image. Some scanners or scanning software will segment pages into monochromatic text and color regions for example, to improve the compression ratio and appearance of the page.

Rasterizing a PDF is the process of generating corresponding raster images. OCR engines like Tesseract work with images, not scalable vector graphics or mixed raster-vector-text graphics such as PDF.

1.3 About PDF/A

PDF/A is an ISO-standardized subset of the full PDF specification that is designed for archiving (the 'A' stands for Archive). PDF/A differs from PDF primarily by omitting features that would make it difficult to read the file in the future, such as embedded Javascript, video, audio and references to external fonts. All fonts and resources needed to interpret the PDF must be contained within it. Because PDF/A disables Javascript and other types of embedded content, it is probably more secure.

There are various conformance levels and versions, such as "PDF/A-2b".

Generally speaking, the best format for scanned documents is PDF/A. Some governments and jurisdictions, US Courts in particular, mandate the use of PDF/A for scanned documents.

Since most people who scan documents are interested in reading them indefinitely into the future, OCRmyPDF generates PDF/A-2b by default.

PDF/A has a few drawbacks. Some PDF viewers include an alert that the file is a PDF/A, which may confuse some users. It also tends to produce larger files than PDF, because it embeds certain resources even if they are commonly available. PDF/A files can be digitally signed, but may not be encrypted, to ensure they can be read in the future. Fortunately, converting from PDF/A to a regular PDF is trivial, and any PDF viewer can view PDF/A.

1.4 What OCRmyPDF does

OCRmyPDF analyzes each page of a PDF to determine the colorspace and resolution (DPI) needed to capture all of the information on that page without losing content. It uses Ghostscript to rasterize the page, and then performs on OCR the rasterized image to create an OCR "layer". The layer is then grafted back onto the original PDF.

While one can use a program like Ghostscript or ImageMagick to get an image and put the image through Tesseract, that actually creates a new PDF and many details may be lost. OCRmyPDF can produce a minimally changed PDF as output.

OCRmyPDF also provides some image processing options, like deskew, which improves the appearance of files and quality of OCR. When these are used, the OCR layer is grafted onto the processed image instead.

By default, OCRmyPDF produces archival PDFs – PDF/A, which are a stricter subset of PDF features designed for long term archives. If regular PDFs are desired, this can be disabled with `--output-type pdf`.

1.5 Why you shouldn't do this manually

A PDF is similar to an HTML file, in that it contains document structure along with images. Sometimes a PDF does nothing more than present a full page image, but often there is additional content that would be lost.

A manual process could work like either of these:

1. Rasterize each page as an image, OCR the images, and combine the output into a PDF. This preserves the layout of each page, but resamples all images (possibly losing quality, increasing file size, introducing compression artifacts, etc.).
2. Extract each image, OCR, and combine the output into a PDF. This loses the context in which images are used in the PDF, meaning that cropping, rotation and scaling of pages may be lost. Some scanned PDFs use multiple images segmented into black and white, grayscale and color regions, with stencil masks to prevent overlap, as this can enhance the appearance of a file while reducing file size. Clearly, reassembling these images will be easy. This also loses and text or vector art on any pages in a PDF with both scanned and pure digital content.

In the case of a PDF that is nothing other than a container of images (no rotation, scaling, cropping, one image per page), the second approach can be lossless.

OCRmyPDF uses several strategies depending on input options and the input PDF itself, but generally speaking it rasterizes a page for OCR and then grafts the OCR back onto the original. As such it can handle complex PDFs and still preserve their contents as much as possible.

OCRmyPDF also supports a many, many edge cases that have cropped over several years of development. We support PDF features like images inside of Form XObjects, and pages with UserUnit scaling. We support rare image formats like non-monochrome 1-bit images. We warn about files you may not to OCR. Thanks to pikepdf and QPDF, we auto-repair PDFs that are damaged. (Not that you need to know what any of these are! You should be able to throw any PDF at it.)

1.6 Limitations

OCRmyPDF is limited by the Tesseract OCR engine. As such it experiences these limitations, as do any other programs that rely on Tesseract:

- The OCR is not as accurate as commercial OCR solutions.
- It is not capable of recognizing handwriting.
- It may find gibberish and report this as OCR output.
- If a document contains languages outside of those given in the `-l LANG` arguments, results may be poor.
- It is not always good at analyzing the natural reading order of documents. For example, it may fail to recognize that a document contains two columns, and may try to join text across columns.
- Poor quality scans may produce poor quality OCR. Garbage in, garbage out.
- It does not expose information about what font family text belongs to.

OCRmyPDF is also limited by the PDF specification:

- PDF encodes the position of text glyphs but does not encode document structure. There is no markup that divides a document in sections, paragraphs, sentences, or even words (since blank spaces are not represented). As such all elements of document structure including the spaces between words must be derived heuristically. Some PDF viewers do a better job of this than others.
- Because some popular open source PDF viewers have a particularly hard time with spaces between words, OCRmyPDF appends a space to each text element as a workaround (when using `--pdf-renderer hocr`). While this mixes document structure with graphical information that ideally should be left to the PDF viewer to interpret, it improves compatibility with some viewers and does not cause problems for better ones.

Ghostscript also imposes some limitations:

- PDFs containing JBIG2-encoded content will be converted to CCITT Group4 encoding, which has lower compression ratios, if Ghostscript PDF/A is enabled.
- PDFs containing JPEG 2000-encoded content will be converted to JPEG encoding, which may introduce compression artifacts, if Ghostscript PDF/A is enabled.
- Ghostscript may transcode grayscale and color images, either lossy to lossless or lossless to lossy, based on an internal algorithm. This behavior can be suppressed by setting `--pdfa-image-compression` to `jpeg` or `lossless` to set all images to one type or the other. Ghostscript has no option to maintain the input image's format. (Ghostscript 9.25+ can copy JPEG images without transcoding them; earlier versions will transcode.)
- Ghostscript's PDF/A conversion removes any XMP metadata that is not one of the standard XMP metadata namespaces for PDFs. In particular, PRISM Metdata is removed.
- Ghostscript's PDF/A conversion seems to remove or deactivate hyperlinks and other active content.

You can use `--output-type pdf` to disable PDF/A conversion and produce a standard, non-archival PDF.

Regarding OCRmyPDF itself:

- PDFs that use transparency are not currently represented in the test suite

1.7 Similar programs

To the author's knowledge, OCRmyPDF is the most feature-rich and thoroughly tested command line OCR PDF conversion tool. If it does not meet your needs, contributions and suggestions are welcome. If not, consider one of these similar open source programs:

- pdf2pdfocr
- pdfsandwich

Ghostscript recently added three "pdfocr" output devices. They work by rasterizing all content and converting all pages to a single colour space.

1.8 Web front-ends

The Docker image `ocrmypdf` provides a web service front-end that allows files to submitted over HTTP and the results "downloaded". This is an HTTP server intended to simplify web services deployments; it is not intended to be deployed on the public internet and no real security measures to speak of.

In addition, the following third-party integrations are available:

- Nextcloud OCR is a free software plugin for the Nextcloud private cloud software

OCRmyPDF is not designed to be secure against malware-bearing PDFs (see Using OCRmyPDF online). Users should ensure they comply with OCRmyPDF's licenses and the licenses of all dependencies. In particular, OCRmyPDF requires Ghostscript, which is licensed under AGPLv3.

CHAPTER

TWO

RELEASE NOTES

OCRmyPDF uses semantic versioning for its command line interface and its public API.

OCRmyPDF's output messages are not considered part of the stable interface - that is, output messages may be improved at any release level, so parsing them may be unreliable. Use the API to depend on precise behavior.

The public API may be useful in scripts that launch OCRmyPDF processes or that wish to use some of its features for working with PDFs.

Note: Python 3.6 reaches end of life on December 23, 2021. We will end support for Python 3.6 around that time. The change will be marked with a major release.

2.1 v12.7.0

- Fixed test suite failure when using pikepdf 3.2.0 that was compiled with pybind11 2.8.0. #843
- Improve advice to user about using `--max-image-mpixels` if OCR fails for this reason.
- Minor documentation fixes. (Thanks to @mara004.)
- Don't require importlib-metadata and importlib-resources backports on versions of Python where the standard library implementation is sufficient. (Thanks to Marco Genasci.)

2.2 v12.6.0

- Implemented `--output-type=none` to skip producing PDFs for applications that only want sidecar files (#787).
- Fixed ambiguities in descriptions of behavior of `--jbig2-lossy`.
- Various improvements to documentation.

2.3 v12.5.0

- Fixed build failure for the combination of PyPy 3.6 and pikepdf 3.0. This combination can work in a source build but does not work with wheels.
- Accepted bot that wanted to upgrade our deprecated requirements.txt.
- Documentation updates.
- Replace pkg_resources and install dependency on setuptools with importlib-metadata and importlib-resources.
- Fixed regression in hocrtransform causing text to be omitted when this renderer was used.
- Fixed some typing errors.

2.4 v12.4.0

- When grafting text layers, use pikepdf's `unparse_content_stream` if available.
- Confirmed support for pluggy 1.0. (Thanks @QuLogic.)
- Fixed some typing issues, improved pre-commit settings, and fixed issues flagged by linters.
- PyPy 7.3.3 (=Python 3.6) is now supported. Note that PyPy does not necessarily run faster, because the vast majority of OCRmyPDF's execution time is spent running OCR or generally executing native code. However, PyPy may bring speed improvements in some areas.

2.5 v12.3.3

- watcher.py: fixed interpretation of boolean env vars (#821).
- Adjust CI scripts to test Tesseract 5 betas.
- Document our support for the Tesseract 5 betas.

2.6 v12.3.2

- Indicate support for flask 2.x, watcher 2.x (#815#816).

2.7 v12.3.1

- Fixed issue with selection of text when using the hOCR renderer (#813).
- Fixed build errors with the Docker image by upgrading to a newer Ubuntu. Also set the timezone of this image to UTC.

2.8 v12.3.0

- Fixed a regression introduced in Pillow 8.3.0. Pillow no longer rounds DPI for image resolutions. We now account for this (#802).
- We no longer use some API calls that are deprecated in the latest versions of pikepdf.
- Improved error message when a language is requested that doesn't look like a typical ISO 639-2 code.
- Fixed some tests that attempted to symlink on Windows, breaking tests on a Windows desktop but not usually on CI.
- Documentation fixes (thanks to @mara004)

2.9 v12.2.0

- Fixed invalid Tesseract version number on Windows (#795).
- Documentation tweaks. Documentation build now depends on sphinx-issues package.

2.10 v12.1.0

- For security reasons we now require Pillow >= 8.2.x. (Older versions will continue to work if upgrading is not an option.)
- The build system was reorganized to rely on `setup.cfg` instead of `setup.py`. All changes should work with previously supported versions of setuptools.
- The files in `requirements/*` are now considered deprecated but will be retained for v12. Instead use `pip install ocrmypdf[test]` instead of `requirements/test.txt`, etc. These files will be removed in v13.

2.11 v12.0.3

- Expand the list of languages supported by the hocr PDF renderer. Several languages were previously considered not supported, particularly those non-European languages that use the Latin alphabet.
- Fixed a case where the exception stack trace was suppressed in verbose mode.
- Improved documentation around commercial OCR.

2.12 v12.0.2

- Fixed exception thrown when using `--remove-background` on files containing small images (#769).
- Improve documentation for description of adding language packs to the Docker image and corrected name of French language pack.

2.13 v12.0.1

- Fixed "invalid version number" for untagged tesseract versions (#770).

2.14 v12.0.0

Breaking changes

- Due to recent security issues in pikepdf, Pillow and reportlab, we now require newer versions of these libraries and some of their dependencies. (If necessary, package maintainers may override these versions at their discretion; lower versions will often work.)
- We now use the "LeaveColorUnchanged" color conversion strategy when directing Ghostscript to create a PDF/A. Generally this is faster than performing a color conversion, which is not always necessary.
- OCR text is now packaged in a Form XObject. This makes it easier to isolate OCR from other document content. However, some poorly implemented PDF text extraction algorithms may fail to detect the text.
- Many API functions have stricter parameter checking or expect keyword arguments were they previously did not.
- Some deprecated functions in `ocrmypdf.optimize` were removed.
- The `ocrmypdf.leptonica` module is now deprecated, due to difficulties with the current strategy of ABI binding on newer platforms like Apple Silicon. It will be removed and replaced, either by repackaging Leptonica as an independent library using or using a different image processing library.
- Continuous integration moved to GitHub Actions.
- We no longer depend on `pytest_helpers_namespace` for testing.

New features

- New plugin hook: `get_progressbar_class`, for progress reporting, allowing developers to replace the standard console progress bar with some other mechanism, such as updating a GUI progress bar.
- New plugin hook: `get_executor`, for replacing the concurrency model. This is primarily to support execution on AWS Lambda, which does not support standard Python `multiprocessing` due to its lack of shared memory.
- New plugin hook: `get_logging_console`, for replacing the standard way OCRmyPDF outputs its messages.
- New plugin hook: `filter_pdf_page`, for modifying individual PDF pages produced by OCRmyPDF.
- OCRmyPDF now runs on nonstandard execution environments that do not have interprocess semaphores, such as AWS Lambda and Android Termux. If the environment does not have semaphores, OCRmyPDF will automatically select an alternate process executor that does not use semaphores.
- Continuous integration moved to GitHub Actions.
- We now generate an ARM64-compatible Docker image alongside the x64 image. Thanks to @andkrause for doing most of the work in a pull request several months ago, which we were finally able to integrate now. Also thanks to @0x326 for review comments.

Fixes

- Fixed a possible deadlock on attempting to flush `sys.stderr` when older versions of Leptonica are in use.
- Some worker processes inherited resources from their parents such as log handlers that may have also lead to deadlocks. These resources are now released.
- Improvements to test coverage.
- Removed vestiges of support for Tesseract versions older than 4.0.0-beta1 (which ships with Ubuntu 18.04).

- OCRmyPDF can now parse all of Tesseract version numbers, since several schemes have been in use.
- Fixed an issue with parsing PDFs that contain images drawn at a scale of 0. (#761)
- Removed a frequently repeated message about disabling mmap.

2.15 v11.7.3

- Exclude CCITT Group 3 images from being optimized. Some libraries OCRmyPDF uses do not seem to handle this obscure compression format properly. You may get errors or possible corrupted output images without this fix.

2.16 v11.7.2

- Updated pinned versions in main.txt, primarily to upgrade Pillow to 8.1.2, due to recently disclosed security vulnerabilities in that software.
- The `--sidecar` parameter now causes an exception if set to the same file as the input or output PDF.

2.17 v11.7.1

- Some exceptions while attempting image optimization were only logged at the debug level, causing them to be suppressed. These errors are now logged appropriately.
- Improved the error message related to `--unpaper-args`.
- Updated documentation to mention the new conda distribution.

2.18 v11.7.0

- We now support using `--sidecar` in conjunction with `--pages`; these arguments used to be mutually exclusive. (#735)
- Fixed a possible issue with PDF/A-1b generation. Acrobat complained that our PDFs use object streams. More robust PDF/A validators like veraPDF don't consider this a problem, but we'll honor Acrobat's objection from here on. This may increase file size of PDF/A-1b files. PDF/A-2b files will not be affected.

2.19 v11.6.2

- Fixed a regression where the wrong page orientation would be produced when using arguments such as `--deskew --rotate-pages` (#730).

2.20 v11.6.1

- Fixed an issue with attempting optimize unusually narrow-width images by excluding these images from optimization (#732).
- Remove an obsolete compatibility shim for a version of pikepdf that is no longer supported.

2.21 v11.6.0

- OCRmyPDF will now automatically register plugins from the same virtual environment with an appropriate setuptools entrypoint.
- Refactor the plugin manager to remove unnecessary complications and make plugin registration more automatic.
- `PageContext` and `PdfContext` are now formally part of the API, as they should have been, since they were part of `ocrmypdf.pluginspec`.

2.22 v11.5.0

- Fixed an issue where the output page size might differ by a fractional amount due to rounding, when `--force-ocr` was used and the page contained objects with multiple resolutions.
- When determining the resolution at which to rasterize a page, we now consider printed text on the page as requiring a higher resolution. This fixes issues with certain pages being rendered with unacceptably low resolution text, but may increase output file sizes in some workflows where low resolution text is acceptable.
- Added a workaround to fix an exception that occurs when trying to `import ocrmypdf.leptonica` on Apple ARM silicon (or potentially, other platforms that do not permit write+executable memory).

2.23 v11.4.5

- Fixed an issue where files may not be closed when the API is used.
- Improved `setup.cfg` with better settings for test coverage.

2.24 v11.4.4

- Fixed `AttributeError: 'NoneType' object has no attribute 'userunit'` (#700), related to OCRmyPDF not properly forwarded an error message from pdfminer.six.
- Adjusted typing of some arguments.
- `ocrmypdf.ocr` now takes a `threading.Lock` for reasons outlined in the documentation.

2.25 v11.4.3

- Removed a redundant debug message.
- Test suite now asserts that most patched functions are called when they should be.
- Test suite now skips a test that fails on two particular versions of piekpdf.

2.26 v11.4.2

- Fixed support for Cygwin, hopefully.
- watcher.py: Fixed an issue with the OCR_LOGLEVEL not being interpreted.

2.27 v11.4.1

- Fixed an issue where invalid pages ranges passed using the `pages` argument, such as "1-0" would cause unhandled exceptions.
- Accepted a user-contributed to the Synology demo script in misc/synology.py.
- Clarified documentation about change of temporary file location `ocrmypdf.io`.
- Fixed Python wheel tag which was incorrectly set to py35 even though we long since dropped support for Python 3.5.

2.28 v11.4.0

- When looking for Tesseract and Ghostscript, we now check the Windows Registry to see if their installers registered the location of their executables. This should help Windows users who have installed these programs to non-standard locations.
- We now report on the progress of PDF/A conversion, since this operation is sometimes slow.
- Improved command line completions.
- The prefix of the temporary folder OCRmyPDF creates has been changed from `com.github.ocrmypdf` to `ocrmypdf.io`. Scripts that chose to depend on this prefix may need to be adjusted. (This has always been an implementation detail so is not considered part of the semantic versioning "contract".)
- Fixed #692, where a particular file with malformed fonts would flood an internal message cue by generating so many debug messages.
- Fixed an exception on processing hOCR files with no page record. Tesseract is not known to generate such files.

2.29 v11.3.4

- Fixed an error message 'called readLinearizationData for file that is not linearized' that may occur when pikepdf 2.1.0 is used. (Upgrading to pikepdf 2.1.1 also fixes the issue.)
- File watcher now automatically includes `.PDF` in addition to `.pdf` to better support case sensitive file systems.
- Some documentation and comment improvements.

2.30 v11.3.3

- If unpaper outputs non-UTF-8 data, quietly fix this rather than choke on the conversion. (Possibly addresses #671.)

2.31 v11.3.2

- Explicitly require pikepdf 2.0.0 or newer when running on Python 3.9. (There are concerns about the stability of pybind11 2.5.x with Python 3.9, which is used in pikepdf 1.x.)
- Fixed another issue related to page rotation.
- Fixed an issue where image marked as image masks were not properly considered as optimization candidates.
- On some systems, unpaper seems to be unable to process the PNGs we offer it as input. We now convert the input to PNM format, which unpaper always accepts. Fixes #665 and #667.
- DPI sent to unpaper is now rounded to a more reasonable number of decimal digits.
- Debug and error messages from unpaper were being suppressed.
- Some documentation tweaks.

2.32 v11.3.1

- Declare support for new versions: pdfminer.six 20201018 and pikepdf 2.x
- Fixed warning related to `--pdfa-image-compression` that appears at the wrong time.

2.33 v11.3.0

- The "OCR" step is describing as "Image processing" in the output messages when OCR is disabled, to better explain the application's behavior.
- Debug logs are now only created when run as a command line, and not when OCR is performed for an API call. It is the calling application's responsibility to set up logging.
- For PDFs with a low number of pages, we gathered information about the input PDF in a thread rather than process (when there are more pages). When run as a thread, we did not close the file handle to the working PDF, leaking one file handle per call of `ocrmypdf.ocr`.
- Fixed an issue where debug messages send by child worker processes did not match the log settings of parent process, causing messages to be dropped. This affected macOS and Windows only where the parent process is not forked.

- Fixed the hookspec of rasterize_pdf_page to remove default parameters that were not handled in an expected way by pluggy.
- Fixed another issue with automatic page rotation (#658) due to the issue above.

2.34 v11.2.1

- Fixed an issue where optimization of a 1-bit image with a color palette or associated ICC that was optimized to JBIG2 could have its colors inverted.

2.35 v11.2.0

- Fixed an issue with optimizing PNG-type images that had soft masks or image masks. This is a regression introduced in (or about) v11.1.0.
- Improved type checking of the `plugins` parameter for the `ocrmypdf.ocr` API call.

2.36 v11.1.2

- Fixed hOCR renderer writing the text in roughly reverse order. This should not affect reasonably smart PDF readers that properly locate the position of all text, but may confuse those that rely on the order of objects in the content stream. (#642)

2.37 v11.1.1

- We now avoid using named temporary files when using pngquant allowing containerized pngquant installs to be used.
- Clarified an error message.
- Highest number of 1's in a release ever!

2.38 v11.1.0

- Fixed page rotation issues: #634#589.
- Fixed some cases where optimization created an invalid image such as a 1-bit "RGB" image: #629#620.
- Page numbers are now displayed in debug logs when pages are being grafted.
- ocrmypdf.optimize.rewrite_png and ocrmypdf.optimize.rewrite_png_as_g4 were marked deprecated. Strictly speaking these should have been internal APIs, but they were never hidden.
- As a precaution, pikepdf mmap-based file access has been disabled due to a rare race condition that causes a crash when certain objects are deallocated. The problem is likely in pikepdf's dependency pybind11.
- Extended the example plugin to demonstrate conversion to mono.

2.39 v11.0.2

- Fixed #612, TypeError exception. Fixed by eliminating unnecessary repair of input PDF metadata in memory.

2.40 v11.0.1

- Blacklist pdfminer.six 20200720, which has a regression fixed in 20200726.
- Approve img2pdf 0.4 as it passes tests.
- Clarify that the GPL-3 portion of pdfa.py was removed with the changes in v11.0.0; the debian/copyright file did not properly annotate this change.

2.41 v11.0.0

- Project license changed to Mozilla Public License 2.0. Some miscellaneous code is now under MIT license and non-code content/media remains under CC-BY-SA 4.0. License changed with approval of all people who were found to have contributed to GPLv3 licensed sections of the project. (#600)
- Because the license changed, this is being treated as a major version number change; however, there are no known breaking changes in functional behavior or API compared to v10.x.

2.42 v10.3.3

- Fixed a "KeyError: 'dpi'" error message when using `--threshold` on an image. (#607)

2.43 v10.3.2

- Fixed a case where we reported "no reason" for a file size increase, when we could determine the reason.
- Enabled support for pdfminer.six 20200726.

2.44 v10.3.1

- Fixed a number of test suite failures with pdfminer.six older than veresion 20200402.
- Enabled support for pdfminer.six 20200720.

2.45 v10.3.0

- Fixed an issue where we would consider images that were already JBIG2-encoded for optimization, potentially producing a less optimized image than the original. We do not believe this issue would ever cause an image to loss fidelity.
- Where available, pikepdf memory mapping is now used. This improves performance.
- When Leptonica 1.79+ is installed, use its new error handling API to avoid a "messy" redirection of stderr which was necessary to capture its error messages.
- For older versions of Leptonica, added a new thread level lock. This fixes a possible race condition in handling error conditions in Leptonica (although there is no evidence it ever caused issues in practice).
- Documentation improvements and more type hinting.

2.46 v10.2.1

- Disabled calculation of text box order with pdfminer. We never needed this result and it is expensive to calculate on files with complex pre-existing text.
- Fixed plugin manager to accept `Path(plugin)` as a path to a plugin.
- Fixed some typing errors.
- Documentation improvements.

2.47 v10.2.0

- Update Docker image to use Ubuntu 20.04.
- Fixed issue PDF/A acquires title "Untitled" after conversion. (#582)
- Fixed a problem where, when using `--pdf-renderer hocr`, some text would be missing from the output when using a more recent version of Tesseract. Tesseract began adding more detailed markup about the semantics of text that our HOCR transform did not recognize, so it ignored them. This option is not the default. If necessary `--redo-ocr` also redoing OCR to fix such issues.
- Fixed an error in Python 3.9 beta, due to removal of deprecated `Element.getchildren()`. (#584)
- Implemented support using the API with `BytesIO` and other file stream objects. (#545)

2.48 v10.1.1

- Fixed `OMP_THREAD_LIMIT` set to invalid value error messages on some input files. (The error was harmless, apart from less than optimal performance in some cases.)

2.49 v10.1.0

- Previously, we `--clean-final` would cause an unpaper-cleaned page image to be produced twice, which was necessary in some cases but not in general. We now take this optimization opportunity and reuse the image if possible.
- We now provide PNG files as input to unpaper, since it accepts them, instead of generating PPM files which can be very large. This can improve performance and temporary disk usage.
- Documentation updated for plugins.

2.50 v10.0.1

- Fixed regression when `-l lang1+lang2` is used from command line.

2.51 v10.0.0

Breaking changes

- Support for pdfminer.six version 20181108 has been dropped, along with a monkeypatch that made this version work.
- Output messages are now displayed in color (when supported by the terminal) and prefixes describing the severity of the message are removed. As such programs that parse OCRmyPDF's log message will need to be revised. (Please consider using OCRmyPDF as a library instead.)
- The minimum version for certain dependencies has increased.
- Many API changes; see developer changes.
- The Python libraries pluggy and coloredlogs are now required.

New features and improvements

- PDF page scanning is now parallelized across CPUs, speeding up this phase dramatically for files with a high page counts.
- PDF page scanning is optimized, addressing some performance regressions.
- PDF page scanning is no longer run on pages that are not selected when the `--pages` argument is used.
- PDF page scanning is now independent of Ghostscript, ending our past reliance on this occasionally unstable feature in Ghostscript.
- A plugin architecture has been added, currently allowing one to more easily use a different OCR engine or PDF renderer from Tesseract and Ghostscript, respectively. A plugin can also override some decisions, such changing the OCR settings after initial scanning.
- Colored log messages.

Developer changes

- The test spoofing mechanism, used to test correct handling of failures in Tesseract and Ghostscript, has been removed in favor of using plugins for testing. The spoofing mechanism was fairly complex and required many special hacks for Windows.
- Code describing the resolution in DPI of images was refactored into a `ocrmypdf.helpers.Resolution` class.
- The module `ocrmypdf._exec` is now private to OCRmyPDF.

- The `ocrmypdf.hocrtransform` module has been updated to follow PEP8 naming conventions.
- Ghostscript is no longer used for finding the location of text in PDFs, and APIs related to this feature have been removed.
- Lots of internal reorganization to support plugins.

2.52 v9.8.2

- Fixed an issue where OCRmyPDF would ignore text inside Form XObject when making certain decisions about whether a document already had text.
- Fixed file size increase warning to take overhead of small files into account.
- Added instructions for installing on Cygwin.

2.53 v9.8.1

- Fixed an issue where unexpected files in the `%PROGRAMFILES%\gs` directory (Windows) caused an exception.
- Mark pdfminer.six 20200517 as supported.
- If jbig2enc is missing and optimization is requested, a warning is issued instead of an error, which was the intended behavior.
- Documentation updates.

2.54 v9.8.0

- Fixed issue where only the first PNG (FlateDecode) image in a file would be considered for optimization. File sizes should be improved from here on.
- Fixed a startup crash when the chosen language was Japanese (#543).
- Added options to configure polling and log level to watcher.py.

2.55 v9.7.2

- Fixed an issue with `ocrmypdf.ocr(...language=)` not accepting a list of languages as documented.
- Updated setup.py to confirm that pdfminer.six version 20200402 is supported.

2.56 v9.7.1

- Fixed version check failing when used with qpdf 10.0.0.
- Added some missing type annotations.
- Updated documentation to warn about need for "ifmain" guard and Windows.

2.57 v9.7.0

- Fixed an error in watcher.py if `OCR_JSON_SETTINGS` was not defined.
- Ghostscript 9.51 is now blacklisted, due to numerous problems with this version.
- Added a workaround for a problem with "txtwrite" in Ghostscript 9.52.
- Fixed an issue where the incorrect number of threads used was shown when `OMP_THREAD_LIMIT` was manipulated.
- Removed a possible performance bottlenecks for files that use hundreds to thousands of images on the same page.
- Documentation improvements.
- Optimization will now be applied to some monochrome images that have a color profile defined instead of only black and white.
- ICC profiles are consulted when determining the simplified colorspace of an image.

2.58 v9.6.1

- Documentation improvements - thanks to many users for their contributions!
  - Fixed installation instructions for ArchLinux (@pigmonkey)
  - Updated installation instructions for FreeBSD and other OSes (@knobix)
  - Added instructions for using Docker Compose with watchdog (@ianalexander, @deisi)
  - Other miscellany (@mb720, @toy, @caiofacchinato)
  - Some scripts provided in the documentation have been migrated out so that they can be copied out as whole files, and to ensure syntax checking is maintained.
- Fixed an error that caused bash completions to fail on macOS. (#502#504; @AlexanderWillner)
- Fixed a rare case where OCRmyPDF threw an exception while processing a PDF with the wrong object type in its `/Trailer /Info`. The error is now logged and incorrect object is ignored. (#497)
- Removed potentially non-free file `enron1.pdf` and simplified the test that used it.
- Removed potentially non-free file `misc/media/logo.afdesign`.

2.59 v9.6.0

- Fixed a regression with transferring metadata from the input PDF to the output PDF in certain situations.
- pdfminer.six is now supported up to version 2020-01-24.
- Messages are explaining page rotation decisions are now shown at the standard verbosity level again when `--rotate-pages`. In some previous version they were set to debug level messages that only appeared with the parameter `-v1`.
- Improvements to `misc/watcher.py`. Thanks to @ianalexander and @svenihoney.
- Documentation improvements.

2.60 v9.5.0

- Added API functions to measure OCR quality.
- Modest improvements to handling PDFs with difficult/non compliant metadata.

2.61 v9.4.0

- Updated recommended dependency versions.
- Improvements to test coverage and changes to facilitate better measurement of test coverage, such as when tests run in subprocesses.
- Improvements to error messages when Leptonica is not installed correctly.
- Fixed use of pytest "session scope" that may have caused some intermittent CI failures.
- When the argument `--keep-temporary-files` or verbosity is set to `-v1`, a debug log file is generated in the working temporary folder.

2.62 v9.3.0

- Improved native Windows support: we now check in the obvious places in the "Program Files" folders installations of Tesseract and Ghostscript, rather than relying on the user to edit `PATH` to specify their location. The `PATH` environment variable can still be used to differentiate when multiple installations are present or the programs are installed to non- standard locations.
- Fixed an exception on parsing Ghostscript error messages.
- Added an improved example demonstrating how to set up a watched folder for automated OCR processing (thanks to @ianalexander for the contribution).

2.63 v9.2.0

- Native Windows is now supported.
- Continuous integration moved to Azure Pipelines.
- Improved test coverage and speed of tests.
- Fixed an issue where a page that was originally a JPEG would be saved as a PNG, increasing file size. This occurred only when a preprocessing option was selected along with `--output-type=pdf` and all images on the original page were JPEGs. Regression since v7.0.0.
- OCRmyPDF no longer depends on the QPDF executable `qpdf` or `libqpdf`. It uses pikepdf (which in turn depends on `libqpdf`). Package maintainers should adjust dependencies so that OCRmyPDF no longer calls for libqpdf on its own. For users of Python binary wheels, this change means a separate installation of QPDF is no longer necessary. This change is mainly to simplify installation on Windows.
- Fixed a rare case where log messages from Tesseract would be discarded.
- Fixed incorrect function signature for pixFindPageForeground, causing exceptions on certain platforms/Leptonica versions.

2.64 v9.1.1

- Expand the range of pdfminer.six versions that are supported.
- Fixed Docker build when using pikepdf 1.7.0.
- Fixed documentation to recommend using pip from get-pip.py.

2.65 v9.1.0

- Improved diagnostics when file size increases at output. Now warns if JBIG2 or pngquant were not available.
- pikepdf 1.7.0 is now required, to pick up changes that remove the need for a source install on Linux systems running Python 3.8.

2.66 v9.0.5

- The Alpine Docker image (jbarlow83/ocrmypdf-alpine) has been dropped due to the difficulties of supporting Alpine Linux.
- The primary Docker image (jbarlow83/ocrmypdf) has been improved to take on the extra features that used to be exclusive to the Alpine image.
- No changes to application code.
- pdfminer.six version 20191020 is now supported.

2.67 v9.0.4

- Fixed compatibility with Python 3.8 (but requires source install for the moment).
- Fixed Tesseract settings for `--user-words` and `--user-patterns`.
- Changed to pikepdf 1.6.5 (for Python 3.8).
- Changed to Pillow 6.2.0 (to mitigate a security vulnerability in earlier Pillow).
- A debug message now mentions when English is automatically selected if the locale is not English.

2.68 v9.0.3

- Embed an encoded version of the sRGB ICC profile in the intermediate Postscript file (used for PDF/A conversion). Previously we included the filename, which required Postscript to run with file access enabled. For security, Ghostscript 9.28 enables `-dSAFER` and as such, no longer permits access to any file by default. This fix is necessary for compatibility with Ghostscript 9.28.
- Exclude a test that sometimes times out and fails in continuous integration from the standard test suite.

2.69 v9.0.2

- The image optimizer now skips optimizing flate (PNG) encoded images in some situations where the optimization effort was likely wasted.
- The image optimizer now ignores images that specify arbitrary decode arrays, since these are rare.
- Fixed an issue that caused inversion of black and white in monochrome images. We are not certain but the problem seems to be linked to Leptonica 1.76.0 and older.
- Fixed some cases where the test suite failed if English or German Tesseract language packs were not installed.
- Fixed a runtime error if the Tesseract English language is not installed.
- Improved explicit closing of Pillow images after use.
- Actually fixed of Alpine Docker image build.
- Changed to pikepdf 1.6.3.

2.70 v9.0.1

- Fixed test suite failing when either of optional dependencies unpaper and pngquant were missing.
- Attempted fix of Alpine Docker image build.
- Documented that FreeBSD ports are now available.
- Changed to pikepdf 1.6.1.

2.71 v9.0.0

Breaking changes

- The `--mask-barcodes` experimental feature has been dropped due to poor reliability and occasional crashes, both due to the underlying library that implements this feature (Leptonica).
- The `-v` (verbosity level) parameter now accepts only `0`, `1`, and `2`.
- Dropped support for Tesseract 4.00.00-alpha releases. Tesseract 4.0 beta and later remain supported.
- Dropped the `ocrmypdf-polyglot` and `ocrmypdf-webservice` images.

New features

- Added a high level API for applications that want to integrate OCRmyPDF. Special thanks to Martin Wind (@mawi1988) whose made significant contributions to this effort.
- Added progress bars for long-running steps.
- We now create linearized ("fast web view") PDFs by default. The new parameter `--fast-web-view` provides control over when this feature is applied.
- Added a new `--pages` feature to limit OCR to only a specific page range. The list may contain commas or single pages, such as `1, 3, 5-11`.
- When the number of pages is small compared to the number of allowed jobs, we run Tesseract in multithreaded (OpenMP) mode when available. This should improve performance on files with low page counts.
- Removed dependency on `ruffus`, and with that, the non-reentrancy restrictions that previous made an API impossible.
- Output and logging messages overhauled so that ocrmypdf may be integrated into applications that use the logging module.
- pikepdf 1.6.0 is required.
- Added a logo.

Bug fixes

- Pages with vector artwork are treated as full color. Previously, vectors were ignored when considering the colorspace needed to cover a page, which could cause loss of color under certain settings.
- Test suite now spawns processes less frequently, allowing more accurate measurement of code coverage.
- Improved test coverage.
- Fixed a rare division by zero (if optimization produced an invalid file).
- Updated Docker images to use newer versions.
- Fixed images encoded as JBIG2 with a colorspace other than `/DeviceGray` were not interpreted correctly.
- Fixed a OCR text-image registration (i.e. alignment) problem when the page when MediaBox had a nonzero corner.

2.72 v8.3.2

- Dropped workaround for macOS that allowed it work without pdfminer.six, now a proper sdist release of pdfminer.six is available.
- pikepdf 1.5.0 is now required.

2.73 v8.3.1

- Fixed an issue where PDFs with malformed metadata would be rendered as blank pages. #398.

2.74 v8.3.0

- Improved the strategy for updating pages when a new image of the page was produced. We now attempt to preserve more content from the original file, for annotations in particular.
- For PDFs with more than 100 pages and a sequence where one PDF page was replaced and one or more subsequent ones were skipped, an intermediate file would be corrupted while grafting OCR text, causing processing to fail. This is a regression, likely introduced in v8.2.4.
- Previously, we resized the images produced by Ghostscript by a small number of pixels to ensure the output image size was an exactly what we wanted. Having discovered a way to get Ghostscript to produce the exact image sizes we require, we eliminated the resizing step.
- Command line completions for `bash` are now available, in addition to `fish`, both in `misc/completion`. Package maintainers, please install these so users can take advantage.
- Updated requirements.
- pikepdf 1.3.0 is now required.

2.75 v8.2.4

- Fixed a false positive while checking for a certain type of PDF that only Acrobat can read. We now more accurately detect Acrobat-only PDFs.
- OCRmyPDF holds fewer open file handles and is more prompt about releasing those it no longer needs.
- Minor optimization: we no longer traverse the table of contents to ensure all references in it are resolved, as changes to libqpdf have made this unnecessary.
- pikepdf 1.2.0 is now required.

2.76 v8.2.3

- Fixed that `--mask-barcodes` would occasionally leave a unwanted temporary file named `junkpixt` in the current working folder.
- Fixed (hopefully) handling of Leptonica errors in an environment where a non-standard `sys.stderr` is present.
- Improved help text for `--verbose`.

2.77 v8.2.2

- Fixed a regression from v8.2.0, an exception that occurred while attempting to report that `unpaper` or another optional dependency was unavailable.
- In some cases, `ocrmypdf [-c|--clean]` failed to exit with an error when `unpaper` is not installed.

2.78 v8.2.1

- This release was canceled.

2.79 v8.2.0

- A major improvement to our Docker image is now available thanks to hard work contributed by @mawi12345. The new Docker image, ocrmypdf-alpine, is based on Alpine Linux, and includes most of the functionality of three existed images in a smaller package. This image will replace the main Docker image eventually but for now all are being built. See documentation for details.
- Documentation reorganized especially around the use of Docker images.
- Fixed a problem with PDF image optimization, where the optimizer would unnecessarily decompress and recompress PNG images, in some cases losing the benefits of the quantization it just had just performed. The optimizer is now capable of embedding PNG images into PDFs without transcoding them.
- Fixed a minor regression with lossy JBIG2 image optimization. All JBIG2 candidates images were incorrectly placed into a single optimization group for the whole file, instead of grouping pages together. This usually makes a larger JBIG2Globals dictionary and results in inferior compression, so it worked less well than designed. However, quality would not be impacted. Lossless JBIG2 was entirely unaffected.
- Updated dependencies, including pikepdf to 1.1.0. This fixes #358.
- The install-time version checks for certain external programs have been removed from setup.py. These tests are now performed at run-time.
- The non-standard option to override install-time checks (`setup.py install --force`) is now deprecated and prints a warning. It will be removed in a future release.

2.80 v8.1.0

- Added a feature, `--unpaper-args`, which allows passing arbitrary arguments to `unpaper` when using `--clean` or `--clean-final`. The default, very conservative unpaper settings are suppressed.
- The argument `--clean-final` now implies `--clean`. It was possible to issue `--clean-final` on its before this, but it would have no useful effect.
- Fixed an exception on traversing corrupt table of contents entries (specifically, those with invalid destination objects)
- Fixed an issue when using `--tesseract-timeout` and image processing features on a file with more than 100 pages. #347
- OCRmyPDF now always calls `os.nice(5)` to signal to operating systems that it is a background process.

2.81 v8.0.1

- Fixed an exception when parsing PDFs that are missing a required field. #325
- pikepdf 1.0.5 is now required, to address some other PDF parsing issues.

2.82 v8.0.0

No major features. The intent of this release is to sever support for older versions of certain dependencies.

Breaking changes

- Dropped support for Tesseract 3.x. Tesseract 4.0 or newer is now required.
- Dropped support for Python 3.5.
- Some `ocrmypdf.pdfa` APIs that were deprecated in v7.x were removed. This functionality has been moved to pikepdf.

Other changes

- Fixed an unhandled exception when attempting to mask barcodes. #322
- It is now possible to use ocrmypdf without pdfminer.six, to support distributions that do not have it or cannot currently use it (e.g. Homebrew). Downstream maintainers should include pdfminer.six if possible.
- A warning is now issue when PDF/A conversion removes some XMP metadata from the input PDF. (Only a "whitelist" of certain XMP metadata types are allowed in PDF/A.)
- Fixed several issues that caused PDF/As to be produced with nonconforming XMP metadata (would fail validation with veraPDF).
- Fixed some instances where invalid DocumentInfo from a PDF cause XMP metadata creation to fail.
- Fixed a few documentation problems.
- pikepdf 1.0.2 is now required.

2.83 v7.4.0

- `--force-ocr` may now be used with the new `--threshold` and `--mask-barcodes` features
- pikepdf >= 0.9.1 is now required.
- Changed metadata handling to pikepdf 0.9.1. As a result, metadata handling of non-ASCII characters in Ghostscript 9.25 or later is fixed.
- chardet >= 3.0.4 is temporarily listed as required. pdfminer.six depends on it, but the most recent release does not specify this requirement. (#326)
- python-xmp-toolkit and libexempi are no longer required.
- A new Docker image is now being provided for users who wish to access OCRmyPDF over a simple HTTP interface, instead of the command line.
- Increase tolerance of PDFs that overflow or underflow the PDF graphics stack. (#325)

2.84 v7.3.1

- Fixed performance regression from v7.3.0; fast page analysis was not selected when it should be.
- Fixed a few exceptions related to the new `--mask-barcodes` feature and improved argument checking
- Added missing detection of TrueType fonts that lack a Unicode mapping

2.85 v7.3.0

- Added a new feature `--redo-ocr` to detect existing OCR in a file, remove it, and redo the OCR. This may be particularly helpful for anyone who wants to take advantage of OCR quality improvements in Tesseract 4.0. Note that OCR added by OCRmyPDF before version 3.0 cannot be detected since it was not properly marked as invisible text in the earliest versions. OCR that constructs a font from visible text, such as Adobe Acrobat's ClearScan.
- OCRmyPDF's content detection is generally more sophisticated. It learns more about the contents of each PDF and makes better recommendations:
  - OCRmyPDF can now detect when a PDF contains text that cannot be mapped to Unicode (meaning it is readable to human eyes but copy-pastes as gibberish). In these cases it recommends `--force-ocr` to make the text searchable.
  - PDFs containing vector objects are now rendered at more appropriate resolution for OCR.
  - We now exit with an error for PDFs that contain Adobe LiveCycle Designer's dynamic XFA forms. Currently the open source community does not have tools to work with these files.
  - OCRmyPDF now warns when a PDF that contains Adobe AcroForms, since such files probably do not need OCR. It can work with these files.
- Added three new **experimental** features to improve OCR quality in certain conditions. The name, syntax and behavior of these arguments is subject to change. They may also be incompatible with some other features.
  - `--remove-vectors` which strips out vector graphics. This can improve OCR quality since OCR will not search artwork for readable text; however, it currently removes "text as curves" as well.
  - `--mask-barcodes` to detect and suppress barcodes in files. We have observed that barcodes can interfere with OCR because they are "text-like" but not actually textual.

  - `--threshold` which uses a more sophisticated thresholding algorithm than is currently in use in Tesseract OCR. This works around a known issue in Tesseract 4.0 with dark text on bright backgrounds.

- Fixed an issue where an error message was not reported when the installed Ghostscript was very old.
- The PDF optimizer now saves files with object streams enabled when the optimization level is `--optimize 1` or higher (the default). This makes files a little bit smaller, but requires PDF 1.5. PDF 1.5 was first released in 2003 and is broadly supported by PDF viewers, but some rudimentary PDF parsers such as PyPDF2 do not understand object streams. You can use the command line tool `qpdf --object-streams=disable` or pikepdf library to remove them.
- New dependency: pdfminer.six 20181108. Note this is a fork of the Python 2-only pdfminer.
- Deprecation notice: At the end of 2018, we will be ending support for Python 3.5 and Tesseract 3.x. OCRmyPDF v7 will continue to work with older versions.

2.86 v7.2.1

- Fixed compatibility with an API change in pikepdf 0.3.5.
- A kludge to support Leptonica versions older than 1.72 in the test suite was dropped. Older versions of Leptonica are likely still compatible. The only impact is that a portion of the test suite will be skipped.

2.87 v7.2.0

Lossy JBIG2 behavior change

A user reported that ocrmypdf was in fact using JBIG2 in **lossy** compression mode. This was not the intended behavior. Users should review the technical concerns with JBIG2 in lossy mode and decide if this is a concern for their use case.

JBIG2 lossy mode does achieve higher compression ratios than any other monochrome compression technology; for large text documents the savings are considerable. JBIG2 lossless still gives great compression ratios and is a major improvement over the older CCITT G4 standard.

Only users who have reviewed the concerns with JBIG2 in lossy mode should opt-in. As such, lossy mode JBIG2 is only turned on when the new argument `--jbig2-lossy` is issued. This is independent of the setting for `--optimize`.

Users who did not install an optional JBIG2 encoder are unaffected.

(Thanks to user 'bsdice' for reporting this issue.)

Other issues

- When the image optimizer quantizes an image to 1 bit per pixel, it will now attempt to further optimize that image as CCITT or JBIG2, instead of keeping it in the "flate" encoding which is not efficient for 1 bpp images. (#297)
- Images in PDFs that are used as soft masks (i.e. transparency masks or alpha channels) are now excluded from optimization.
- Fixed handling of Tesseract 4.0-rc1 which now accepts invalid Tesseract configuration files, which broke the test suite.

2.88 v7.1.0

- Improve the performance of initial text extraction, which is done to determine if a file contains existing text of some kind or not. On large files, this initial processing is now about 20x times faster. (#299)
- pikepdf 0.3.3 is now required.
- Fixed #231, a problem with JPEG2000 images where image metadata was only available inside the JPEG2000 file.
- Fixed some additional Ghostscript 9.25 compatibility issues.
- Improved handling of KeyboardInterrupt error messages. (#301)
- README.md is now served in GitHub markdown instead of reStructuredText.

2.89 v7.0.6

- Blacklist Ghostscript 9.24, now that 9.25 is available and fixes many regressions in 9.24.

2.90 v7.0.5

- Improve capability with Ghostscript 9.24, and enable the JPEG passthrough feature when this version in installed.
- Ghostscript 9.24 lost the ability to set PDF title, author, subject and keyword metadata to Unicode strings. OCRmyPDF will set ASCII strings and warn when Unicode is suppressed. Other software may be used to update metadata. This is a short term work around.
- PDFs generated by Kodak Capture Desktop, or generally PDFs that contain indirect references to null objects in their table of contents, would have an invalid table of contents after processing by OCRmyPDF that might interfere with other viewers. This has been fixed.
- Detect PDFs generated by Adobe LiveCycle, which can only be displayed in Adobe Acrobat and Reader currently. When these are encountered, exit with an error instead of performing OCR on the "Please wait" error message page.

2.91 v7.0.4

- Fixed exception thrown when trying to optimize a certain type of PNG embedded in a PDF with the `-O2`
- Update to pikepdf 0.3.2, to gain support for optimizing some additional image types that were previously excluded from optimization (CMYK and grayscale). Fixes #285.

2.92 v7.0.3

- Fixed #284, an error when parsing inline images that have are also image masks, by upgrading pikepdf to 0.3.1

2.93 v7.0.2

- Fixed a regression with `--rotate-pages` on pages that already had rotations applied. (#279)
- Improve quality of page rotation in some cases by rasterizing a higher quality preview image. (#281)

2.94 v7.0.1

- Fixed compatibility with img2pdf >= 0.3.0 by rejecting input images that have an alpha channel
- Add forward compatibility for pikepdf 0.3.0 (unrelated to img2pdf)
- Various documentation updates for v7.0.0 changes

2.95 v7.0.0

- The core algorithm for combining OCR layers with existing PDF pages has been rewritten and improved considerably. PDFs are no longer split into single page PDFs for processing; instead, images are rendered and the OCR results are grafted onto the input PDF. The new algorithm uses less temporary disk space and is much more performant especially for large files.
- New dependency: pikepdf. pikepdf is a powerful new Python PDF library driving the latest OCRmyPDF features, built on the QPDF C++ library (libqpdf).
- New feature: PDF optimization with `-O` or `--optimize`. After OCR, OCRmyPDF will perform image optimizations relevant to OCR PDFs.
  - If a JBIG2 encoder is available, then monochrome images will be converted, with the potential for huge savings on large black and white images, since JBIG2 is far more efficient than any other monochrome (bi-level) compression. (All known US patents related to JBIG2 have probably expired, but it remains the responsibility of the user to supply a JBIG2 encoder such as jbig2enc. OCRmyPDF does not implement JBIG2 encoding.)
  - If `pngquant` is installed, OCRmyPDF will optionally use it to perform lossy quantization and compression of PNG images.
  - The quality of JPEGs can also be lowered, on the assumption that a lower quality image may be suitable for storage after OCR.
  - This image optimization component will eventually be offered as an independent command line utility.
  - Optimization ranges from `-O0` through `-O3`, where `0` disables optimization and `3` implements all options. `1`, the default, performs only safe and lossless optimizations. (This is similar to GCC's optimization parameter.) The exact type of optimizations performed will vary over time.
- Small amounts of text in the margins of a page, such as watermarks, page numbers, or digital stamps, will no longer prevent the rest of a page from being OCRed when `--skip-text` is issued. This behavior is based on a heuristic.
- Removed features

  - The deprecated `--pdf-renderer tesseract` PDF renderer was removed.
  - `-g`, the option to generate debug text pages, was removed because it was a maintenance burden and only worked in isolated cases. HOCR pages can still be previewed by running the hocrtransform.py with appropriate settings.
- Removed dependencies
  - `PyPDF2`
  - `defusedxml`
  - `PyMuPDF`
- The `sandwich` PDF renderer can be used with all supported versions of Tesseract, including that those prior to v3.05 which don't support `-c textonly`. (Tesseract v4.0.0 is recommended and more efficient.)
- `--pdf-renderer auto` option and the diagnostics used to select a PDF renderer now work better with old versions, but may make different decisions than past versions.
- If everything succeeds but PDF/A conversion fails, a distinct return code is now returned (`ExitCode.pdfa_conversion_failed (10)`) where this situation previously returned `ExitCode.invalid_output_pdf (4)`. The latter is now returned only if there is some indication that the output file is invalid.
- Notes for downstream packagers
  - There is also a new dependency on `python-xmp-toolkit` which in turn depends on `libexempi3`.
  - It may be necessary to separately `pip install pycparser` to avoid another Python 3.7 issue.

2.96 v6.2.5

- Disable a failing test due to Tesseract 4.0rc1 behavior change. Previously, Tesseract would exit with an error message if its configuration was invalid, and OCRmyPDF would intercept this message. Now Tesseract issues a warning, which OCRmyPDF v6.2.5 may relay or ignore. (In v7.x, OCRmyPDF will respond to the warning.)
- This release branch no longer supports using the optional PyMuPDF installation, since it was removed in v7.x.
- This release branch no longer supports macOS. macOS users should upgrade to v7.x.

2.97 v6.2.4

- Backport Ghostscript 9.25 compatibility fixes, which removes support for setting Unicode metadata
- Backport blacklisting Ghostscript 9.24
- Older versions of Ghostscript are still supported

2.98 v6.2.3

- Fixed compatibility with img2pdf >= 0.3.0 by rejecting input images that have an alpha channel
- This version will be included in Ubuntu 18.10

2.99 v6.2.2

- Backport compatibility fixes for Python 3.7 and ruffus 2.7.0 from v7.0.0
- Backport fix to ignore masks when deciding what colors are on a page
- Backport some minor improvements from v7.0.0: better argument validation and warnings about the Tesseract 4.0.0 `--user-words` regression

2.100 v6.2.1

- Fixed recent versions of Tesseract (after 4.0.0-beta1) not being detected as supporting the `sandwich` renderer (#271).

2.101 v6.2.0

- **Docker**: The Docker image `ocrmypdf-tess4` has been removed. The main Docker images, `ocrmypdf` and `ocrmypdf-polyglot` now use Ubuntu 18.04 as a base image, and as such Tesseract 4.0.0-beta1 is now the Tesseract version they use. There is no Docker image based on Tesseract 3.05 anymore.
- Creation of PDF/A-3 is now supported. However, there is no ability to attach files to PDF/A-3.
- Lists more reasons why the file size might grow.
- Fixed #262, `--remove-background` error on PDFs contained colormapped (paletted) images.
- Fixed another XMP metadata validation issue, in cases where the input file's creation date has no timezone and the creation date is not overridden.

2.102 v6.1.5

- Fixed #253, a possible division by zero when using the `hocr` renderer.
- Fixed incorrectly formatted `<xmp:ModifyDate>` field inside XMP metadata for PDF/As. veraPDF flags this as a PDF/A validation failure. The error is caused the timezone and final digit of the seconds of modified time to be omitted, so at worst the modification time stamp is rounded to the nearest 10 seconds.

2.103 v6.1.4

- Fixed #248 `--clean` argument may remove OCR from left column of text on certain documents. We now set `--layout none` to suppress this.
- The test cache was updated to reflect the change above.
- Change test suite to accommodate Ghostscript 9.23's new ability to insert JPEGs into PDFs without transcoding.
- XMP metadata in PDFs is now examined using `defusedxml` for safety.
- If an external process exits with a signal when asked to report its version, we now print the system error message instead of suppressing it. This occurred when the required executable was found but was missing a shared library.
- qpdf 7.0.0 or newer is now required as the test suite can no longer pass without it.

2.103.1 Notes

- An apparent regression in Ghostscript 9.23 will cause some ocrmypdf output files to become invalid in rare cases; the workaround for the moment is to set `--force-ocr`.

2.104 v6.1.3

- Fixed #247, `/CreationDate` metadata not copied from input to output.
- A warning is now issued when Python 3.5 is used on files with a large page count, as this case is known to regress to single core performance. The cause of this problem is unknown.

2.105 v6.1.2

- Upgrade to PyMuPDF v1.12.5 which includes a more complete fix to #239.
- Add `defusedxml` dependency.

2.106 v6.1.1

- Fixed text being reported as found on all pages if PyMuPDF is not installed.

2.107 v6.1.0

- PyMuPDF is now an optional but recommended dependency, to alleviate installation difficulties on platforms that have less access to PyMuPDF than the author anticipated. (For version 6.x only) install OCRmyPDF with `pip install ocrmypdf[fitz]` to use it to its full potential.
- Fixed `FileExistsError` that could occur if OCR timed out while it was generating the output file. (#218)
- Fixed table of contents/bookmarks all being redirected to page 1 when generating a PDF/A (with PyMuPDF). (Without PyMuPDF the table of contents is removed in PDF/A mode.)
- Fixed "RuntimeError: invalid key in dict" when table of contents/bookmarks titles contained the character `)`. (#239)

- Added a new argument `--skip-repair` to skip the initial PDF repair step if the PDF is already well-formed (because another program repaired it).

2.108 v6.0.0

- The software license has been changed to GPLv3 [it has since changed again]. Test resource files and some individual sources may have other licenses.
- OCRmyPDF now depends on PyMuPDF. Including PyMuPDF is the primary reason for the change to GPLv3.
- Other backward incompatible changes
  - The `OCRMYPDF_TESSERACT`, `OCRMYPDF_QPDF`, `OCRMYPDF_GS` and `OCRMYPDF_UNPAPER` environment variables are no longer used. Change `PATH` if you need to override the external programs OCRmyPDF uses.
  - The `ocrmypdf` package has been moved to `src/ocrmypdf` to avoid issues with accidental import.
  - The function `ocrmypdf.exec.get_program` was removed.
  - The deprecated module `ocrmypdf.pageinfo` was removed.
  - The `--pdf-renderer tess4` alias for `sandwich` was removed.
- Fixed an issue where OCRmyPDF failed to detect existing text on pages, depending on how the text and fonts were encoded within the PDF. (#233#232)
- Fixed an issue that caused dramatic inflation of file sizes when `--skip-text --output-type pdf` was used. OCRmyPDF now removes duplicate resources such as fonts, images and other objects that it generates. (#237)
- Improved performance of the initial page splitting step. Originally this step was not believed to be expensive and ran in a process. Large file testing revealed it to be a bottleneck, so it is now parallelized. On a 700 page file with quad core machine, this change saves about 2 minutes. (#234)
- The test suite now includes a cache that can be used to speed up test runs across platforms. This also does not require computing checksums, so it's faster. (#217)

2.109 v5.7.0

- Fixed an issue that caused poor CPU utilization on machines with more than 4 cores when running Tesseract 4. (Related to #217.)
- The 'hocr' renderer has been improved. The 'sandwich' and 'tesseract' renderers are still better for most use cases, but 'hocr' may be useful for people who work with the PDF.js renderer in English/ASCII languages. (#225)
  - It now formats text in a matter that is easier for certain PDF viewers to select and extract copy and paste text. This should help macOS Preview and PDF.js in particular.
  - The appearance of selected text and behavior of selecting text is improved.
  - The PDF content stream now uses relative moves, making it more compact and easier for viewers to determine when two words on the same line.
  - It can now deal with text on a skewed baseline.
  - Thanks to @cforcey for the pull request, @jbreiden for many helpful suggestions, @ctbarbour for another round of improvements, and @acaloiaro for an independent review.

2.110 v5.6.3

- Suppress two debug messages that were too verbose

2.111 v5.6.2

- Development branch accidentally tagged as release. Do not use.

2.112 v5.6.1

- Fixed #219: change how the final output file is created to avoid triggering permission errors when the output is a special file such as `/dev/null`
- Fixed test suite failures due to a qpdf 8.0.0 regression and Python 3.5's handling of symlink
- The "encrypted PDF" error message was different depending on the type of PDF encryption. Now a single clear message appears for all types of PDF encryption.
- ocrmypdf is now in Homebrew. Homebrew users are advised to the version of ocrmypdf in the official homebrew-core formulas rather than the private tap.
- Some linting

2.113 v5.6.0

- Fixed #216: preserve "text as curves" PDFs without rasterizing file
- Related to the above, messages about rasterizing are more consistent
- For consistency versions minor releases will now get the trailing .0 they always should have had.

2.114 v5.5

- Add new argument `--max-image-mpixels`. Pillow 5.0 now raises an exception when images may be decompression bombs. This argument can be used to override the limit Pillow sets.
- Fixed output page cropped when using the sandwich renderer and OCR is skipped on a rotated and image-processed page
- A warning is now issued when old versions of Ghostscript are used in cases known to cause issues with non-Latin characters
- Fixed a few parameter validation checks for `-output-type pdfa-1` and `pdfa-2`

2.115 v5.4.4

- Fixed #181: fix final merge failure for PDFs with more pages than the system file handle limit (`ulimit -n`)
- Fixed #200: an uncommon syntax for formatting decimal numbers in a PDF would cause qpdf to issue a warning, which ocrmypdf treated as an error. Now this the warning is relayed.
- Fixed an issue where intermediate PDFs would be created at version 1.3 instead of the version of the original file. It's possible but unlikely this had side effects.
- A warning is now issued when older versions of qpdf are used since issues like #200 cause qpdf to infinite-loop
- Address issue #140: if Tesseract outputs invalid UTF-8, escape it and print its message instead of aborting with a Unicode error
- Adding previously unlisted setup requirement, pytest-runner
- Update documentation: fix an error in the example script for Synology with Docker images, improved security guidance, advised `pip install --user`

2.116 v5.4.3

- If a subprocess fails to report its version when queried, exit cleanly with an error instead of throwing an exception
- Added test to confirm that the system locale is Unicode-aware and fail early if it's not
- Clarified some copyright information
- Updated pinned requirements.txt so the homebrew formula captures more recent versions

2.117 v5.4.2

- Fixed a regression from v5.4.1 that caused sidecar files to be created as empty files

2.118 v5.4.1

- Add workaround for Tesseract v4.00alpha crash when trying to obtain orientation and the latest language packs are installed

2.119 v5.4

- Change wording of a deprecation warning to improve clarity
- Added option to generate PDF/A-1b output if desired (`--output-type pdfa-1`); default remains PDF/A-2b generation
- Update documentation

2.120 v5.3.3

- Fixed missing error message that should occur when trying to force `--pdf-renderer sandwich` on old versions of Tesseract
- Update copyright information in test files
- Set system `LANG` to UTF-8 in Dockerfiles to avoid UTF-8 encoding errors

2.121 v5.3.2

- Fixed a broken test case related to language packs

2.122 v5.3.1

- Fixed wrong return code given for missing Tesseract language packs
- Fixed "brew audit" crashing on Travis when trying to auto-brew

2.123 v5.3

- Added `--user-words` and `--user-patterns` arguments which are forwarded to Tesseract OCR as words and regular expressions respective to use to guide OCR. Supplying a list of subject-domain words should assist Tesseract with resolving words. (#165)
- Using a non Latin-1 language with the "hocr" renderer now warns about possible OCR quality and recommends workarounds (#176)
- Output file path added to error message when that location is not writable (#175)
- Otherwise valid PDFs with leading whitespace at the beginning of the file are now accepted

2.124 v5.2

- When using Tesseract 3.05.01 or newer, OCRmyPDF will select the "sandwich" PDF renderer by default, unless another PDF renderer is specified with the `--pdf-renderer` argument. The previous behavior was to select `--pdf-renderer=hocr`.
- The "tesseract" PDF renderer is now deprecated, since it can cause problems with Ghostscript on Tesseract 3.05.00
- The "tess4" PDF renderer has been renamed to "sandwich". "tess4" is now a deprecated alias for "sandwich".

2.125 v5.1

- Files with pages larger than 200" (5080 mm) in either dimension are now supported with `--output-type=pdf` with the page size preserved (in the PDF specification this feature is called UserUnit scaling). Due to Ghostscript limitations this is not available in conjunction with PDF/A output.

2.126 v5.0.1

- Fixed #169, exception due to failure to create sidecar text files on some versions of Tesseract 3.04, including the jbarlow83/ocrmypdf Docker image

2.127 v5.0

- Backward incompatible changes
  - Support for Python 3.4 dropped. Python 3.5 is now required.
  - Support for Tesseract 3.02 and 3.03 dropped. Tesseract 3.04 or newer is required. Tesseract 4.00 (alpha) is supported.
  - The OCRmyPDF.sh script was removed.
- Add a new feature, `--sidecar`, which allows creating "sidecar" text files which contain the OCR results in plain text. These OCR text is more reliable than extracting text from PDFs. Closes #126.
- New feature: `--pdfa-image-compression`, which allows overriding Ghostscript's lossy-or-lossless image encoding heuristic and making all images JPEG encoded or lossless encoded as desired. Fixes #163.
- Fixed #143, added `--quiet` to suppress "INFO" messages
- Fixed #164, a typo
- Removed the command line parameters `-n` and `--just-print` since they have not worked for some time (reported as Ubuntu bug #1687308)

2.128 v4.5.6

- Fixed #156, 'NoneType' object has no attribute 'getObject' on pages with no optional /Contents record. This should resolve all issues related to pages with no /Contents record.
- Fixed #158, ocrmypdf now stops and terminates if Ghostscript fails on an intermediate step, as it is not possible to proceed.
- Fixed #160, exception thrown on certain invalid arguments instead of error message

2.129 v4.5.5

- Automated update of macOS homebrew tap
- Fixed #154, KeyError '/Contents' when searching for text on blank pages that have no /Contents record. Note: incomplete fix for this issue.

2.130 v4.5.4

- Fixed `--skip-big` raising an exception if a page contains no images (#152) (thanks to @TomRaz)
- Fixed an issue where pages with no images might trigger "cannot write mode P as JPEG" (#151)

2.131 v4.5.3

- Added a workaround for Ghostscript 9.21 and probably earlier versions would fail with the error message "VMerror -25", due to a Ghostscript bug in XMP metadata handling
- High Unicode characters (U+10000 and up) are no longer accepted for setting metadata on the command line, as Ghostscript may not handle them correctly.
- Fixed an issue where the `tess4` renderer would duplicate content onto output pages if tesseract failed or timed out
- Fixed `tess4` renderer not recognized when lossless reconstruction is possible

2.132 v4.5.2

- Fixed #147, `--pdf-renderer tess4 --clean` will produce an oversized page containing the original image in the bottom left corner, due to loss DPI information.
- Make "using Tesseract 4.0" warning less ominous
- Set up machinery for homebrew OCRmyPDF tap

2.133 v4.5.1

- Fixed #137, proportions of images with a non-square pixel aspect ratio would be distorted in output for `--force-ocr` and some other combinations of flags

2.134 v4.5

- PDFs containing "Form XObjects" are now supported (issue #134; PDF reference manual 8.10), and images they contain are taken into account when determining the resolution for rasterizing
- The Tesseract 4 Docker image no longer includes all languages, because it took so long to build something would tend to fail
- OCRmyPDF now warns about using `--pdf-renderer tesseract` with Tesseract 3.04 or lower due to issues with Ghostscript corrupting the OCR text in these cases

2.135 v4.4.2

- The Docker images (ocrmypdf, ocrmypdf-polyglot, ocrmypdf-tess4) are now based on Ubuntu 16.10 instead of Debian stretch
  - This makes supporting the Tesseract 4 image easier
  - This could be a disruptive change for any Docker users who built customized these images with their own changes, and made those changes in a way that depends on Debian and not Ubuntu
- OCRmyPDF now prevents running the Tesseract 4 renderer with Tesseract 3.04, which was permitted in v4.4 and v4.4.1 but will not work

2.136 v4.4.1

- To prevent a TIFF output error caused by img2pdf >= 0.2.1 and Pillow <= 3.4.2, dependencies have been tightened
- The Tesseract 4.00 simultaneous process limit was increased from 1 to 2, since it was observed that 1 lowers performance
- Documentation improvements to describe the `--tesseract-config` feature
- Added test cases and fixed error handling for `--tesseract-config`
- Tweaks to setup.py to deal with issues in the v4.4 release

2.137 v4.4

- Tesseract 4.00 is now supported on an experimental basis.
  - A new rendering option `--pdf-renderer tess4` exploits Tesseract 4's new text-only output PDF mode. See the documentation on PDF Renderers for details.
  - The `--tesseract-oem` argument allows control over the Tesseract 4 OCR engine mode (tesseract's `--oem`). Use `--tesseract-oem 2` to enforce the new LSTM mode.
  - Fixed poor performance with Tesseract 4.00 on Linux
- Fixed an issue that caused corruption of output to stdout in some cases
- Removed test for Pillow JPEG and PNG support, as the minimum supported version of Pillow now enforces this
- OCRmyPDF now tests that the intended destination file is writable before proceeding
- The test suite now requires `pytest-helpers-namespace` to run (but not install)

- Significant code reorganization to make OCRmyPDF re-entrant and improve performance. All changes should be backward compatible for the v4.x series.
  - However, OCRmyPDF's dependency "ruffus" is not re-entrant, so no Python API is available. Scripts should continue to use the command line interface.

2.138 v4.3.5

- Update documentation to confirm Python 3.6.0 compatibility. No code changes were needed, so many earlier versions are likely supported.

2.139 v4.3.4

- Fixed "decimal.InvalidOperation: quantize result has too many digits" for high DPI images

2.140 v4.3.3

- Fixed PDF/A creation with Ghostscript 9.20 properly
- Fixed an exception on inline stencil masks with a missing optional parameter

2.141 v4.3.2

- Fixed a PDF/A creation issue with Ghostscript 9.20 (note: this fix did not actually work)

2.142 v4.3.1

- Fixed an issue where pages produced by the "hocr" renderer after a Tesseract timeout would be rotated incorrectly if the input page was rotated with a /Rotate marker
- Fixed a file handle leak in LeptonicaErrorTrap that would cause a "too many open files" error for files around hundred pages of pages long when `--deskew` or `--remove-background` or other Leptonica based image processing features were in use, depending on the system value of `ulimit -n`
- Ability to specify multiple languages for multilingual documents is now advertised in documentation
- Reduced the file sizes of some test resources
- Cleaned up debug output
- Tesseract caching in test cases is now more cautious about false cache hits and reproducing exact output, not that any problems were observed

2.143 v4.3

- New feature `--remove-background` to detect and erase the background of color and grayscale images
- Better documentation
- Fixed an issue with PDFs that draw images when the raster stack depth is zero
- ocrmypdf can now redirect its output to stdout for use in a shell pipeline
  - This does not improve performance since temporary files are still used for buffering
  - Some output validation is disabled in this mode

2.144 v4.2.5

- Fixed an issue (#100) with PDFs that omit the optional /BitsPerComponent parameter on images
- Removed non-free file milk.pdf

2.145 v4.2.4

- Fixed an error (#90) caused by PDFs that use stencil masks properly
- Fixed handling of PDFs that try to draw images or stencil masks without properly setting up the graphics state (such images are now ignored for the purposes of calculating DPI)

2.146 v4.2.3

- Fixed an issue with PDFs that store page rotation (/Rotate) in an indirect object
- Integrated a few fixes to simplify downstream packaging (Debian)
  - The test suite no longer assumes it is installed
  - If running Linux, skip a test that passes Unicode on the command line
- Added a test case to check explicit masks and stencil masks
- Added a test case for indirect objects and linearized PDFs
- Deprecated the OCRmyPDF.sh shell script

2.147 v4.2.2

- Improvements to documentation

2.148 v4.2.1

- Fixed an issue where PDF pages that contained stencil masks would report an incorrect DPI and cause Ghostscript to abort
- Implemented stdin streaming

2.149 v4.2

- ocrmypdf will now try to convert single image files to PDFs if they are provided as input (#15)
  - This is a basic convenience feature. It only supports a single image and always makes the image fill the whole page.
  - For better control over image to PDF conversion, use `img2pdf` (one of ocrmypdf's dependencies)
- New argument `--output-type {pdf|pdfa}` allows disabling Ghostscript PDF/A generation
  - `pdfa` is the default, consistent with past behavior
  - `pdf` provides a workaround for users concerned about the increase in file size from Ghostscript forcing JBIG2 images to CCITT and transcoding JPEGs
  - `pdf` preserves as much as it can about the original file, including problems that PDF/A conversion fixes
- PDFs containing images with "non-square" pixel aspect ratios, such as 200x100 DPI, are now handled and converted properly (fixing a bug that caused to be cropped)
- `--force-ocr` rasterizes pages even if they contain no images
  - supports users who want to use OCRmyPDF to reconstruct text information in PDFs with damaged Unicode maps (copy and paste text does not match displayed text)
  - supports reinterpreting PDFs where text was rendered as curves for printing, and text needs to be recovered
  - fixes issue #82
- Fixes an issue where, with certain settings, monochrome images in PDFs would be converted to 8-bit grayscale, increasing file size (#79)
- Support for Ubuntu 12.04 LTS "precise" has been dropped in favor of (roughly) Ubuntu 14.04 LTS "trusty"
  - Some Ubuntu "PPAs" (backports) are needed to make it work
- Support for some older dependencies dropped
  - Ghostscript 9.15 or later is now required (available in Ubuntu trusty with backports)
  - Tesseract 3.03 or later is now required (available in Ubuntu trusty)
- Ghostscript now runs in "safer" mode where possible

2.150 v4.1.4

- Bug fix: monochrome images with an ICC profile attached were incorrectly converted to full color images if lossless reconstruction was not possible due to other settings; consequence was increased file size for these images

2.151 v4.1.3

- More helpful error message for PDFs with version 4 security handler
- Update usage instructions for Windows/Docker users
- Fixed order of operations for matrix multiplication (no effect on most users)
- Add a few leptonica wrapper functions (no effect on most users)

2.152 v4.1.2

- Replace IEC sRGB ICC profile with Debian's sRGB (from icc-profiles-free) which is more compatible with the MIT license
- More helpful error message for an error related to certain types of malformed PDFs

2.153 v4.1

- `--rotate-pages` now only rotates pages when reasonably confidence in the orientation. This behavior can be adjusted with the new argument `--rotate-pages-threshold`
- Fixed problems in error checking if `unpaper` is uninstalled or missing at run-time
- Fixed problems with "RethrownJobError" errors during error handling that suppressed the useful error messages

2.154 v4.0.7

- Minor correction to Ghostscript output settings

2.155 v4.0.6

- Update install instructions
- Provide a sRGB profile instead of using Ghostscript's

2.156 v4.0.5

- Remove some verbose debug messages from v4.0.4
- Fixed temporary that wasn't being deleted
- DPI is now calculated correctly for cropped images, along with other image transformations
- Inline images are now checked during DPI calculation instead of rejecting the image

2.157 v4.0.4

Released with verbose debug message turned on. Do not use. Skip to v4.0.5.

2.158 v4.0.3

New features

- Page orientations detected are now reported in a summary comment

Fixes

- Show stack trace if unexpected errors occur
- Treat "too few characters" error message from Tesseract as a reason to skip that page rather than abort the file
- Docker: fix blank JPEG2000 issue by insisting on Ghostscript versions that have this fixed

2.159 v4.0.2

Fixes

- Fixed compatibility with Tesseract 3.04.01 release, particularly its different way of outputting orientation information
- Improved handling of Tesseract errors and crashes
- Fixed use of chmod on Docker that broke most test cases

2.160 v4.0.1

Fixes

- Fixed a KeyError if tesseract fails to find page orientation information

2.161 v4.0

New features

- Automatic page rotation (`-r`) is now available. It uses ignores any prior rotation information on PDFs and sets rotation based on the dominant orientation of detectable text. This feature is fairly reliable but some false positives occur especially if there is not much text to work with. (#4)
- Deskewing is now performed using Leptonica instead of unpaper. Leptonica is faster and more reliable at image deskewing than unpaper.

Fixes

- Fixed an issue where lossless reconstruction could cause some pages to be appear incorrectly if the page was rotated by the user in Acrobat after being scanned (specifically if it a /Rotate tag)
- Fixed an issue where lossless reconstruction could misalign the graphics layer with respect to text layer if the page had been cropped such that its origin is not (0, 0) (#49)

Changes

- Logging output is now much easier to read
- `--deskew` is now performed by Leptonica instead of unpaper (#25)
- libffi is now required
- Some changes were made to the Docker and Travis build environments to support libffi
- `--pdf-renderer=tesseract` now displays a warning if the Tesseract version is less than 3.04.01, the planned release that will include fixes to an important OCR text rendering bug in Tesseract 3.04.00. You can also manually install ./share/sharp2.ttf on top of pdf.ttf in your Tesseract tessdata folder to correct the problem.

2.162 v3.2.1

Changes

- Fixed #47 "convert() got and unexpected keyword argument 'dpi'" by upgrading to img2pdf 0.2
- Tweaked the Dockerfiles

2.163 v3.2

New features

- Lossless reconstruction: when possible, OCRmyPDF will inject text layers without otherwise manipulating the content and layout of a PDF page. For example, a PDF containing a mix of vector and raster content would see the vector content preserved. Images may still be transcoded during PDF/A conversion. (`--deskew` and `--clean-final` disable this mode, necessarily.)
- New argument `--tesseract-pagesegmode` allows you to pass page segmentation arguments to Tesseract OCR. This helps for two column text and other situations that confuse Tesseract.
- Added a new "polyglot" version of the Docker image, that generates Tesseract with all languages packs installed, for the polyglots among us. It is much larger.

Changes

- JPEG transcoding quality is now 95 instead of the default 75. Bigger file sizes for less degradation.

2.164 v3.1.1

Changes

- Fixed bug that caused incorrect page size and DPI calculations on documents with mixed page sizes

2.165 v3.1

Changes

- Default output format is now PDF/A-2b instead of PDF/A-1b
- Python 3.5 and macOS El Capitan are now supported platforms - no changes were needed to implement support
- Improved some error messages related to missing input files
- Fixed #20: uppercase .PDF extension not accepted
- Fixed an issue where OCRmyPDF failed to text that certain pages contained previously OCR'ed text, such as OCR text produced by Tesseract 3.04
- Inserts /Creator tag into PDFs so that errors can be traced back to this project
- Added new option `--pdf-renderer=auto`, to let OCRmyPDF pick the best PDF renderer. Currently it always chooses the 'hocrtransform' renderer but that behavior may change.
- Set up Travis CI automatic integration testing

2.166 v3.0

New features

- Easier installation with a Docker container or Python's `pip` package manager
- Eliminated many external dependencies, so it's easier to setup
- Now installs `ocrmypdf` to `/usr/local/bin` or equivalent for system-wide access and easier typing
- Improved command line syntax and usage help (`--help`)
- Tesseract 3.03+ PDF page rendering can be used instead for better positioning of recognized text (`--pdf-renderer tesseract`)
- PDF metadata (title, author, keywords) are now transferred to the output PDF
- PDF metadata can also be set from the command line (`--title`, etc.)
- Automatic repairs malformed input PDFs if possible
- Added test cases to confirm everything is working
- Added option to skip extremely large pages that take too long to OCR and are often not OCRable (e.g. large scanned maps or diagrams); other pages are still processed (`--skip-big`)
- Added option to kill Tesseract OCR process if it seems to be taking too long on a page, while still processing other pages (`--tesseract-timeout`)
- Less common colorspaces (CMYK, palette) are now supported by conversion to RGB
- Multiple images on the same PDF page are now supported

Changes

- New, robust rewrite in Python 3.4+ with ruffus pipelines
- Now uses Ghostscript 9.14's improved color conversion model to preserve PDF colors
- OCR text is now rendered in the PDF as invisible text. Previous versions of OCRmyPDF incorrectly rendered visible text with an image on top.
- All "tasks" in the pipeline can be executed in parallel on any available CPUs, increasing performance
- The `-o DPI` argument has been phased out, in favor of `--oversample DPI`, in case we need `-o OUTPUTFILE` in the future
- Removed several dependencies, so it's easier to install. We no longer use:
  - GNU parallel
  - ImageMagick
  - Python 2.7
  - Poppler
  - MuPDF tools
  - shell scripts
  - Java and JHOVE
  - libxml2
- Some new external dependencies are required or optional, compared to v2.x:
  - Ghostscript 9.14+
  - qpdf 5.0.0+
  - Unpaper 6.1 (optional)
  - some automatically managed Python packages

Release candidates^

- rc9:
  - Fix #118: report error if ghostscript iccprofiles are missing
  - fixed another issue related to #111: PDF rasterized to palette file
  - add support image files with a palette
  - don't try to validate PDF file after an exception occurs
- rc8:
  - Fix #111: exception thrown if PDF is missing DocumentInfo dictionary
- rc7:
  - fix error when installing direct from pip, "no such file 'requirements.txt'"
- rc6:
  - dropped libxml2 (Python lxml) since Python 3's internal XML parser is sufficient
  - set up Docker container
  - fix Unicode errors if recognized text contains Unicode characters and system locale is not UTF-8
- rc5:

  - dropped Java and JHOVE in favour of qpdf
  - improved command line error output
  - additional tests and bug fixes
  - tested on Ubuntu 14.04 LTS
- rc4:
  - dropped MuPDF in favour of qpdf
  - fixed some installer issues and errors in installation instructions
  - improve performance: run Ghostscript with multithreaded rendering
  - improve performance: use multiple cores by default
  - bug fix: checking for wrong exception on process timeout
- rc3: skipping version number intentionally to avoid confusion with Tesseract
- rc2: first release for public testing to test-PyPI, Github
- rc1: testing release process

2.167 Compatibility notes

- `./OCRmyPDF.sh` script is still available for now
- Stacking the verbosity option like `-vvv` is no longer supported
- The configuration file `config.sh` has been removed. Instead, you can feed a file to the arguments for common settings:

```
ocrmypdf input.pdf output.pdf @settings.txt
```

where `settings.txt` contains *one argument per line*, for example:

```
-l
deu
--author
A. Merkel
--pdf-renderer
tesseract
```

Fixes

- Handling of filenames containing spaces: fixed

Notes and known issues

- Some dependencies may work with lower versions than tested, so try overriding dependencies if they are "in the way" to see if they work.
- `--pdf-renderer tesseract` will output files with an incorrect page size in Tesseract 3.03, due to a bug in Tesseract.
- PDF files containing "inline images" are not supported and won't be for the 3.0 release. Scanned images almost never contain inline images.

2.168 v2.2-stable (2014-09-29)

OCRmyPDF versions 1 and 2 were implemented as shell scripts. OCRmyPDF 3.0+ is a fork that gradually replaced all shell scripts with Python while maintaining the existing command line arguments. No one is maintaining old versions.

For details on older versions, see the final version of its release notes.

CHAPTER

THREE

INSTALLING OCRMYPDF

The easiest way to install OCRmyPDF is to follow the steps for your operating system/platform. This version may be out of date, however.

These platforms have one-liner installs:

Debian, Ubuntu	`apt install ocrmypdf`
Windows Subsystem for Linux	`apt install ocrmypdf`
Fedora	`dnf install ocrmypdf`
macOS	`brew install ocrmypdf`
LinuxBrew	`brew install ocrmypdf`
FreeBSD	`pkg install py38-ocrmypdf`
Conda (WSL, macOS, Linux)	`conda install ocrmypdf`

More detailed procedures are outlined below. If you want to do a manual install, or install a more recent version than your platform provides, read on.

Platform-specific steps

- *Installing on Linux*
  - *Debian and Ubuntu 18.04 or newer*
  - *Fedora*
  - *Installing the latest version on Ubuntu 20.04 LTS*
  - *Ubuntu 18.04 LTS*
  - *Ubuntu 16.04 LTS*
  - *Arch Linux (AUR)*
  - *Alpine Linux*
  - *Mageia 7*
  - *Other Linux packages*
- *Installing on macOS*
  - *Homebrew*
  - *Manual installation on macOS*
- *Installing on Windows*

  - *Native Windows*
  - *Windows Subsystem for Linux*
  - *Cygwin64*
  - *Docker*
- *Installing on FreeBSD*
- *Installing the Docker image*
- *Installing with Python pip*
  - *Requirements for pip and HEAD install*
- *Installing HEAD revision from sources*
  - *For development*
- *Shell completions*

3.1 Installing on Linux

3.1.1 Debian and Ubuntu 18.04 or newer

OCRmyPDF versions in Debian & Ubuntu

Users of Debian 9 ("stretch") or later, or Ubuntu 18.04 or later, including users of Windows Subsystem for Linux, may simply

```
apt-get install ocrmypdf
```

As indicated in the table above, Debian and Ubuntu releases may lag behind the latest version. If the version available for your platform is out of date, you could opt to install the latest version from source. See *Installing HEAD revision from sources*. Ubuntu 16.10 to 17.10 inclusive also had ocrmypdf, but these versions are end of life.

For full details on version availability for your platform, check the Debian Package Tracker or Ubuntu launchpad.net.

Note: OCRmyPDF for Debian and Ubuntu currently omit the JBIG2 encoder. OCRmyPDF works fine without it but will produce larger output files. If you build jbig2enc from source, ocrmypdf 7.0.0 and later will automatically detect it (specifically the `jbig2` binary) on the `PATH`. To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.2 Fedora

OCRmyPDF version

Users of Fedora 29 or later may simply

```
dnf install ocrmypdf
```

For full details on version availability, check the Fedora Package Tracker.

If the version available for your platform is out of date, you could opt to install the latest version from source. See *Installing HEAD revision from sources*.

Note: OCRmyPDF for Fedora currently omits the JBIG2 encoder due to patent issues. OCRmyPDF works fine without it but will produce larger output files. If you build jbig2enc from source, ocrmypdf 7.0.0 and later will automatically detect it on the `PATH`. To add JBIG2 encoding, see Installing the JBIG2 encoder.

3.1.3 Installing the latest version on Ubuntu 20.04 LTS

Ubuntu 20.04 includes ocrmypdf 9.6.0 - you can install that with `apt`. To install a more recent version, uninstall the system-provided version of ocrmypdf, and install the following dependencies:

```
sudo apt-get -y remove ocrmypdf  # remove system ocrmypdf, if installed
sudo apt-get -y update
sudo apt-get -y install \
    ghostscript \
    icc-profiles-free \
    liblept5 \
    libxml2 \
    pngquant \
    python3-pip \
    tesseract-ocr \
    zlib1g
```

To install ocrmypdf for the system:

```
pip3 install ocrmypdf
```

To install for the current user only:

```
export PATH=$HOME/.local/bin:$PATH
pip3 install --user ocrmypdf
```

3.1.4 Ubuntu 18.04 LTS

Ubuntu 18.04 includes ocrmypdf 6.1.2 - you can install that with `apt`, but it is quite old now. To install a more recent version, uninstall the old version of ocrmypdf, and install the following dependencies:

```
sudo apt-get -y remove ocrmypdf
sudo apt-get -y update
sudo apt-get -y install \
    ghostscript \
    icc-profiles-free \
    liblept5 \
    libxml2 \
    pngquant \
    python3-cffi \
    python3-distutils \
    python3-pkg-resources \
    python3-reportlab \
    qpdf \
    tesseract-ocr \
    zlib1g \
    unpaper
```

We will need a newer version of `pip` then was available for Ubuntu 18.04:

```
wget https://bootstrap.pypa.io/get-pip.py && python3 get-pip.py
```

Then install the most recent ocrmypdf for the local user and set the user's `PATH` to check for the user's Python packages.

```
export PATH=$HOME/.local/bin:$PATH
python3 -m pip install --user ocrmypdf
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.5 Ubuntu 16.04 LTS

No package is available for Ubuntu 16.04. OCRmyPDF 8.0 and newer require Python 3.6. Ubuntu 16.04 ships Python 3.5, but you can install Python 3.6 on it. Or, you can skip Python 3.6 and install OCRmyPDF 7.x or older - for that procedure, please see the installation documentation for the version of OCRmyPDF you plan to use.

Install system packages for OCRmyPDF

```
sudo apt-get update
sudo apt-get install -y software-properties-common python-software-properties
sudo add-apt-repository -y \
    ppa:jonathonf/python-3.6 \
    ppa:alex-p/tesseract-ocr
sudo apt-get update
sudo apt-get install -y \
    ghostscript \
    libexempi3 \
    libffi6 \
    pngquant \
    python3.6 \
    qpdf \
```

(continues on next page)

(continued from previous page)

```
    tesseract-ocr \
    unpaper
```

This will install a Python 3.6 binary at `/usr/bin/python3.6` alongside the system's Python 3.5. Do not remove the system Python. This will also install Tesseract 4.0 from a PPA, since the version available in Ubuntu 16.04 is too old for OCRmyPDF.

Now install pip for Python 3.6. This will install the Python 3.6 version of `pip` at `/usr/local/bin/pip`.

```
curl https://bootstrap.pypa.io/get-pip.py | sudo python3.6
```

Install OCRmyPDF

OCRmyPDF requires the locale to be set for UTF-8. **On some minimal Ubuntu installations**, such as the Ubuntu 16.04 Docker images it may be necessary to set the locale.

```
# Optional: Only need to set these if they are not already set
export LC_ALL=C.UTF-8
export LANG=C.UTF-8
```

Now install OCRmyPDF for the current user, and ensure that the `PATH` environment variable contains `$HOME/.local/bin`.

```
export PATH=$HOME/.local/bin:$PATH
pip3.6 install --user ocrmypdf
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.1.6 Arch Linux (AUR)

There is an Arch User Repository (AUR) package for OCRmyPDF.

Installing AUR packages as root is not allowed, so you must first setup a non-root user and configure sudo. The standard Docker image, `archlinux/base:latest`, does **not** have a non-root user configured, so users of that image must follow these guides. If you are using a VM image, such as the official Vagrant image, this work may already be completed for you.

Next you should install the base-devel package group. This includes the standard tooling needed to build packages, such as a compiler and binary tools.

```
sudo pacman -S base-devel
```

Now you are ready to install the OCRmyPDF package.

```
curl -O https://aur.archlinux.org/cgit/aur.git/snapshot/ocrmypdf.tar.gz
tar xvzf ocrmypdf.tar.gz
cd ocrmypdf
makepkg -sri
```

At this point you will have a working install of OCRmyPDF, but the Tesseract install won't include any OCR language data. You can install the tesseract-data package group to add all supported languages, or use that package listing to identify the appropriate package for your desired language.

```
sudo pacman -S tesseract-data-eng
```

As an alternative to this manual procedure, consider using an AUR helper. Such a tool will automatically fetch, build and install the AUR package, resolve dependencies (including dependencies on AUR packages), and ease the upgrade procedure.

If you have any difficulties with installation, check the repository package page.

Note: The OCRmyPDF AUR package currently omits the JBIG2 encoder. OCRmyPDF works fine without it but will produce larger output files. The encoder is available from the jbig2enc-git AUR package and may be installed using the same series of steps as for the installation OCRmyPDF AUR package. Alternatively, it may be built manually from source following the instructions in Installing the JBIG2 encoder. If JBIG2 is installed, OCRmyPDF 7.0.0 and later will automatically detect it.

3.1.7 Alpine Linux

To install OCRmyPDF for Alpine Linux:

```
apk add ocrmypdf
```

3.1.8 Mageia 7

There is no OS-level packaging available for Mageia, so you must install the dependencies:

```
# As root user
urpmi.update -a
urpmi \
    ghostscript \
    icc-profiles-openicc \
    jbig2dec \
    lib64leptonica5 \
    pngquant \
    python3-pip \
    python3-cffi \
    python3-distutils-extra \
    python3-pkg-resources \
    python3-reportlab \
    qpdf \
    tesseract \
    tesseract-osd \
    tesseract-eng \
    tesseract-fra
```

To install ocrmypdf for the system:

```
# As root user
pip3 install ocrmypdf
ldconfig
```

Or, to install for the current user only:

```
export PATH=$HOME/.local/bin:$PATH
pip3 install --user ocrmypdf
```

3.1.9 Other Linux packages

See the Repology page.

In general, first install the OCRmyPDF package for your system, then optionally use the procedure *Installing with Python pip* to install a more recent version.

3.2 Installing on macOS

3.2.1 Homebrew

OCRmyPDF is now a standard Homebrew formula. To install on macOS:

```
brew install ocrmypdf
```

This will include only the English language pack. If you need other languages you can optionally install them all:

```
brew install tesseract-lang  # Optional: Install all language packs
```

Note: Users who previously installed OCRmyPDF on macOS using `pip install ocrmypdf` should remove the pip version (`pip3 uninstall ocrmypdf`) before switching to the Homebrew version.

Note: Users who previously installed OCRmyPDF from the private tap should switch to the mainline version (`brew untap jbarlow83/ocrmypdf`) and install from there.

3.2.2 Manual installation on macOS

These instructions probably work on all macOS supported by Homebrew, and are for installing a more current version of OCRmyPDF than is available from Homebrew. Note that the Homebrew versions usually track the release versions fairly closely.

If it's not already present, install Homebrew.

Update Homebrew:

```
brew update
```

Install or upgrade the required Homebrew packages, if any are missing. To do this, use `brew edit ocrmypdf` to obtain a recent list of Homebrew dependencies. You could also check the `.workflows/build.yml`.

This will include the English, French, German and Spanish language packs. If you need other languages you can optionally install them all:

```
brew install tesseract-lang  # Option 2: for all language packs
```

Update the homebrew pip:

```
pip3 install --upgrade pip
```

You can then install OCRmyPDF from PyPI, for the current user:

```
pip3 install --user ocrmypdf
```

or system-wide:

```
pip3 install ocrmypdf
```

The command line program should now be available:

```
ocrmypdf --help
```

3.3 Installing on Windows

3.3.1 Native Windows

Note: Administrator privileges will be required for some of these steps.

You must install the following for Windows:

- Python 3.7 (64-bit) or later
- Tesseract 4.0 or later
- Ghostscript 9.50 or later

Using the Chocolatey package manager, install the following when running in an Administrator command prompt:

- `choco install python3`
- `choco install --pre tesseract`
- `choco install ghostscript`
- `choco install pngquant` (optional)

The commands above will install Python 3.x (latest version), Tesseract, Ghostscript and pngquant. Chocolatey may also need to install the Windows Visual C++ Runtime DLLs or other Windows patches, and may require a reboot.

You may then use `pip` to install ocrmypdf. (This can performed by a user or Administrator.):

- `pip install ocrmypdf`

Chocolatey automatically selects appropriate versions of these applications. If you are installing them manually, please install 64-bit versions of all applications for 64-bit Windows, or 32-bit versions of all applications for 32-bit Windows. Mixing the "bitness" of these programs will lead to errors.

OCRmyPDF will check the Windows Registry and standard locations in your Program Files for third party software it needs (specifically, Tesseract and Ghostscript). To override the versions OCRmyPDF selects, you can modify the `PATH` environment variable. Follow these directions to change the PATH.

Warning: As of early 2021, users have reported problems with the Microsoft Store version of Python and OCRmyPDF. These issues affect many other third party Python packages. Please download Python from Python.org or Chocolatey instead, and do not use the Microsoft Store version.

3.3.2 Windows Subsystem for Linux

1. Install Ubuntu 18.04 for Windows Subsystem for Linux, if not already installed.
2. Follow the procedure to install *OCRmyPDF on Ubuntu 18.04*.
3. Open the Windows command prompt and create a symlink:

```
wsl sudo ln -s  /home/$USER/.local/bin/ocrmypdf /usr/local/bin/ocrmypdf
```

Then confirm that the expected version from PyPI () is installed:

```
wsl ocrmypdf --version
```

You can then run OCRmyPDF in the Windows command prompt or Powershell, prefixing `wsl`, and call it from Windows programs or batch files.

3.3.3 Cygwin64

First install the the following prerequisite Cygwin packages using `setup-x86_64.exe`:

```
python36 (or later)
python3?-devel
python3?-pip
python3?-lxml
python3?-imaging

   (where 3? means match the version of python3 you installed)

gcc-g++
ghostscript (<=9.50 or >=9.52-2 see note below)
libexempi3
libexempi-devel
libffi6
libffi-devel
pngquant
qpdf
libqpdf-devel
tesseract-ocr
tesseract-ocr-devel
```

Note: The Cygwin package for Ghostscript in versions 9.52 and 9.52-1 contained a bug that caused an exception to occur when ocrmypdf invoked gs. Make sure you have either 9.50 (or earlier) or 9.52-2 (or later).

Then open a Cygwin terminal (i.e. `mintty`), run the following commands. Note that if you are using the version of `pip` that was installed with the Cygwin Python package, the command name will be `pip3`. If you have since updated `pip` (with, for instance `pip3 install --upgrade pip`) the the command is likely just `pip` instead of `pip3`:

```
pip3 install wheel
pip3 install ocrmypdf
```

The optional dependency "unpaper" that is currently not available under Cygwin. Without it, certain options such as `--clean` will produce an error message. However, the OCR-to-text-layer functionality is available.

3.3.4 Docker

You can also *Install the Docker* container on Windows. Ensure that your command prompt can run the docker "hello world" container.

3.4 Installing on FreeBSD

FreeBSD 11.3, 12.0, 12.1-RELEASE and 13.0-CURRENT are supported. Other versions likely work but have not been tested.

```
pkg install py38-ocrmypdf
```

To install a more recent version, you could attempt to first install the system version with `pkg`, then use `pip install --user ocrmypdf`.

3.5 Installing the Docker image

For some users, installing the Docker image will be easier than installing all of OCRmyPDF's dependencies.

See *OCRmyPDF Docker image* for more information.

3.6 Installing with Python pip

OCRmyPDF is delivered by PyPI because it is a convenient way to install the latest version. However, PyPI and `pip` cannot address the fact that `ocrmypdf` depends on certain non-Python system libraries and programs being installed.

Warning: Debian and Ubuntu users: unfortunately, Debian and Ubuntu customize Python in non-standard ways, and the nature of these customizations varies from release to release. This can make for a frustrating user experience. The instructions below work on almost all platforms that have Python installed, except for Debian and Ubuntu, where you may need to take additional steps. For best results on Debian and Ubuntu, use the `apt` packages; or if these are too old, run `apt install python3-pip python3-venv`, create a virtual environment, and install OCRmyPDF in that environment.

See here for more inforation on Debian-Python issues.

For best results, first install your platform's version of `ocrmypdf`, using the instructions elsewhere in this document. Then you can use `pip` to get the latest version if your platform version is out of date. Chances are that this will satisfy most dependencies.

Use `ocrmypdf --version` to confirm what version was installed.

Then you can install the latest OCRmyPDF from the Python wheels. First try:

```
pip3 install --user ocrmypdf
```

You should then be able to run `ocrmypdf --version` and see that the latest version was located.

Since `pip3 install --user` does not work correctly on some platforms, notably Ubuntu 16.04 and older, and the Homebrew version of Python, instead use this for a system wide installation:

```
pip3 install ocrmypdf
```

Note: AArch64 (ARM64) users: this process will be difficult because most Python packages are not available as binary wheels for your platform. You're probably better off using a platform install on Debian, Ubuntu, or Fedora.

3.6.1 Requirements for pip and HEAD install

OCRmyPDF currently requires these external programs and libraries to be installed, and must be satisfied using the operating system package manager. `pip` cannot provide them.

- Python 3.6 or newer
- Ghostscript 9.15 or newer
- qpdf 8.1.0 or newer
- Tesseract 4.0.0-beta or newer

As of ocrmypdf 7.2.1, the following versions are recommended:

- Python 3.7 or 3.8
- Ghostscript 9.23 or newer
- qpdf 8.2.1
- Tesseract 4.0.0 or newer
- jbig2enc 0.29 or newer
- pngquant 2.5 or newer
- unpaper 6.1

jbig2enc, pngquant, and unpaper are optional. If missing certain features are disabled. OCRmyPDF will discover them as soon as they are available.

jbig2enc, if present, will be used to optimize the encoding of monochrome images. This can significantly reduce the file size of the output file. It is not required. jbig2enc is not generally available for Ubuntu or Debian due to lingering concerns about patent issues, but can easily be built from source. To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

pngquant, if present, is optionally used to optimize the encoding of PNG-style images in PDFs (actually, any that are that losslessly encoded) by lossily quantizing to a smaller color palette. It is only activated then the `--optimize` argument is `2` or `3`.

unpaper, if present, enables the `--clean` and `--clean-final` command line options.

These are in addition to the Python packaging dependencies, meaning that unfortunately, the `pip install` command cannot satisfy all of them.

3.7 Installing HEAD revision from sources

If you have `git` and Python 3.6 or newer installed, you can install from source. When the `pip` installer runs, it will alert you if dependencies are missing.

If you prefer to build every from source, you will need to build pikepdf from source. First ensure you can build and install pikepdf.

To install the HEAD revision from sources in the current Python 3 environment:

```
pip3 install git+https://github.com/jbarlow83/OCRmyPDF.git
```

Or, to install in development mode, allowing customization of OCRmyPDF, use the `-e` flag:

```
pip3 install -e git+https://github.com/jbarlow83/OCRmyPDF.git
```

You may find it easiest to install in a virtual environment, rather than system-wide:

```
git clone -b master https://github.com/jbarlow83/OCRmyPDF.git
python3 -m venv
source venv/bin/activate
cd OCRmyPDF
pip3 install .
```

However, `ocrmypdf` will only be accessible on the system PATH when you activate the virtual environment.

To run the program:

```
ocrmypdf --help
```

If not yet installed, the script will notify you about dependencies that need to be installed. The script requires specific versions of the dependencies. Older version than the ones mentioned in the release notes are likely not to be compatible to OCRmyPDF.

3.7.1 For development

To install all of the development and test requirements:

```
git clone -b master https://github.com/jbarlow83/OCRmyPDF.git
python3 -m venv
source venv/bin/activate
cd OCRmyPDF
pip install -e .[test]
```

To add JBIG2 encoding, see *Installing the JBIG2 encoder*.

3.8 Shell completions

Completions for `bash` and `fish` are available in the project's `misc/completion` folder. The `bash` completions are likely `zsh` compatible but this has not been confirmed. Package maintainers, please install these at the appropriate locations for your system.

To manually install the `bash` completion, copy `misc/completion/ocrmypdf.bash` to `/etc/bash_completion.d/ocrmypdf` (rename the file).

To manually install the `fish` completion, copy `misc/completion/ocrmypdf.fish` to `~/.config/fish/completions/ocrmypdf.fish`.

CHAPTER

FOUR

PDF OPTIMIZATION

OCRmyPDF includes an image-oriented PDF optimizer. By default, the optimizer runs with safe settings with the goal of improving compression at no loss of quality. At higher optimization levels, lossy optimizations may be applied and tuned. Optimization occurs after OCR, and only if OCR succeeded. It does not perform other possible optimizations such as deduplicating resources, consolidating fonts, simplifying vector drawings, or anything of that nature.

Optimization ranges from `-O0` through `-O3`, where `0` disables optimization and `3` implements all options. `1`, the default, performs only safe and lossless optimizations. (This is similar to GCC's optimization parameter.) The exact type of optimizations performed will vary over time.

PDF optimization requires third-party, optional tools for certain optimizations. If these are not installed or cannot be found by OCRmyPDF, optimization will not be as good.

4.1 Optimizations that always occurs

OCRmyPDF will automatically replace obsolete or inferior compression schemes such as RLE or LZW with superior schemes such as Deflate and converting monochrome images to CCITT G4. Since this is harmless it always occurs and there is no way to disable it. Other non-image compressed objects are compressed as well.

4.2 Fast web view

OCRmyPDF automatically optimizes PDFs for "fast web view" in Adobe Acrobat's parlance, or equivalently, linearizes PDFs so that the resources they reference are presented in the order a viewer needs them for sequential display. This reduces the latency of viewing a PDF both online and from local storage. This actually slightly increases the file size.

To disable this optimization and all others, use `ocrmypdf --optimize 0 ...` or the shorthand `-O0`.

4.3 Lossless optimizations

At optimization level `-O1` (the default), OCRmyPDF will also attempt lossless image optimization.

If a JBIG2 encoder is available, then monochrome images will be converted to JBIG2, with the potential for huge savings on large black and white images, since JBIG2 is far more efficient than any other monochrome (bi-level) compression. (All known US patents related to JBIG2 have probably expired, but it remains the responsibility of the user to supply a JBIG2 encoder such as jbig2enc. OCRmyPDF does not implement JBIG2 encoding on its own.)

OCRmyPDF currently does not attempt to recompress losslessly compressed objects more aggressively.

4.4 Lossy optimizations

At optimization level `-O2` and `-O3`, OCRmyPDF will some attempt lossy image optimization.

If `pngquant` is installed, OCRmyPDF will use it to perform quantize paletted images to reduce their size.

The quality of JPEGs may be lowered, on the assumption that a lower quality image may be suitable for storage after OCR.

It is not possible to optimize all image types. Uncommon image types may be skipped by the optimizer.

OCRmyPDF provides *lossy mode JBIG2* as an advanced feature that additional requires the argument `--jbig2-lossy`.

CHAPTER

FIVE

INSTALLING ADDITIONAL LANGUAGE PACKS

OCRmyPDF uses Tesseract for OCR, and relies on its language packs for all languages. On most platforms, English is installed with Tesseract by default, but not always.

Tesseract supports most languages. Languages are identified by standardized three-letter codes (called ISO 639-2 Alpha-3). Tesseract's documentation also lists the three-letter code for your language. Some are anglicized, e.g. Spanish is `spa` rather than `esp`, while others are not, e.g. German is `deu` and French is `fra`.

After you have installed a language pack, you can use it with `ocrmypdf -l <language>`, for example `ocrmypdf -l spa`. For multilingual documents, you can specify all languages to be expected, e.g. `ocrmypdf -l eng+fra` for English and French. English is assumed by default unless other language(s) are specified.

For Linux users, you can often find packages that provide language packs:

5.1 Debian and Ubuntu users

```
# Display a list of all Tesseract language packs
apt-cache search tesseract-ocr

# Install Chinese Simplified language pack
apt-get install tesseract-ocr-chi-sim
```

You can then pass the `-l LANG` argument to OCRmyPDF to give a hint as to what languages it should search for. Multiple languages can be requested using either `-l eng+fra` (English and French) or `-l eng -l fra`.

5.2 Fedora users

```
# Display a list of all Tesseract language packs
dnf search tesseract

# Install Chinese Simplified language pack
dnf install tesseract-langpack-chi_sim
```

You can then pass the `-l LANG` argument to OCRmyPDF to give a hint as to what languages it should search for. Multiple languages can be requested using either `-l eng+fra` (English and French) or `-l eng -l fra`.

5.3 macOS users

You can install additional language packs by *installing Tesseract using Homebrew with all language packs*.

5.4 Docker users

Users of the OCRmyPDF Docker image should install language packs into a derived Docker image as *described in that section*.

5.5 Windows users

The Tesseract installer provided by Chocolatey currently includes only English language. To install other languages, download the respective language pack (`.traineddata` file) from https://github.com/tesseract-ocr/tessdata/ and place it in `C:\\Program Files\\Tesseract-OCR\\tessdata` (or wherever Tesseract OCR is installed).

CHAPTER

SIX

INSTALLING THE JBIG2 ENCODER

Most Linux distributions do not include a JBIG2 encoder since JBIG2 encoding was patented for a long time. All known JBIG2 US patents have expired as of 2017, but it is possible that unknown patents exist.

JBIG2 encoding is recommended for OCRmyPDF and is used to losslessly create smaller PDFs. If JBIG2 encoding is not available, lower quality encodings will be used.

JBIG2 decoding is not patented and is performed automatically by most PDF viewers. It is widely supported and has been part of the PDF specification since 2001.

On macOS, Homebrew packages jbig2enc and OCRmyPDF includes it by default. The Docker image for OCRmyPDF also builds its own JBIG2 encoder from source.

For all other Linux, you must build a JBIG2 encoder from source:

```
git clone https://github.com/agl/jbig2enc
cd jbig2enc
./autogen.sh
./configure && make
[sudo] make install
```

6.1 Lossy mode JBIG2

OCRmyPDF provides lossy mode JBIG2 as an advanced feature. Users should review the technical concerns with JBIG2 in lossy mode and decide if this feature is acceptable for their use case.

JBIG2 lossy mode does achieve higher compression ratios than any other monochrome (bitonal) compression technology; for large text documents the savings are considerable. JBIG2 lossless still gives great compression ratios and is a major improvement over the older CCITT G4 standard. As explained above, there is some risk of substitution errors.

To turn on JBIG2 lossy mode, add the argument `--jbig2-lossy`. `--optimize {1,2,3}` are necessary for the argument to take effect also required. Also, a JBIG2 encoder must be installed as described in the previous section.

Due to an oversight, ocrmypdf v7.0 and v7.1 used lossy mode by default.

CHAPTER

SEVEN

COOKBOOK

7.1 Basic examples

7.1.1 Help!

ocrmypdf has built-in help.

```
ocrmypdf --help
```

7.1.2 Add an OCR layer and convert to PDF/A

```
ocrmypdf input.pdf output.pdf
```

7.1.3 Add an OCR layer and output a standard PDF

```
ocrmypdf --output-type pdf input.pdf output.pdf
```

7.1.4 Create a PDF/A with all color and grayscale images converted to JPEG

```
ocrmypdf --output-type pdfa --pdfa-image-compression jpeg input.pdf output.pdf
```

7.1.5 Modify a file in place

The file will only be overwritten if OCRmyPDF is successful.

```
ocrmypdf myfile.pdf myfile.pdf
```

7.1.6 Correct page rotation

OCR will attempt to automatic correct the rotation of each page. This can help fix a scanning job that contains a mix of landscape and portrait pages.

```
ocrmypdf --rotate-pages myfile.pdf myfile.pdf
```

You can increase (decrease) the parameter `--rotate-pages-threshold` to make page rotation more (less) aggressive. The threshold number is the ratio of how confidence the OCR engine is that the document image should be changed, compared to kept the same. The default value is quite conservative; on some files it may not attempt rotations at all unless it is very confident that the current rotation is wrong. A lower value of `2.0` will produce more rotations, and more false positives. Run with `-v1` to see the confidence level for each page to see if there may be a better value for your files.

If the page is "just a little off horizontal", like a crooked picture, then you want `--deskew`. `--rotate-pages` is for when the cardinal angle is wrong.

7.1.7 OCR languages other than English

OCRmyPDF assumes the document is in English unless told otherwise. OCR quality may be poor if the wrong language is used.

```
ocrmypdf -l fra LeParisien.pdf LeParisien.pdf
ocrmypdf -l eng+fra Bilingual-English-French.pdf Bilingual-English-French.pdf
```

Language packs must be installed for all languages specified. See *Installing additional language packs*.

Unfortunately, the Tesseract OCR engine has no ability to detect the language when it is unknown.

7.1.8 Produce PDF and text file containing OCR text

This produces a file named "output.pdf" and a companion text file named "output.txt".

```
ocrmypdf --sidecar output.txt input.pdf output.pdf
```

Note: The sidecar file contains the **OCR text** found by OCRmyPDF. If the document contains pages that already have text, that text will not appear in the sidecar. If the option `--pages` is used, only those pages on which OCR was performed will be included in the sidecar. If certain pages were skipped because of options like `--skip-big` or `--tesseract-timeout`, those pages will not be in the sidecar.

To extract all text from a PDF, whether generated from OCR or otherwise, use a program like Poppler's `pdftotext` or `pdfgrep`.

7.1.9 OCR images, not PDFs

Option: use Tesseract

If you are starting with images, you can just use Tesseract directly to convert images to PDFs:

```
tesseract my-image.jpg output-prefix pdf
```

```
# When there are multiple images
tesseract text-file-containing-list-of-image-filenames.txt output-prefix pdf
```

Tesseract's PDF output is quite good – OCRmyPDF uses it internally, in some cases. However, OCRmyPDF has many features not available in Tesseract like image processing, metadata control, and PDF/A generation.

Option: use img2pdf

You can also use a program like img2pdf to convert your images to PDFs, and then pipe the results to run ocrmypdf. The `-` tells ocrmypdf to read standard input.

```
img2pdf my-images*.jpg | ocrmypdf - myfile.pdf
```

`img2pdf` is recommended because it does an excellent job at generating PDFs without transcoding images.

Option: use OCRmyPDF (single images only)

For convenience, OCRmyPDF can also convert single images to PDFs on its own. If the resolution (dots per inch, DPI) of an image is not set or is incorrect, it can be overridden with `--image-dpi`. (As 1 inch is 2.54 cm, 1 dpi = 0.39 dpcm).

```
ocrmypdf --image-dpi 300 image.png myfile.pdf
```

If you have multiple images, you must use `img2pdf` to convert the images to PDF.

Not recommended

We caution against using ImageMagick or Ghostscript to convert images to PDF, since they may transcode images or produce downsampled images, sometimes without warning.

7.2 Image processing

OCRmyPDF perform some image processing on each page of a PDF, if desired. The same processing is applied to each page. It is suggested that the user review files after image processing as these commands might remove desirable content, especially from poor quality scans.

- `--rotate-pages` attempts to determine the correct orientation for each page and rotates the page if necessary.
- `--remove-background` attempts to detect and remove a noisy background from grayscale or color images. Monochrome images are ignored. This should not be used on documents that contain color photos as it may remove them.
- `--deskew` will correct pages were scanned at a skewed angle by rotating them back into place. Skew determination and correction is performed using Postl's variance of line sums algorithm as implemented in Leptonica.

- `--clean` uses unpaper to clean up pages before OCR, but does not alter the final output. This makes it less likely that OCR will try to find text in background noise.
- `--clean-final` uses unpaper to clean up pages before OCR and inserts the page into the final output. You will want to review each page to ensure that unpaper did not remove something important.

Note: In many cases image processing will rasterize PDF pages as images, potentially losing quality.

Warning: `--clean-final` and `-remove-background` may leave undesirable visual artifacts in some images where their algorithms have shortcomings. Files should be visually reviewed after using these options.

7.2.1 Example: OCR and correct document skew (crooked scan)

Deskew:

```
ocrmypdf --deskew input.pdf output.pdf
```

Image processing commands can be combined. The order in which options are given does not matter. OCRmyPDF always applies the steps of the image processing pipeline in the same order (rotate, remove background, deskew, clean).

```
ocrmypdf --deskew --clean --rotate-pages input.pdf output.pdf
```

7.3 Don't actually OCR my PDF

If you set `--tesseract-timeout 0` OCRmyPDF will apply its image processing without performing OCR, if all you want to is to apply image processing or PDF/A conversion.

```
ocrmypdf --tesseract-timeout=0 --remove-background input.pdf output.pdf
```

7.3.1 Optimize images without performing OCR

You can also optimize all images without performing any OCR:

```
ocrmypdf --tesseract-timeout=0 --optimize 3 --skip-text input.pdf output.pdf
```

7.3.2 Perform OCR only certain pages

You can ask OCRmyPDF to only apply OCR to certain pages.

```
ocrmypdf --pages 2,3,13-17 input.pdf output.pdf
```

Hyphens denote a range of pages and commas separate page numbers. If you prefer to use spaces, quote all of the page numbers: `--pages '2, 3, 5, 7'`.

OCRmyPDF will warn if your list of page numbers contains duplicates or overlap pages. OCRmyPDF does not currently account for document page numbers, such as an introduction section of a book that uses Roman numerals. It simply counts the number of virtual pieces of paper since the start.

Regardless of the argument to `--pages`, OCRmyPDF will optimize all pages in the file and convert it to PDF/A, unless you disable those options. In this example, we want to OCR only the title and otherwise change the PDF as little as possible:

```
ocrmypdf --pages 1 --output-type pdf --optimize 0 input.pdf output.pdf
```

7.4 Redo existing OCR

To redo OCR on a file OCRed with other OCR software or a previous version of OCRmyPDF and/or Tesseract, you may use the `--redo-ocr` argument. (Normally, OCRmyPDF will exit with an error if asked to modify a file with OCR.)

This may be helpful for users who want to take advantage of accuracy improvements in Tesseract 4.0 for files they previously OCRed with an earlier version of Tesseract and OCRmyPDF.

```
ocrmypdf --redo-ocr input.pdf output.pdf
```

This method will replace OCR without rasterizing, reducing quality or removing vector content. If a file contains a mix of pure digital text and OCR, digital text will be ignored and OCR will be replaced. As such this mode is incompatible with image processing options, since they alter the appearance of the file.

In some cases, existing OCR cannot be detected or replaced. Files produced by OCRmyPDF v2.2 or earlier, for example, are internally represented as having visible text with an opaque image drawn on top. This situation cannot be detected.

If `--redo-ocr` does not work, you can use `--force-ocr`, which will force rasterization of all pages, potentially reducing quality or losing vector content.

7.5 Improving OCR quality

The *Image processing* features can improve OCR quality.

Rotating pages and deskewing helps to ensure that the page orientation is correct before OCR begins. Removing the background and/or cleaning the page can also improve results. The `--oversample DPI` argument can be specified to resample images to higher resolution before attempting OCR; this can improve results as well.

OCR quality will suffer if the resolution of input images is not correct (since the range of pixel sizes that will be checked for possible fonts will also be incorrect).

7.6 PDF optimization

By default OCRmyPDF will attempt to perform lossless optimizations on the images inside PDFs after OCR is complete. Optimization is performed even if no OCR text is found.

The `--optimize N` (short form `-O`) argument controls optimization, where `N` ranges from 0 to 3 inclusive, analogous to the optimization levels in the GCC compiler.

Level	Comments
`--optimize 0`	Disables optimization.
`--optimize 1`	Enables lossless optimizations, such as transcoding images to more efficient formats. Also compress other uncompressed objects in the PDF and enables the more efficient "object streams" within the PDF.
`--optimize 2`	All of the above, and enables lossy optimizations and color quantization.
`--optimize 3`	All of the above, and enables more aggressive optimizations and targets lower image quality.

Optimization is improved when a JBIG2 encoder is available and when `pngquant` is installed. If either of these components are missing, then some types of images cannot be optimized.

The types of optimization available may expand over time. By default, OCRmyPDF compresses data streams inside PDFs, and will change inefficient compression modes to more modern versions. A program like `qpdf` can be used to change encodings, e.g. to inspect the internals fo a PDF.

```
ocrmypdf --optimize 3 in.pdf out.pdf  # Make it small
```

Some users may consider enabling lossy JBIG2. See: *Lossy mode JBIG2*.

CHAPTER

EIGHT

OCRMYPDF DOCKER IMAGE

OCRmyPDF is also available in a Docker image that packages recent versions of all dependencies.

For users who already have Docker installed this may be an easy and convenient option. However, it is less performant than a system installation and may require Docker engine configuration.

OCRmyPDF needs a generous amount of RAM, CPU cores, temporary storage space, whether running in a Docker container or on its own. It may be necessary to ensure the container is provisioned with additional resources.

8.1 Installing the Docker image

If you have Docker installed on your system, you can install a Docker image of the latest release.

If you can run this command successfully, your system is ready to download and execute the image:

```
docker run hello-world
```

The recommended OCRmyPDF Docker image is currently named `ocrmypdf`:

```
docker pull jbarlow83/ocrmypdf
```

OCRmyPDF will use all available CPU cores. By default, the VirtualBox machine instance on Windows and macOS has only a single CPU core enabled. Use the VirtualBox Manager to determine the name of your Docker engine host, and then follow these optional steps to enable multiple CPUs:

```
# Optional step for Mac OS X users
docker-machine stop "yourVM"
VBoxManage modifyvm "yourVM" --cpus 2  # or whatever number of core is desired
docker-machine start "yourVM"
eval $(docker-machine env "yourVM")
```

See the Docker documentation for adjusting memory and CPU on other platforms.

8.2 Using the Docker image on the command line

Unlike typical Docker containers, in this section the OCRmyPDF Docker container is emphemeral – it runs for one OCR job and terminates, just like a command line program. We are using Docker to deliver an application (as opposed to the more conventional case, where a Docker container runs as a server).

To start a Docker container (instance of the image):

```
docker tag jbarlow83/ocrmypdf ocrmypdf
docker run --rm -i ocrmypdf (... all other arguments here...) - -
```

For convenience, create a shell alias to hide the Docker command. It is easier to send the input file as stdin and read the output from stdout – **this avoids the messy permission issues with Docker entirely**.

```
alias docker_ocrmypdf='docker run --rm -i ocrmypdf'
docker_ocrmypdf --version  # runs docker version
docker_ocrmypdf - - <input.pdf >output.pdf
```

Or in the wonderful fish shell:

```
alias docker_ocrmypdf 'docker run --rm ocrmypdf'
funcsave docker_ocrmypdf
```

Alternately, you could mount the local current working directory as a Docker volume:

```
alias docker_ocrmypdf='docker run --rm  -i --user "$(id -u):$(id -g)" --workdir /data -v
↪"$PWD:/data" ocrmypdf'
docker_ocrmypdf /data/input.pdf /data/output.pdf
```

8.3 Adding languages to the Docker image

By default the Docker image includes English, German, Simplified Chinese, French, Portuguese and Spanish, the most popular languages for OCRmyPDF users based on feedback. You may add other languages by creating a new Dockerfile based on the public one.

```
FROM jbarlow83/ocrmypdf

# Example: add Italian
RUN apt install tesseract-ocr-ita
```

To install language packs (training data) such as the tessdata_best suite or custom data, you first need to determine the version of Tesseract data files, which may differ from the Tesseract program version. Use this command to determine the data file version:

```
docker run -i --rm --entrypoint /bin/ls jbarlow83/ocrmypdf /usr/share/tesseract-ocr
```

As of 2021, the data file version is probably `4.00`.

You can then add new data with either a Dockerfile:

```
FROM jbarlow83/ocrmypdf
```

(continues on next page)

(continued from previous page)

```
# Example: add a tessdata_best file
COPY chi_tra_vert.traineddata /usr/share/tesseract-ocr/<data version>/tessdata/
```

Alternately, you can copy training data into a Docker container as follows:

```
docker cp mycustomtraining.traineddata name_of_container:/usr/share/tesseract-ocr/
↪<tesseract version>/tessdata/
```

8.4 Executing the test suite

The OCRmyPDF test suite is installed with image. To run it:

```
docker run --entrypoint python3  jbarlow83/ocrmypdf -m pytest
```

8.5 Accessing the shell

To use the bash shell in the Docker image:

```
docker run -it --entrypoint bash  jbarlow83/ocrmypdf
```

8.6 Using the OCRmyPDF web service wrapper

The OCRmyPDF Docker image includes an example, barebones HTTP web service. The webservice may be launched as follows:

```
docker run --entrypoint python3 -p 5000:5000  jbarlow83/ocrmypdf webservice.py
```

This will configure the machine to listen on port 5000. On Linux machines this is port 5000 of localhost. On macOS or Windows machines running Docker, this is port 5000 of the virtual machine that runs your Docker images. You can find its IP address using the command `docker-machine ip`.

Unlike command line usage this program will open a socket and wait for connections.

Warning: The OCRmyPDF web service wrapper is intended for demonstration or development. It provides no security, no authentication, no protection against denial of service attacks, and no load balancing. The default Flask WSGI server is used, which is intended for development only. The server is single-threaded and so can respond to only one client at a time. While running OCR, it cannot respond to any other clients.

Clients must keep their open connection while waiting for OCR to complete. This may entail setting a long timeout; this interface is more useful for internal HTTP API calls.

Unlike the rest of OCRmyPDF, this web service is licensed under the Affero GPLv3 (AGPLv3) since Ghostscript is also licensed in this way.

In addition to the above, please read our *general remarks on using OCRmyPDF as a service*.

CHAPTER

NINE

ADVANCED FEATURES

9.1 Control of unpaper

OCRmyPDF uses `unpaper` to provide the implementation of the `--clean` and `--clean-final` arguments. unpaper provides a variety of image processing filters to improve images.

By default, OCRmyPDF uses only `unpaper` arguments that were found to be safe to use on almost all files without having to inspect every page of the file afterwards. This is particularly true when only `--clean` is used, since that instructs OCRmyPDF to only clean the image before OCR and not the final image.

However, if you wish to use the more aggressive options in `unpaper`, you may use `--unpaper-args '...'` to override the OCRmyPDF's defaults and forward other arguments to unpaper. This option will forward arguments to `unpaper` without any knowledge of what that program considers to be valid arguments. The string of arguments must be quoted as shown in the examples below. No filename arguments may be included. OCRmyPDF will assume it can append input and output filename of intermediate images to the `--unpaper-args` string.

In this example, we tell `unpaper` to expect two pages of text on a sheet (image), such as occurs when two facing pages of a book are scanned. `unpaper` uses this information to deskew each independently and clean up the margins of both.

```
ocrmypdf --clean --clean-final --unpaper-args '--layout double' input.pdf output.pdf
ocrmypdf --clean --clean-final --unpaper-args '--layout double --no-noisefilter' input.
↪pdf output.pdf
```

Warning: Some `unpaper` features will reposition text within the image. `--clean-final` is recommended to avoid this issue.

Warning: Some `unpaper` features cause multiple input or output files to be consumed or produced. OCRmyPDF requires `unpaper` to consume one file and produce one file. An deviation from that condition will result in errors.

Note: `unpaper` uses uncompressed PBM/PGM/PPM files for its intermediate files. For large images or documents, it can take a lot of temporary disk space.

9.2 Control of OCR options

OCRmyPDF provides many features to control the behavior of the OCR engine, Tesseract.

9.2.1 When OCR is skipped

If a page in a PDF seems to have text, by default OCRmyPDF will exit without modifying the PDF. This is to ensure that PDFs that were previously OCRed or were "born digital" rather than scanned are not processed.

If `--skip-text` is issued, then no OCR will be performed on pages that already have text. The page will be copied to the output. This may be useful for documents that contain both "born digital" and scanned content, or to use OCRmyPDF to normalize and convert to PDF/A regardless of their contents.

If `--redo-ocr` is issued, then a detailed text analysis is performed. Text is categorized as either visible or invisible. Invisible text (OCR) is stripped out. Then an image of each page is created with visible text masked out. The page image is sent for OCR, and any additional text is inserted as OCR. If a file contains a mix of text and bitmap images that contain text, OCRmyPDF will locate the additional text in images without disrupting the existing text.

If `--force-ocr` is issued, then all pages will be rasterized to images, discarding any hidden OCR text, and rasterizing any printable text. This is useful for redoing OCR, for fixing OCR text with a damaged character map (text is selectable but not searchable), and destroying redacted information. Any forms and vector graphics will be rasterized as well.

9.2.2 Time and image size limits

By default, OCRmyPDF permits tesseract to run for three minutes (180 seconds) per page. This is usually more than enough time to find all text on a reasonably sized page with modern hardware.

If a page is skipped, it will be inserted without OCR. If preprocessing was requested, the preprocessed image layer will be inserted.

If you want to adjust the amount of time spent on OCR, change `--tesseract-timeout`. You can also automatically skip images that exceed a certain number of megapixels with `--skip-big`. (A 300 DPI, 8.5×11" page is 8.4 megapixels.)

```
# Allow 300 seconds for OCR; skip any page larger than 50 megapixels
ocrmypdf --tesseract-timeout 300 --skip-big 50 bigfile.pdf output.pdf
```

9.2.3 Overriding default tesseract

OCRmyPDF checks the system `PATH` for the `tesseract` binary.

Some relevant environment variables that influence Tesseract's behavior include:

TESSDATA_PREFIX
: Overrides the path to Tesseract's data files. This can allow simultaneous installation of the "best" and "fast" training data sets. OCRmyPDF does not manage this environment variable.

OMP_THREAD_LIMIT
: Controls the number of threads Tesseract will use. OCRmyPDF will manage this environment variable if it is not already set.

For example, if you have a development build of Tesseract don't wish to use the system installation, you can launch OCRmyPDF as follows:

```
env \
    PATH=/home/user/src/tesseract/api:$PATH \
    TESSDATA_PREFIX=/home/user/src/tesseract \
    ocrmypdf input.pdf output.pdf
```

In this example `TESSDATA_PREFIX` is required to redirect Tesseract to an alternate folder for its "tessdata" files.

9.2.4 Overriding other support programs

In addition to tesseract, OCRmyPDF uses the following external binaries:

- `gs` (Ghostscript)
- `unpaper`
- `pngquant`
- `jbig2`

In each case OCRmyPDF will search the `PATH` environment variable to locate the binaries.

9.2.5 Changing tesseract configuration variables

You can override tesseract's default control parameters with a configuration file.

As an example, this configuration will disable Tesseract's dictionary for current language. Normally the dictionary is helpful for interpolating words that are unclear, but it may interfere with OCR if the document does not contain many words (for example, a list of part numbers).

Create a file named "no-dict.cfg" with these contents:

```
load_system_dawg 0
language_model_penalty_non_dict_word 0
language_model_penalty_non_freq_dict_word 0
```

then run ocrmypdf as follows (along with any other desired arguments):

```
ocrmypdf --tesseract-config no-dict.cfg input.pdf output.pdf
```

> **Warning:** Some combinations of control parameters will break Tesseract or break assumptions that OCRmyPDF makes about Tesseract's output.

9.3 Changing the PDF renderer

rasterizing Converting a PDF to an image for display.

rendering Creating a new PDF from other data (such as an existing PDF).

OCRmyPDF has these PDF renderers: `sandwich` and `hocr`. The renderer may be selected using `--pdf-renderer`. The default is `auto` which lets OCRmyPDF select the renderer to use. Currently, `auto` always selects `sandwich`.

9.3.1 The `sandwich` renderer

The `sandwich` renderer uses Tesseract's new text-only PDF feature, which produces a PDF page that lays out the OCR in invisible text. This page is then "sandwiched" onto the original PDF page, allowing lossless application of OCR even to PDF pages that contain other vector objects.

Currently this is the best renderer for most uses, however it is implemented in Tesseract so OCRmyPDF cannot influence it. Currently some problematic PDF viewers like Mozilla PDF.js and macOS Preview have problems with segmenting its text output, and mightrunseveralwordstogether.

When image preprocessing features like `--deskew` are used, the original PDF will be rendered as a full page and the OCR layer will be placed on top.

9.3.2 The `hocr` renderer

The `hocr` renderer works with older versions of Tesseract. The image layer is copied from the original PDF page if possible, avoiding potentially lossy transcoding or loss of other PDF information. If preprocessing is specified, then the image layer is a new PDF.

Unlike `sandwich` this renderer is implemented within OCRmyPDF; anyone looking to customize how OCR is presented should look here. A major disadvantage of this renderer is it not capable of correctly handling text outside the Latin alphabet (specifically, it supports the ISO 8859-1 character). Pull requests to improve the situation are welcome.

Currently, this renderer has the best compatibility with Mozilla's PDF.js viewer.

This works in all versions of Tesseract.

9.3.3 The `tesseract` renderer

The `tesseract` renderer was removed. OCRmyPDF's new approach to text layer grafting makes it functionally equivalent to `sandwich`.

9.4 Return code policy

OCRmyPDF writes all messages to `stderr`. `stdout` is reserved for piping output files. `stdin` is reserved for piping input files.

The return codes generated by the OCRmyPDF are considered part of the stable user interface. They may be imported from `ocrmypdf.exceptions`.

Table 1: Return codes

Code	Name	Interpretation
0	`ExitCode.ok`	Everything worked as expected.
1	`ExitCode.bad_args`	Invalid arguments, exited with an error.
2	`ExitCode.input_file`	The input file does not seem to be a valid PDF.
3	`ExitCode.missing_dependency`	An external program required by OCRmyPDF is missing.
4	`ExitCode.invalid_output_pdf`	An output file was created, but it does not seem to be a valid PDF. The file will be available.
5	`ExitCode.file_access_error`	The user running OCRmyPDF does not have sufficient permissions to read the input file and write the output file.
6	`ExitCode.already_done_ocr`	The file already appears to contain text so it may not need OCR. See output message.
7	`ExitCode.child_process_error`	An error occurred in an external program (child process) and OCRmyPDF cannot continue.
8	`ExitCode.encrypted_pdf`	The input PDF is encrypted. OCRmyPDF does not read encrypted PDFs. Use another program such as `qpdf` to remove encryption.
9	`ExitCode.invalid_config`	A custom configuration file was forwarded to Tesseract using `--tesseract-config`, and Tesseract rejected this file.
10	`ExitCode.pdfa_conversion_failed`	A valid PDF was created, PDF/A conversion failed. The file will be available.
15	`ExitCode.other_error`	Some other error occurred.
130	`ExitCode.ctrl_c`	The program was interrupted by pressing Ctrl+C.

9.5 Debugging the intermediate files

OCRmyPDF normally saves its intermediate results to a temporary folder and deletes this folder when it exits, whether it succeeded or failed.

If the `-k` argument is issued on the command line, OCRmyPDF will keep the temporary folder and print the location, whether it succeeded or failed (provided the Python interpreter did not crash). An example message is:

```
Temporary working files retained at:
/tmp/ocrmypdf.io.u20wpz07
```

The organization of this folder is an implementation detail and subject to change between releases. However the general organization is that working files on a per page basis have the page number as a prefix (starting with page 1), an infix indicates the processing stage, and a suffix indicates the file type. Some important files include:

- `_rasterize.png` - what the input page looks like
- `_ocr.png` - the file that is sent to Tesseract for OCR; depending on arguments this may differ from the presentation image
- `_pp_deskew.png` - the image, after deskewing
- `_pp_clean.png` - the image, after cleaning with unpaper
- `_ocr_tess.pdf` - the OCR file; appears as a blank page with invisible text embedded
- `_ocr_tess.txt` - the OCR text (not necessarily all text on the page, if the page is mixed format)
- `fix_docinfo.pdf` - a temporary file created to fix the PDF DocumentInfo data structure
- `graft_layers.pdf` - the rendered PDF with OCR layers grafted on
- `pdfa.pdf` - `graft_layers.pdf` after conversion to PDF/A

- `pdfa.ps` - a PostScript file used by Ghostscript for PDF/A conversion
- `optimize.pdf` - the PDF generated before optimization
- `optimize.out.pdf` - the PDF generated by optimization
- `origin` - the input file
- `origin.pdf` - the input file or the input image converted to PDF
- `images/*` - images extracted during the optimization process; here the prefix indicates a PDF object ID not a page number

CHAPTER

TEN

BATCH PROCESSING

This article provides information about running OCRmyPDF on multiple files or configuring it as a service triggered by file system events.

10.1 Batch jobs

Consider using the excellent GNU Parallel to apply OCRmyPDF to multiple files at once.

Both `parallel` and `ocrmypdf` will try to use all available processors. To maximize parallelism without overloading your system with processes, consider using `parallel -j 2` to limit parallel to running two jobs at once.

This command will run all ocrmypdf all files named `*.pdf` in the current directory and write them to the previous created `output/` folder. It will not search subdirectories.

The `--tag` argument tells parallel to print the filename as a prefix whenever a message is printed, so that one can trace any errors to the file that produced them.

```
parallel --tag -j 2 ocrmypdf '{}' 'output/{}' ::: *.pdf
```

OCRmyPDF automatically repairs PDFs before parsing and gathering information from them.

10.2 Directory trees

This will walk through a directory tree and run OCR on all files in place, printing the output in a way that makes

```
find . -printf '%p' -name '*.pdf' -exec ocrmypdf '{}' '{}' \;
```

Alternatively, with a docker container (mounts a volume to the container where the PDFs are stored):

```
find . -printf '%p' -name '*.pdf' -exec docker run --rm -v <host dir>:<container dir>␣
↪jbarlow83/ocrmypdf '<container dir>/{}' '<container dir>/{}' \;
```

This only runs one `ocrmypdf` process at a time. This variation uses `find` to create a directory list and `parallel` to parallelize runs of `ocrmypdf`, again updating files in place.

```
find . -name '*.pdf' | parallel --tag -j 2 ocrmypdf '{}' '{}'
```

In a Windows batch file, use

```
for /r %%f in (*.pdf) do ocrmypdf %%f %%f
```

10.2.1 Sample script

This user contributed script also provides an example of batch processing.

Listing 1: misc/batch.py

```
#!/usr/bin/env python3
# Copyright 2016 findingorder: https://github.com/findingorder
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

# This script must be edited to meet your needs.

import logging
import os
import sys

import ocrmypdf

# pylint: disable=logging-format-interpolation
# pylint: disable=logging-not-lazy

script_dir = os.path.dirname(os.path.realpath(__file__))
print(script_dir + '/batch.py: Start')

if len(sys.argv) > 1:
    start_dir = sys.argv[1]
else:
    start_dir = '.'

if len(sys.argv) > 2:
    log_file = sys.argv[2]
else:
    log_file = script_dir + '/ocr-tree.log'

logging.basicConfig(
    level=logging.INFO,
```

(continues on next page)

(continued from previous page)

```
    format='%(asctime)s %(message)s',
    filename=log_file,
    filemode='w',
)

ocrmypdf.configure_logging(ocrmypdf.Verbosity.default)

for dir_name, _subdirs, file_list in os.walk(start_dir):
    logging.info(dir_name + '\n')
    os.chdir(dir_name)
    for filename in file_list:
        file_ext = os.path.splitext(filename)[1]
        if file_ext == '.pdf':
            full_path = dir_name + '/' + filename
            print(full_path)
            result = ocrmypdf.ocr(filename, filename, deskew=True)
            if result == ocrmypdf.ExitCode.already_done_ocr:
                print("Skipped document because it already contained text")
            elif result == ocrmypdf.ExitCode.ok:
                print("OCR complete")
            logging.info(result)
```

10.2.2 Synology DiskStations

Synology DiskStations (Network Attached Storage devices) can run the Docker image of OCRmyPDF if the Synology Docker package is installed. Attached is a script to address particular quirks of using OCRmyPDF on one of these devices.

This is only possible for x86-based Synology products. Some Synology products use ARM or Power processors and do not support Docker. Further adjustments might be needed to deal with the Synology's relatively limited CPU and RAM.

Listing 2: misc/synology.py - Sample script for Synology DiskStations

```
#!/bin/env python3
# Copyright 2017 github.com/Enantiomerie
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
```

(continues on next page)

(continued from previous page)

```
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

# This script must be edited to meet your needs.

import logging
import os
import shutil
import subprocess
import sys
import time

# pylint: disable=logging-format-interpolation
# pylint: disable=logging-not-lazy

script_dir = os.path.dirname(os.path.realpath(__file__))
timestamp = time.strftime("%Y-%m-%d-%H%M_")
log_file = script_dir + '/' + timestamp + 'ocrmypdf.log'
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s %(message)s',
    filename=log_file,
    filemode='w',
)

if len(sys.argv) > 1:
    start_dir = sys.argv[1]
else:
    start_dir = '.'

for dir_name, _subdirs, file_list in os.walk(start_dir):
    logging.info(dir_name)
    os.chdir(dir_name)
    for filename in file_list:
        file_stem, file_ext = os.path.splitext(filename)
        if file_ext != '.pdf':
            continue
        full_path = os.path.join(dir_name, filename)
        timestamp_ocr = time.strftime("%Y-%m-%d-%H%M_OCR_")
        filename_ocr = timestamp_ocr + file_stem + '.pdf'
        # create string for pdf processing
        # the script is processed as root user via chron
        cmd = [
            'docker',
            'run',
            '--rm',
            '-i',
            'jbarlow83/ocrmypdf',
            '--deskew',
            '-',
            '-',
```

(continues on next page)

(continued from previous page)

```
        ]
        logging.info(cmd)
        full_path_ocr = os.path.join(dir_name, filename_ocr)
        with open(filename, 'rb') as input_file, open(
            full_path_ocr, 'wb'
        ) as output_file:
            proc = subprocess.run(
                cmd,
                stdin=input_file,
                stdout=output_file,
                stderr=subprocess.PIPE,
                check=False,
                text=True,
                errors='ignore',
            )
        logging.info(proc.stderr)
        os.chmod(full_path_ocr, 0o664)
        os.chmod(full_path, 0o664)
        full_path_ocr_archive = sys.argv[2]
        full_path_archive = sys.argv[2] + '/no_ocr'
        shutil.move(full_path_ocr, full_path_ocr_archive)
        shutil.move(full_path, full_path_archive)
logging.info('Finished.\n')
```

10.2.3 Huge batch jobs

If you have thousands of files to work with, contact the author. Consulting work related to OCRmyPDF helps fund this open source project and all inquiries are appreciated.

10.3 Hot (watched) folders

10.3.1 Watched folders with watcher.py

OCRmyPDF has a folder watcher called watcher.py, which is currently included in source distributions but not part of the main program. It may be used natively or may run in a Docker container. Native instances tend to give better performance. watcher.py works on all platforms.

Users may need to customize the script to meet their requirements.

```
pip3 install ocrmypdf[watcher]

env OCR_INPUT_DIRECTORY=/mnt/input-pdfs \
    OCR_OUTPUT_DIRECTORY=/mnt/output-pdfs \
    OCR_OUTPUT_DIRECTORY_YEAR_MONTH=1 \
    python3 watcher.py
```

Table 1: watcher.py environment variables

Environment variable	Description
OCR_INPUT_DIRECTORY	Set input directory to monitor (recursive)
OCR_OUTPUT_DIRECTORY	Set output directory (should not be under input)
OCR_ON_SUCCESS_DELETE	This will delete the input file if the exit code is 0 (OK)
OCR_OUTPUT_DIRECTORY_YEAR_MONTH	This will place files in the output in `{output}/{year}/{month}/{filename}`
OCR_DESKEW	Apply deskew to crooked input PDFs
OCR_JSON_SETTINGS	A JSON string specifying any other arguments for `ocrmypdf.ocr`, e.g. `'OCR_JSON_SETTINGS={"rotate_pages": true}'`.
OCR_POLL_NEW_FILE_SECONDS	Polling interval
OCR_LOGLEVEL	Level of log messages to report

One could configure a networked scanner or scanning computer to drop files in the watched folder.

10.3.2 Watched folders with Docker

The watcher service is included in the OCRmyPDF Docker image. To run it:

```
docker run \
    -v <path to files to convert>:/input \
    -v <path to store results>:/output \
    -e OCR_OUTPUT_DIRECTORY_YEAR_MONTH=1 \
    -e OCR_ON_SUCCESS_DELETE=1 \
    -e OCR_DESKEW=1 \
    -e PYTHONUNBUFFERED=1 \
    -it --entrypoint python3 \
    jbarlow83/ocrmypdf \
    watcher.py
```

This service will watch for a file that matches `/input/\*.pdf` and will convert it to a OCRed PDF in `/output/`. The parameters to this image are:

Table 2: watcher.py parameters for Docker

Parameter	Description
`-v <path to files to convert>:/input`	Files placed in this location will be OCRed
`-v <path to store results>:/output`	This is where OCRed files will be stored
`-e OCR_OUTPUT_DIRECTORY_YEAR_MONTH=1`	Define environment variable OCR_OUTPUT_DIRECTORY_YEAR_MONTH=1
`-e OCR_ON_SUCCESS_DELETE=1`	Define environment variable
`-e OCR_DESKEW=1`	Define environment variable
`-e PYTHONBUFFERED=1`	This will force STDOUT to be unbuffered and allow you to see messages in docker logs

This service relies on polling to check for changes to the filesystem. It may not be suitable for some environments, such as filesystems shared on a slow network.

A configuration manager such as Docker Compose could be used to ensure that the service is always available.

Listing 3: misc/docker-compose.example.yml

```
---
version: "3.3"
services:
  ocrmypdf:
    restart: always
    container_name: ocrmypdf
    image: jbarlow83/ocrmypdf
    volumes:
      - "/media/scan:/input"
      - "/mnt/scan:/output"
    environment:
      - OCR_OUTPUT_DIRECTORY_YEAR_MONTH=0
    user: "<SET TO YOUR USER ID>:<SET TO YOUR GROUP ID>"
    entrypoint: python3
    command: watcher.py
```

10.3.3 Caveats

- `watchmedo` may not work properly on a networked file system, depending on the capabilities of the file system client and server.
- This simple recipe does not filter for the type of file system event, so file copies, deletes and moves, and directory operations, will all be sent to ocrmypdf, producing errors in several cases. Disable your watched folder if you are doing anything other than copying files to it.
- If the source and destination directory are the same, watchmedo may create an infinite loop.
- On BSD, FreeBSD and older versions of macOS, you may need to increase the number of file descriptors to monitor more files, using `ulimit -n 1024` to watch a folder of up to 1024 files.

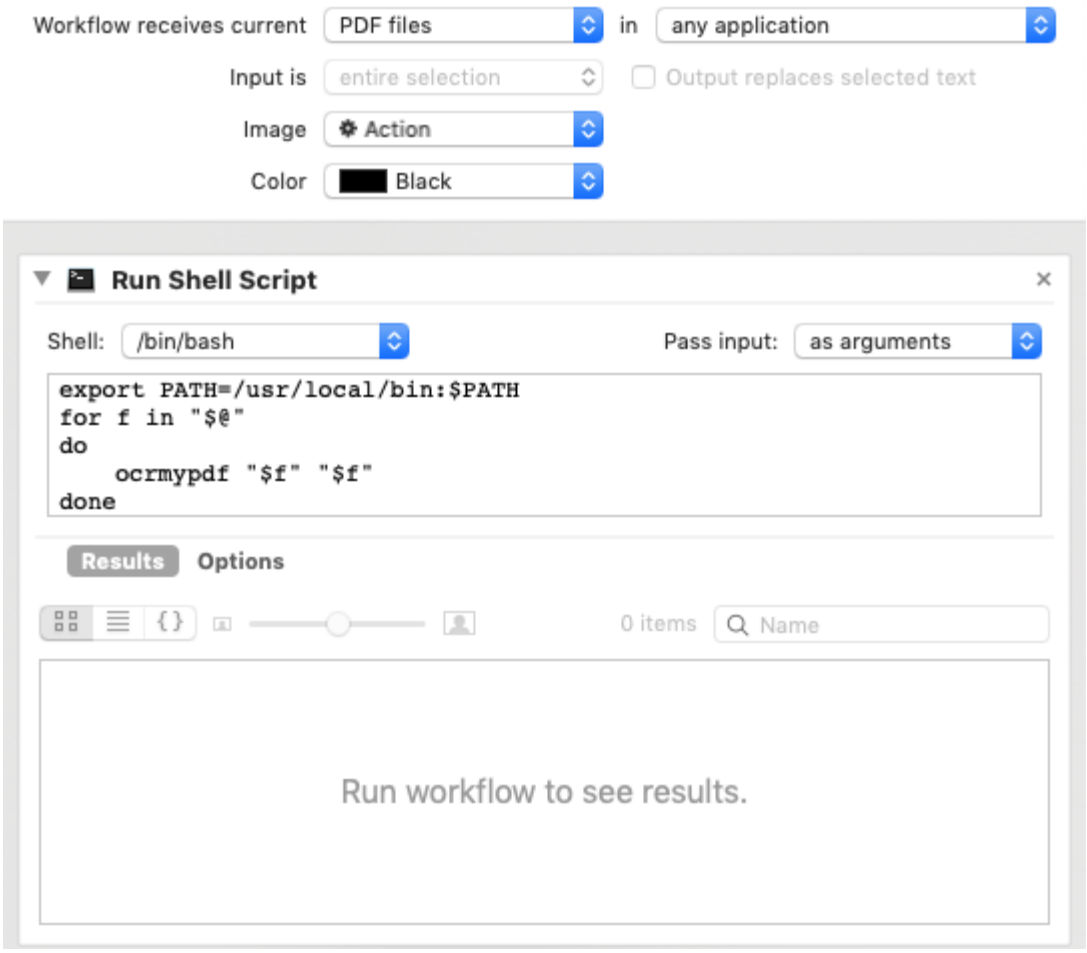
10.3.4 Alternatives

- On Linux, systemd user services can be configured to automatically perform OCR on a collection of files.
- Watchman is a more powerful alternative to `watchmedo`.

10.4 macOS Automator

You can use the Automator app with macOS, to create a Workflow or Quick Action. Use a *Run Shell Script* action in your workflow. In the context of Automator, the `PATH` may be set differently your Terminal's `PATH`; you may need to explicitly set the PATH to include `ocrmypdf`. The following example may serve as a starting point:

You may customize the command sent to ocrmypdf.


Workflow receives current
PDF files
in
any application
Input is
entire selection
Output replaces selected text
Image
Action
Color
Black
Run Shell Script
Shell:
/bin/bash
Pass input:
as arguments
export PATH=/usr/local/bin:$PATH
for f in "$@"
do
ocrmypdf "$f" "$f"
done
Results
Options
0 items
Name
Run workflow to see results.

CHAPTER

ELEVEN

PERFORMANCE

Some users have noticed that current versions of OCRmyPDF do not run as quickly as some older versions (specifically 6.x and older). This is because OCRmyPDF added image optimization as a postprocessing step, and it is enabled by default.

11.1 Speed

If running OCRmyPDF quickly is your main goal, you can use settings such as:

- `--optimize 0` to disable file size optimization
- `--output-type pdf` to disable PDF/A generation
- `--fast-web-view 0` to disable fast web view optimization
- `--skip-big` to skip large images, if some pages have large images

You can also avoid:

- `--force-ocr`
- Image preprocessing

CHAPTER

TWELVE

PDF SECURITY ISSUES

OCRmyPDF should only be used on PDFs you trust. It is not designed to protect you against malware.

Recognizing that many users have an interest in handling PDFs and applying OCR to PDFs they did not generate themselves, this article discusses the security implications of PDFs and how users can protect themselves.

The disclaimer applies: this software has no warranties of any kind.

12.1 PDFs may contain malware

PDF is a rich, complex file format. The official PDF 1.7 specification, ISO 32000:2008, is hundreds of pages long and references several annexes each of which are similar in length. PDFs can contain video, audio, XML, JavaScript and other programming, and forms. In some cases, they can open internet connections to pre-selected URLs. All of these are possible attack vectors.

In short, PDFs may contain viruses.

This article describes a high-paranoia method which allows potentially hostile PDFs to be viewed and rasterized safely in a disposable virtual machine. A trusted PDF created in this manner is converted to images and loses all information making it searchable and losing all compression. OCRmyPDF could be used to restore searchability.

12.2 How OCRmyPDF processes PDFs

OCRmyPDF must open and interpret your PDF in order to insert an OCR layer. First, it runs all PDFs through pikepdf, a library based on qpdf, a program that repairs PDFs with syntax errors. This is done because, in the author's experience, a significant number of PDFs in the wild, especially those created by scanners, are not well-formed files. qpdf makes it more likely that OCRmyPDF will succeed, but offers no security guarantees. qpdf is also used to split the PDF into single page PDFs.

Finally, OCRmyPDF rasterizes each page of the PDF using Ghostscript in `-dSAFER` mode.

Depending on the options specified, OCRmyPDF may graft the OCR layer into the existing PDF or it may essentially reconstruct ("re-fry") a visually identical PDF that may be quite different at the binary level. That said, OCRmyPDF is not a tool designed for sanitizing PDFs.

12.3 Using OCRmyPDF online or as a service

OCRmyPDF is not designed for use as a public web service where a malicious user could upload a chosen PDF. In particular, it is not necessarily secure against PDF malware or PDFs that cause denial of service. OCRmyPDF relies on Ghostscript, and therefore, if deployed online one should be prepared to comply with Ghostscript's Affero GPL license, and any other licenses.

Setting aside these concerns, a side effect of OCRmyPDF is that it may incidentally sanitize PDFs containing certain types of malware. It repairs the PDF with pikepdf/libqpdf, which could correct malformed PDF structures that are part of an attack. When PDF/A output is selected (the default), the input PDF is partially reconstructed by Ghostscript. When `--force-ocr` is used, all pages are rasterized and reconverted to PDF, which could remove malware in embedded images.

OCRmyPDF should be relatively safe to use in a trusted intranet, with some considerations:

12.3.1 Limiting CPU usage

OCRmyPDF will attempt to use all available CPUs and storage, so executing `nice ocrmypdf` or limiting the number of jobs with the `-j` argument may ensure the server remains available. Another option would be to run OCRmyPDF jobs inside a Docker container, a virtual machine, or a cloud instance, which can impose its own limits on CPU usage and be terminated "from orbit" if it fails to complete.

12.3.2 Temporary storage requirements

OCRmyPDF will use a large amount of temporary storage for its work, proportional to the total number of pixels needed to rasterize the PDF. The raster image of a 8.5×11" color page at 300 DPI takes 25 MB uncompressed; OCRmyPDF saves its intermediates as PNG, but that still means it requires about 9 MB per intermediate based on average compression ratios. Multiple intermediates per page are also required, depending on the command line given. A rule of thumb would be to allow 100 MB of temporary storage per page in a file – meaning that a small cloud servers or small VM partitions should be provisioned with plenty of extra space, if say, a 500 page file might be sent.

To check temporary storage usage on actual files, run `ocrmypdf -k ...` which will preserve and print the path to temporary storage when the job is done.

To change where temporary files are stored, change the `TMPDIR` environment variable for ocrmypdf's environment. (Python's `tempfile.gettempdir()` returns the root directory in which temporary files will be stored.) For example, one could redirect `TMPDIR` to a large RAM disk to avoid wear on HDD/SSD and potentially improve performance. On Amazon Web Services, `TMPDIR` can be set to empheral storage.

12.3.3 Timeouts

To prevent excessively long OCR jobs consider setting `--tesseract-timeout` and/or `--skip-big` arguments. `--skip-big` is particularly helpful if your PDFs include documents such as reports on standard page sizes with large images attached - often large images are not worth OCR'ing anyway.

12.3.4 Commercial alternatives

The author also provides professional services that include OCR and building databases around PDFs, and is happy to provide consultation.

Abbyy Cloud OCR is viable commercial alternative with a web services API. Amazon Textract, Google Cloud Vision, and Microsoft Azure Computer Vision provide advanced OCR but have less PDF rendering capability.

12.4 Password protection, digital signatures and certification

Password protected PDFs usually have two passwords, and owner and user password. When the user password is set to empty, PDF readers will open the file automatically and marked it as "(SECURED)". While not as reliable as a digital signature, this indicates that whoever set the password approved of the file at that time. When the user password is set, the document cannot be viewed without the password.

Either way, OCRmyPDF does not remove passwords from PDFs and exits with an error on encountering them.

`qpdf` can remove passwords. If the owner and user password are set, a password is required for `qpdf`. If only the owner password is set, then the password can be stripped, even if one does not have the owner password.

After OCR is applied, password protection is not permitted on PDF/A documents but the file can be converted to regular PDF.

Many programs exist which are capable of inserting an image of someone's signature. On its own, this offers no security guarantees. It is trivial to remove the signature image and apply it to other files. This practice offers no real security.

Important documents can be digitally signed and certified to attest to their authorship. OCRmyPDF cannot do this. Open source tools such as pdfbox (Java) have this capability as does Adobe Acrobat.

CHAPTER

THIRTEEN

COMMON ERROR MESSAGES

13.1 Page already has text

```
ERROR -    1: page already has text! - aborting (use --force-ocr to force OCR)
```

You ran ocrmypdf on a file that already contains printable text or a hidden OCR text layer (it can't quite tell the difference). You probably don't want to do this, because the file is already searchable.

As the error message suggests, your options are:

- `ocrmypdf --force-ocr` to *rasterize* all vector content and run OCR on the images. This is useful if a previous OCR program failed, or if the document contains a text watermark.
- `ocrmypdf --skip-text` to skip OCR and other processing on any pages that contain text. Text pages will be copied into the output PDF without modification.
- `ocrmypdf --redo-ocr` to scan the file for any existing OCR (non-printing text), remove it, and do OCR again. This is one way to take advantage of improvements in OCR accuracy. Printable vector text is excluded from OCR, so this can be used on files that contain a mix of digital and scanned files.

13.2 Input file 'filename' is not a valid PDF

OCRmyPDF checks files with pikepdf, a library that in turn uses libqpdf to fixes errors in PDFs, before it tries to work on them. In most cases this happens because the PDF is corrupt and truncated (incomplete file copying) and not much can be done.

You can try rewriting the file with Ghostscript:

```
gs -o output.pdf -dSAFER -sDEVICE=pdfwrite input.pdf
```

`pdftk` can also rewrite PDFs:

```
pdftk input.pdf cat output output.pdf
```

Sometimes Acrobat can repair PDFs with its Preflight tool.

CHAPTER

FOURTEEN

USING THE OCRMYPDF API

OCRmyPDF originated as a command line program and continues to have this legacy, but parts of it can be imported and used in other Python applications.

Some applications may want to consider running ocrmypdf from a subprocess call anyway, as this provides isolation of its activities.

14.1 Example

OCRmyPDF provides one high-level function to run its main engine from an application. The parameters are symmetric to the command line arguments and largely have the same functions.

```
import ocrmypdf

if __name__ == '__main__':  # To ensure correct behavior on Windows and macOS
    ocrmypdf.ocr('input.pdf', 'output.pdf', deskew=True)
```

With some exceptions, all of the command line arguments are available and may be passed as equivalent keywords.

A few differences are that `verbose` and `quiet` are not available. Instead, output should be managed by configuring logging.

14.1.1 Parent process requirements

The *ocrmypdf.ocr()* function runs OCRmyPDF similar to command line execution. To do this, it will:

- create a monitoring thread
- create worker processes (on Linux, forking itself; on Windows and macOS, by spawning)
- manage the signal flags of its worker processes
- execute other subprocesses (forking and executing other programs)

The Python process that calls *ocrmypdf.ocr()* must be sufficiently privileged to perform these actions.

There currently is no option to manage how jobs are scheduled other than the argument `jobs=` which will limit the number of worker processes.

Creating a child process to call *ocrmypdf.ocr()* is suggested. That way your application will survive and remain interactive even if OCRmyPDF fails for any reason.

Programs that call *ocrmypdf.ocr()* should also install a SIGBUS signal handler (except on Windows), to raise an exception if access to a memory mapped file fails. OCRmyPDF may use memory mapping.

ocrmypdf.ocr() will take a threading lock to prevent multiple runs of itself in the same Python interpreter process. This is not thread-safe, because of how OCRmyPDF's plugins and Python's library import system work. If you need to parallelize OCRmyPDF, use processes.

Warning: On Windows and macOS, the script that calls *ocrmypdf.ocr()* must be protected by an "ifmain" guard (`if __name__ == '__main__'`). If you do not take at least one of these steps, process semantics will prevent OCRmyPDF from working correctly.

14.1.2 Logging

OCRmyPDF will log under loggers named `ocrmypdf`. In addition, it imports `pdfminer` and `PIL`, both of which post log messages under those logging namespaces.

You can configure the logging as desired for your application or call *ocrmypdf.configure_logging()* to configure logging the same way OCRmyPDF itself does. The command line parameters such as `--quiet` and `--verbose` have no equivalents in the API; you must use the provided configuration function or do configuration in a way that suits your use case.

14.1.3 Progress monitoring

OCRmyPDF uses the `tqdm` package to implement its progress bars. *ocrmypdf.configure_logging()* will set up logging output to `sys.stderr` in a way that is compatible with the display of the progress bar. Use `ocrmypdf.ocr(...progress_bar=False)` to disable the progress bar.

14.1.4 Exceptions

OCRmyPDF may throw standard Python exceptions, `ocrmypdf.exceptions.*` exceptions, some exceptions related to multiprocessing, and `KeyboardInterrupt`. The parent process should provide an exception handler. OCRmyPDF will clean up its temporary files and worker processes automatically when an exception occurs.

Programs that call OCRmyPDF should consider trapping KeyboardInterrupt so that they allow OCR to terminate with the whole program terminating.

When OCRmyPDF succeeds conditionally, it returns an integer exit code.

14.1.5 Reference

ocrmypdf.**ocr**(*input_file: Union[BinaryIO, pathlib.Path, AnyStr]*, *output_file: Union[BinaryIO, pathlib.Path, AnyStr]*, ***, *language: Optional[Iterable[str]] = None*, *image_dpi: Optional[int] = None*, *output_type=None*, *sidecar: Optional[Union[pathlib.Path, AnyStr]] = None*, *jobs: Optional[int] = None*, *use_threads: Optional[bool] = None*, *title: Optional[str] = None*, *author: Optional[str] = None*, *subject: Optional[str] = None*, *keywords: Optional[str] = None*, *rotate_pages: Optional[bool] = None*, *remove_background: Optional[bool] = None*, *deskew: Optional[bool] = None*, *clean: Optional[bool] = None*, *clean_final: Optional[bool] = None*, *unpaper_args: Optional[str] = None*, *oversample: Optional[int] = None*, *remove_vectors: Optional[bool] = None*, *threshold: Optional[bool] = None*, *force_ocr: Optional[bool] = None*, *skip_text: Optional[bool] = None*, *redo_ocr: Optional[bool] = None*, *skip_big: Optional[float] = None*, *optimize: Optional[int] = None*, *jpg_quality: Optional[int] = None*, *png_quality: Optional[int] = None*, *jbig2_lossy: Optional[bool] = None*, *jbig2_page_group_size: Optional[int] = None*, *pages: Optional[str] = None*, *max_image_mpixels: Optional[float] = None*, *tesseract_config: Optional[Iterable[str]] = None*, *tesseract_pagesegmode: Optional[int] = None*, *tesseract_oem: Optional[int] = None*, *pdf_renderer=None*, *tesseract_timeout: Optional[float] = None*, *rotate_pages_threshold: Optional[float] = None*, *pdfa_image_compression=None*, *user_words: Optional[os.PathLike] = None*, *user_patterns: Optional[os.PathLike] = None*, *fast_web_view: Optional[float] = None*, *plugins: Optional[Iterable[Union[pathlib.Path, AnyStr]]] = None*, *plugin_manager=None*, *keep_temporary_files: Optional[bool] = None*, *progress_bar: Optional[bool] = None*, ***kwargs*)

Run OCRmyPDF on one PDF or image.

For most arguments, see documentation for the equivalent command line parameter. A few specific arguments are discussed here:

Parameters

- **use_threads** – Use worker threads instead of processes. This reduces performance but may make debugging easier since it is easier to set breakpoints.
- **input_file** – If a `pathlib.Path`, `str` or `bytes`, this is interpreted as file system path to the input file. If the object appears to be a readable stream (with methods such as `.read()` and `.seek()`), the object will be read in its entirety and saved to a temporary file. If `input_file` is `"-"`, standard input will be read.
- **output_file** – If a `pathlib.Path`, `str` or `bytes`, this is interpreted as file system path to the output file. If the object appears to be a writable stream (with methods such as `.write()` and `.seek()`), the output will be written to this stream. If `output_file` is `"-"`, the output will be written to `sys.stdout` (provided that standard output does not seem to be a terminal device). When a stream is used as output, whether via a writable object or `"-"`, some final validation steps are not performed (we do not read back the stream after it is written).

Raises

- **ocrmypdf.PdfMergeFailedError** – If the input PDF is malformed, preventing merging with the OCR layer.
- **ocrmypdf.MissingDependencyError** – If a required dependency program is missing or was not found on PATH.
- **ocrmypdf.UnsupportedImageFormatError** – If the input file type was an image that could not be read, or some other file type that is not a PDF.
- **ocrmypdf.DpiError** – If the input file is an image, but the resolution of the image is not credible (allowing it to proceed would cause poor OCR).
- **ocrmypdf.OutputFileAccessError** – If an attempt to write to the intended output file failed.

- **ocrmypdf.PriorOcrFoundError** – If the input PDF seems to have OCR or digital text already, and settings did not tell us to proceed.
- **ocrmypdf.InputFileError** – Any other problem with the input file.
- **ocrmypdf.SubprocessOutputError** – Any error related to executing a subprocess.
- **ocrmypdf.EncryptedPdfERror** – If the input PDF is encrypted (password protected). OCRmyPDF does not remove passwords.
- **ocrmypdf.TesseractConfigError** – If Tesseract reported its configuration was not valid.

Returns ocrmypdf.ExitCode

class ocrmypdf.**Verbosity**(*value*)

Verbosity level for configure_logging.

debug = 1

Output ocrmypdf debug messages

debug_all = 2

More detailed debugging from ocrmypdf and dependent modules

default = 0

Default level of logging

quiet = -1

Suppress most messages

ocrmypdf.**configure_logging**(*verbosity:* ocrmypdf.api.Verbosity, *, *progress_bar_friendly: bool = True*, *manage_root_logger: bool = False*, *plugin_manager=None*)

Set up logging.

Before calling *ocrmypdf.ocr()*, you can use this function to configure logging if you want ocrmypdf's output to look like the ocrmypdf command line interface. It will register log handlers, log filters, and formatters, configure color logging to standard error, and adjust the log levels of third party libraries. Details of this are fine-tuned and subject to change. The `verbosity` argument is equivalent to the argument `--verbose` and applies those settings. If you have a wrapper script for ocrmypdf and you want it to be very similar to ocrmypdf, use this function; if you are using ocrmypdf as part of an application that manages its own logging, you probably do not want this function.

If this function is not called, ocrmypdf will not configure logging, and it is up to the caller of `ocrmypdf.ocr()` to set up logging as it wishes using the Python standard library's logging module. If this function is called, the caller may of course make further adjustments to logging.

Regardless of whether this function is called, ocrmypdf will perform all of its logging under the `"ocrmypdf"` logging namespace. In addition, ocrmypdf imports pdfminer, which logs under `"pdfminer"`. A library user may wish to configure both; note that pdfminer is extremely chatty at the log level `logging.INFO`.

This function does not set up the `debug.log` log file that the command line interface does at certain verbosity levels. Applications should configure their own debug logging.

Parameters

- **verbosity** – Verbosity level.
- **progress_bar_friendly** – If True (the default), install a custom log handler that is compatible with progress bars and colored output.
- **manage_root_logger** – Configure the process's root logger.
- **plugin_manager** – The plugin manager, used for obtaining the custom log handler.

Returns The toplevel logger for ocrmypdf (or the root logger, if we are managing it).

CHAPTER

FIFTEEN

PLUGINS

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

You can use plugins to customize the behavior of OCRmyPDF at certain points of interest.

Currently, it is possible to:

- add new command line arguments
- override the decision for whether or not to perform OCR on a particular file
- modify the image is about to be sent for OCR
- modify the page image before it is converted to PDF
- replace the Tesseract OCR with another OCR engine that has similar behavior
- replace Ghostscript with another PDF to image converter (rasterizer) or PDF/A generator

OCRmyPDF plugins are based on the Python `pluggy` package and conform to its conventions. Note that: plugins installed with as setuptools entrypoints are not checked currently, because OCRmyPDF assumes you may not want to enable plugins for all files.

15.1 Script plugins

Script plugins may be called from the command line, by specifying the name of a file. Script plugins may be convenient for informal or "one-off" plugins, when a certain batch of files needs a special processing step for example.

```
ocrmypdf --plugin ocrmypdf_example_plugin.py input.pdf output.pdf
```

Multiple plugins may be installed by issuing the `--plugin` argument multiple times.

15.2 Packaged plugins

Installed plugins may be installed into the same virtual environment as OCRmyPDF is installed into. They may be invoked using Python standard module naming. If you are intending to distribute a plugin, please package it.

```
ocrmypdf --plugin ocrmypdf_fancypants.pockets.contents input.pdf output.pdf
```

OCRmyPDF does not automatically import plugins, because the assumption is that plugins affect different files differently and you may not want them activated all the time. The command line or `ocrmypdf.ocr(plugin='...')` must call for them.

Third parties that wish to distribute packages for ocrmypdf should package them as packaged plugins, and these modules should begin with the name `ocrmypdf_` similar to `pytest` packages such as `pytest-cov` (the package) and `pytest_cov` (the module).

Note: We recommend plugin authors name their plugins with the prefix `ocrmypdf-` (for the package name on PyPI) and `ocrmypdf_` (for the module), just like pytest plugins. At the same time, please make it clear that your package is not official.

15.3 Setuptools plugins

You can also create a plugin that OCRmyPDF will always automatically load if both are installed in the same virtual environment, using a setuptools entrypoint.

Your package's `setup.py` would need to contain the following, for a plugin named `ocrmypdf-exampleplugin`:

```
# sample ./setup.py file
from setuptools import setup

setup(
    name="ocrmypdf-exampleplugin",
    packages=["exampleplugin"],
    # the following makes a plugin available to pytest
    entry_points={"ocrmypdf": ["exampleplugin = exampleplugin.pluginmodule"]},
)
```

```
# equivalent setup.cfg
[options.entry_points]
ocrmypdf =
    exampleplugin = exampleplugin.pluginmodule
```

15.4 Plugin requirements

OCRmyPDF generally uses multiple worker processes. When a new worker is started, Python will import all plugins again, including all plugins that were imported earlier. This means that the global state of a plugin in one worker will not be shared with other workers. As such, plugin hook implementations should be stateless, relying only on their inputs. Hook implementations may use their input parameters to to obtain a reference to shared state prepared by another hook implementation. Plugins must expect that other instances of the plugin will be running simultaneously.

The `context` object that is passed to many hooks can be used to share information about a file being worked on. Plugins must write private, plugin-specific data to a subfolder named `{options.work_folder}/ocrmypdf-plugin-name`. Plugins MAY read and write files in `options.work_folder`, but should be aware that their semantics are subject to change.

OCRmyPDF will delete `options.work_folder` when it has finished OCRing a file, unless invoked with `--keep-temporary-files`.

The documentation for some plugin hooks contain a detailed description of the execution context in which they will be called.

Plugins should be prepared to work whether executed in worker threads or worker processes. Generally, OCRmyPDF uses processes, but has a semi-hidden threaded argument that simplifies debugging.

15.5 Plugin hooks

A plugin may provide the following hooks. Hooks must be decorated with `ocrmypdf.hookimpl`, for example:

```
from ocrmpydf import hookimpl

@hookimpl
def add_options(parser):
    pass
```

The following is a complete list of hooks that are available, and when they are called.

Note on firstresult hooks

If multiple plugins install implementations for this hook, they will be called in the reverse of the order in which they are installed (i.e., last plugin wins). When each hook implementation is called in order, the first implementation that returns a value other than `None` will "win" and prevent execution of all other hooks. As such, you cannot "chain" a series of plugin filters together in this way. Instead, a single hook implementation should be responsible for any such chaining operations.

15.5.1 Custom command line arguments

`ocrmypdf.pluginspec.add_options`(*parser: argparse.ArgumentParser*) → None

Allows the plugin to add its own command line and API arguments.

OCRmyPDF converts command line arguments to API arguments, so adding arguments here will cause new arguments to be processed for API calls to `ocrmypdf.ocr`, or when invoked on the command line.

Note: This hook will be called from the main process, and may modify global state before child worker processes are forked.

`ocrmypdf.pluginspec.check_options`(*options: argparse.Namespace*) → None

Called to ask the plugin to check all of the options.

The plugin may check if options that it added are valid.

Warnings or other messages may be passed to the user by creating a logger object using `log = logging.getLogger(__name__)` and logging to this.

The plugin may also modify the *options*. All objects that are in options must be picklable so they can be marshalled to child worker processes.

Raises *ocrmypdf.exceptions.ExitCodeException* – If options are not acceptable and the application should terminate gracefully with an informative message and error code.

Note: This hook will be called from the main process, and may modify global state before child worker processes are forked.

15.5.2 Execution and progress reporting

ocrmypdf.pluginspec.**get_logging_console**() → logging.Handler
Returns a custom logging handler.

Generally this is necessary when both logging output and a progress bar are both outputting to `sys.stderr`.

Note: This is a *firstresult hook*.

---

ocrmypdf.pluginspec.**get_executor**(*progressbar_class*) → ocrmypdf._concurrent.Executor
Called to obtain an object that manages parallel execution.

This may be used to replace OCRmyPDF's default parallel execution system with a third party alternative. For example, you could make OCRmyPDF run in a distributed environment.

OCRmyPDF's executors are analogous to the standard Python executors in `conconcurrent.futures`, but they do not work the same way. Executors may be reused for different, unrelated batch operations, since all of the context for a given job are passed to `Executor.__call__()`.

Should be of type `Executor` or otherwise conforming to the protocol of that call.

Parameters **progressbar_class** – A progress bar class, which will be created when

---

Note: This hook will be called from the main process, and may modify global state before child worker processes are forked.

Note: This is a *firstresult hook*.

---

ocrmypdf.pluginspec.**get_progressbar_class**()
Called to obtain a class that can be used to monitor progress.

A progress bar is assumed, but this could be used for any type of monitoring.

The class should follow a tqdm-like protocol. Calling the class should return a new progress bar object, which is activated with `__enter__` and terminated `__exit__`. An update method is called whenever the progress bar is updated. Progress bar objects will not be reused; a new one will be created for each group of tasks.

The progress bar is held in the main process/thread and not updated by child process/threads. When a child notifies the parent of completed work, the parent updates the progress bar.

The arguments are the same as tqdm accepts.

Progress bars should never write to `sys.stdout`, or they will corrupt the output if OCRmyPDF writes a PDF to standard output.

The type of events that OCRmyPDF reports to a progress bar may change in minor releases.

Here is how OCRmyPDF will use the progress bar:

Example

pbar_class = pm.hook.get_progressbar_class() with pbar_class(**tqdm_kwargs) as pbar:

... pbar.update(1)

15.5.3 Applying special behavior before processing

ocrmypdf.pluginspec.**validate**(*pdfinfo: PdfInfo*, *options: argparse.Namespace*) → None

Called to give a plugin an opportunity to review *options* and *pdfinfo*.

options contains the "work order" to process a particular file. *pdfinfo* contains information about the input file obtained after loading and parsing. The plugin may modify the *options*. For example, you could decide that a certain type of file should be treated with `options.force_ocr = True` based on information in its *pdfinfo*.

Raises ***ocrmypdf.exceptions.ExitCodeException*** – If options or pdfinfo are not acceptable and the application should terminate gracefully with an informative message and error code.

Note: This hook will be called from the main process, and may modify global state before child worker processes are forked.

15.5.4 PDF page to image

ocrmypdf.pluginspec.**rasterize_pdf_page**(*input_file: pathlib.Path*, *output_file: pathlib.Path*, *raster_device: str*, *raster_dpi: ocrmypdf.helpers.Resolution*, *pageno: int*, *page_dpi: Optional[ocrmypdf.helpers.Resolution]*, *rotation: Optional[int]*, *filter_vector: bool*) → pathlib.Path

Rasterize one page of a PDF at resolution raster_dpi in canvas units.

The image is sized to match the integer pixels dimensions implied by raster_dpi even if those numbers are non-integer. The image's DPI will be overridden with the values in page_dpi.

Parameters

- **input_file** – The PDF to rasterize.
- **output_file** – The desired name of the rasterized image.
- **raster_device** – Type of image to produce at output_file
- **raster_dpi** – Resolution at which to rasterize page
- **pageno** – Page number to rasterize (beginning at page 1)
- **page_dpi** – Resolution, overriding output image DPI
- **rotation** – Cardinal angle, clockwise, to rotate page
- **filter_vector** – If True, remove vector graphics objects

Returns *Path* – output_file if successful

Note: This hook will be called from child processes. Modifying global state will not affect the main process or other child processes.

Note: This is a *firstresult hook*.

15.5.5 Modifying intermediate images

ocrmypdf.pluginspec.**filter_ocr_image**(*page: PageContext*, *image: Image.Image*) → Image.Image
Called to filter the image before it is sent to OCR.

This is the image that OCR sees, not what the user sees when they view the PDF. If `redo_ocr` is enabled, portions of the image will be masked so they are not shown to OCR. The main use of this hook is expected to be hiding content from OCR.

The input image may be color, grayscale, or monochrome, and the output image may differ. The pixel width and height of the output image must be identical to the input image, or misalignment between the OCR text layer and visual position of the text will occur. Likewise, the output must be a faithful representation of the input, or alignment errors may occurs.

Tesseract OCR only deals with monochrome images, and internally converts non-monochrome images to OCR.

Note: This hook will be called from child processes. Modifying global state will not affect the main process or other child processes.

Note: This is a *firstresult hook*.

ocrmypdf.pluginspec.**filter_page_image**(*page: PageContext*, *image_filename: pathlib.Path*) → pathlib.Path
Called to filter the whole page before it is inserted into the PDF.

A whole page image is only produced when preprocessing command line arguments are issued or when `--force-ocr` is issued. If no whole page is image is produced for a given page, this function will not be called. This is not the image that will be shown to OCR.

If the function does not want to modify the image, it should return `image_filename`. The hook may overwrite `image_filename` with a new file.

The output image should preserve the same physical unit dimensions, that is (width * dpi_x, height * dpi_y). That is, if the image is resized, the DPI must be adjusted by the reciprocal. If this is not preserved, the PDF page will be resized and the OCR layer misaligned. OCRmyPDF does not nothing to enforce these constraints; it is up to the plugin to do sensible things.

OCRmyPDF will create the PDF page based on the image format used (unless the hook is overriden). If you convert the image to a JPEG, the output page will be created as a JPEG, etc. If you change the colorspace, that change will be kept. Note that the OCRmyPDF image optimization stage, if enabled, may ultimately chose a different format.

If the return value is a file that does not exist, `FileNotFoundError` will occur. The return value should be a path to a file in the same folder as `image_filename`.

Implementation detail: If the value returned is falsy, OCRmyPDF will ignore the return value and assume the input file was unmodified. This is deprecated. To leave the image unmodified, `image_filename` should be returned.

Note: This hook will be called from child processes. Modifying global state will not affect the main process or

other child processes.

Note: This is a *firstresult hook*.

ocrmypdf.pluginspec.**filter_pdf_page**(*page: PageContext*, *image_filename: pathlib.Path*, *output_pdf: pathlib.Path*) → pathlib.Path

Called to convert a filtered whole page image into a PDF.

A whole page image is only produced when preprocessing command line arguments are issued or when `--force-ocr` is issued. If no whole page is image is produced for a given page, this function will not be called. This is not the image that will be shown to OCR. The whole page image is filtered in the hook above, `filter_page_image`, then this function is called for PDF conversion.

This function will only be called when OCRmyPDF runs in a mode such as "force OCR" mode where rasterizing of all content is performed.

Clever things could be done at this stage such as segmenting the page image into color regions or vector equivalents.

The provider of the hook implementation is responsible for ensuring that the OCR text layer is aligned with the PDF produced here, or text misalignment will result.

Currently this function must produce a single page PDF or the pipeline will fail. If the intent is to remove the PDF, then create a single page empty PDF.

Parameters

- **page** – Context for this page.
- **image_filename** – Filename of the input image used to create output_pdf, for "reference" if recreating the output_pdf entirely.
- **output_pdf** – The previous created output_pdf.

Returns output_pdf

Note: This hook will be called from child processes. Modifying global state will not affect the main process or other child processes.

Note: This is a *firstresult hook*.

15.5.6 OCR engine

ocrmypdf.pluginspec.**get_ocr_engine**() → *ocrmypdf.pluginspec.OcrEngine*

Returns an OcrEngine to use for processing this file.

The OcrEngine may be instantiated multiple times, by both the main process and child process. As such, it must be obtain store any state in `options` or some common location.

Note: This is a *firstresult hook*.

class ocrmypdf.pluginspec.**OcrEngine**

A class representing an OCR engine with capabilities similar to Tesseract OCR.

This could be used to create a plugin for another OCR engine instead of Tesseract OCR.

abstract __str__()

Returns name of OCR engine and version.

This is used when OCRmyPDF wants to mention the name of the OCR engine to the user, usually in an error message.

abstract static creator_tag(*options: argparse.Namespace*) → str

Returns the creator tag to identify this software's role in creating the PDF.

This tag will be inserted in the XMP metadata and DocumentInfo dictionary as appropriate. Ideally you should include the name of the OCR engine and its version. The text should not contain line breaks. This is to help developers like yourself identify the software that produced this file.

OCRmyPDF will always prepend its name to this value.

abstract static generate_hocr(*input_file: pathlib.Path*, *output_hocr: pathlib.Path*, *output_text: pathlib.Path*, *options: argparse.Namespace*) → None

Called to produce a hOCR file and sidecar text file.

abstract static generate_pdf(*input_file: pathlib.Path*, *output_pdf: pathlib.Path*, *output_text: pathlib.Path*, *options: argparse.Namespace*) → None

Called to produce a text only PDF.

Parameters

- **input_file** – A page image on which to perform OCR.
- **output_pdf** – The expected name of the output PDF, which must be a single page PDF with no visible content of any kind, sized to the dimensions implied by the input_file's width, height and DPI. The image will be grafted onto the input PDF page.

abstract static get_orientation(*input_file: pathlib.Path*, *options: argparse.Namespace*) → *ocrmypdf.pluginspec.OrientationConfidence*

Returns the orientation of the image.

abstract static languages(*options: argparse.Namespace*) → AbstractSet[str]

Returns the set of all languages that are supported by the engine.

Languages are typically given in 3-letter ISO 3166-1 codes, but actually can be any value understood by the OCR engine.

abstract static version() → str

Returns the version of the OCR engine.

class ocrmypdf.pluginspec.**OrientationConfidence**(*angle*, *confidence*)

Expresses an OCR engine's confidence in page rotation.

angle

The clockwise angle (0, 90, 180, 270) that the page should be rotated. 0 means no rotation.

Type int

confidence

How confident the OCR engine is that this the correct rotation. 0 is not confident, 15 is very confident. Arbitrary units.

Type float

15.5.7 PDF/A production

ocrmypdf.pluginspec.**generate_pdfa**(*pdf_pages: List[pathlib.Path]*, *pdfmark: pathlib.Path*, *output_file: pathlib.Path*, *compression: str*, *pdf_version: str*, *pdfa_part: str*, *progressbar_class*) → pathlib.Path

Generate a PDF/A.

This API strongly assumes a PDF/A generator with Ghostscript's semantics.

OCRmyPDF will modify the metadata and possibly linearize the PDF/A after it is generated.

Parameters

- **pdf_pages** – A list of one or more filenames, will be merged into output_file.
- **pdfmark** – A PostScript file intended for Ghostscript with details on how to perform the PDF/A conversion.
- **output_file** – The name of the desired output file.
- **compression** – One of `'jpeg'`, `'lossless'`, `''`. For `'jpeg'`, the PDF/A generator should convert all images to JPEG encoding where possible. For lossless, all images should be converted to FlateEncode (lossless PNG). If an empty string, the PDF generator should make its own decisions about how to encode images.
- **pdf_version** – The minimum PDF version that the output file should be. At its own discretion, the PDF/A generator may raise the version, but should not lower it.
- **pdfa_part** – The desired PDF/A compliance level, such as `'2B'`.
- **progressbar_class** – The class of a progress bar with a tqdm-like API. An instance of this class will be initialized when PDF/A conversion begins, using `instance = progressbar_class(total: int, desc: str, unit:str)`, defining the number of work units, a user-visible description, and the name of the work units ("page"). Then `instance.update()` will be called when a work unit is completed. If `None`, no progress information is reported.

Returns *Path* – If successful, the hook should return `output_file`.

Note: This is a *firstresult hook*.

See also:

https://github.com/tqdm/tqdm

CHAPTER

SIXTEEN

API REFERENCE

This page summarizes the rest of the public API. Generally speaking this should mainly of interest to plugin developers.

16.1 ocrmypdf

class ocrmypdf.**PageContext**(*pdf_context:* ocrmypdf._jobcontext.PdfContext, *pageno*)

Holds our context for a page.

Must be pickable, so stores only intrinsic/simple data elements or those capable of their serializing themselves via `__getstate__`.

get_path(*name: str*) → pathlib.Path

Generate a `Path` for a file that is part of processing this page.

The path will be based in a common temporary folder and have a prefix based on the page number.

options: argparse.Namespace

The specified options for processing this PDF.

origin: pathlib.Path

The filename of the original input file.

pageinfo: ocrmypdf.pdfinfo.info.PageInfo

Information on this page.

pageno: int

This page number (zero-based).

plugin_manager: pluggy._manager.PluginManager

PluginManager for processing the current PDF.

class ocrmypdf.**PdfContext**(*options: argparse.Namespace*, *work_folder: pathlib.Path*, *origin: pathlib.Path*, *pdfinfo: ocrmypdf.pdfinfo.info.PdfInfo*, *plugin_manager*)

Holds the context for a particular run of the pipeline.

get_page_contexts() → Iterator[*ocrmypdf._jobcontext.PageContext*]

Get all `PageContext` for this PDF.

get_path(*name: str*) → pathlib.Path

Generate a `Path` for an intermediate file involved in processing.

The path will be in a temporary folder that is common for all processing of this particular PDF.

options: argparse.Namespace

The specified options for processing this PDF.

origin: pathlib.Path
The filename of the original input file.

pdfinfo: ocrmypdf.pdfinfo.info.PdfInfo
Detailed data for this PDF.

plugin_manager: pluggy._manager.PluginManager
PluginManager for processing the current PDF.

16.2 ocrmypdf.exceptions

exception ocrmypdf.exceptions.**BadArgsError**

exit_code = 1

exception ocrmypdf.exceptions.**DpiError**

exit_code = 2

exception ocrmypdf.exceptions.**EncryptedPdfError**

exit_code = 8

message = "Input PDF is encrypted. The encryption must be removed to\nperform OCR.\n\nFor information about this PDF's security use\n qpdf --show-encryption infilename\n\nYou can remove the encryption using\n qpdf --decrypt [--password=[password]] infilename\n"

class ocrmypdf.exceptions.**ExitCode**(*value*)
An enumeration.

already_done_ocr = 6

bad_args = 1

child_process_error = 7

ctrl_c = 130

encrypted_pdf = 8

file_access_error = 5

input_file = 2

invalid_config = 9

invalid_output_pdf = 4

missing_dependency = 3

ok = 0

other_error = 15

pdfa_conversion_failed = 10

exception ocrmypdf.exceptions.**ExitCodeException**

exit_code = 15

message = ''

exception ocrmypdf.exceptions.**InputFileError**

exit_code = 2

exception ocrmypdf.exceptions.**MissingDependencyError**

exit_code = 3

exception ocrmypdf.exceptions.**OutputFileAccessError**

exit_code = 5

exception ocrmypdf.exceptions.**PdfMergeFailedError**

exit_code = 2

message = 'Failed to merge PDF image layer with OCR layer\n\nUsually this happens because the input PDF file is malformed and\nocrmypdf cannot automatically correct the problem on its own.\n\nTry using\n ocrmypdf --pdf-renderer sandwich [..other args..]\n'

exception ocrmypdf.exceptions.**PriorOcrFoundError**

exit_code = 6

exception ocrmypdf.exceptions.**SubprocessOutputError**

exit_code = 7

exception ocrmypdf.exceptions.**TesseractConfigError**

exit_code = 9

message = 'Error occurred while parsing a Tesseract configuration file'

exception ocrmypdf.exceptions.**UnsupportedImageFormatError**

exit_code = 2

16.3 ocrmypdf.helpers

@ocrmypdf.helpers.**deprecated**
Warn that function is deprecated.

exception ocrmypdf.helpers.**NeverRaise**
An exception that is never raised

class ocrmypdf.helpers.**Resolution**(*x*, *y*)
The number of pixels per inch in each 2D direction.

Resolution objects are considered "equal" for == purposes if they are equal to a reasonable tolerance.

ocrmypdf.helpers.**available_cpu_count**() → int
Returns number of CPUs in the system.

ocrmypdf.helpers.**check_pdf**(*input_file: pathlib.Path*) → bool
Check if a PDF complies with the PDF specification.

Checks for proper formatting and proper linearization. Uses pikepdf (which in turn, uses QPDF) to perform the checks.

ocrmypdf.helpers.**clamp**(*n*, *smallest*, *largest*)
Clamps the value of `n` to between `smallest` and `largest`.

ocrmypdf.helpers.**deprecated**(*func*)
Warn that function is deprecated.

ocrmypdf.helpers.**is_file_writable**(*test_file: os.PathLike*) → bool
Intentionally racy test if target is writable.

We intend to write to the output file if and only if we succeed and can replace it atomically. Before doing the OCR work, make sure the location is writable.

ocrmypdf.helpers.**is_iterable_notstr**(*thing: Any*) → bool
Is this is an iterable type, other than a string?

ocrmypdf.helpers.**monotonic**(*L: Sequence*) → bool
Does this sequence increase monotonically?

ocrmypdf.helpers.**page_number**(*input_file: os.PathLike*) → int
Get one-based page number implied by filename (000002.pdf -> 2)

ocrmypdf.helpers.**remove_all_log_handlers**(*logger*)
Remove all log handlers, usually used in a child process.

ocrmypdf.helpers.**safe_symlink**(*input_file: os.PathLike*, *soft_link_name: os.PathLike*)
Create a symbolic link at `soft_link_name`, which references `input_file`.

Think of this as copying `input_file` to `soft_link_name` with less overhead.

Use symlinks safely. Self-linking loops are prevented. On Windows, file copy is used since symlinks may require administrator privileges. An existing link at the destination is removed.

16.4 ocrmypdf.hocrtransform

class ocrmypdf.hocrtransform.**HocrTransform**(*, *hocr_filename: Union[str, pathlib.Path]*, *dpi: float*)
A class for converting documents from the hOCR format. For details of the hOCR format, see: http://kba.cloud/hocr-spec/

classmethod baseline(*element: xml.etree.ElementTree.Element*) → Tuple[float, float]
Returns a tuple containing the baseline slope and intercept.

classmethod element_coordinates(*element: xml.etree.ElementTree.Element*) → *ocrmypdf.hocrtransform.Rect*
Returns a tuple containing the coordinates of the bounding box around an element

pt_from_pixel(*pxl*) → *ocrmypdf.hocrtransform.Rect*
Returns the quantity in PDF units (pt) given quantity in pixels

classmethod replace_unsupported_chars(*s: str*) → str
Given an input string, returns the corresponding string that: * is available in the Helvetica facetype * does not contain any ligature (to allow easy search in the PDF file)

to_pdf(*, *out_filename: pathlib.Path*, *image_filename: Optional[pathlib.Path] = None*, *show_bounding_boxes: bool = False*, *fontname: str = 'Helvetica'*, *invisible_text: bool = False*, *interword_spaces: bool = False*) → None

Creates a PDF file with an image superimposed on top of the text. Text is positioned according to the bounding box of the lines in the hOCR file. The image need not be identical to the image used to create the hOCR file. It can have a lower resolution, different color mode, etc.

Parameters

- **out_filename** – Path of PDF to write.
- **image_filename** – Image to use for this file. If omitted, the OCR text is shown.
- **show_bounding_boxes** – Show bounding boxes around various text regions, for debugging.
- **fontname** – Name of font to use.
- **invisible_text** – If True, text is rendered invisible so that is selectable but never drawn. If False, text is visible and may be seen if the image is skipped or deleted in Acrobat.
- **interword_spaces** – If True, insert spaces between words rather than drawing each word without spaces. Generally this improves text extraction.

exception ocrmypdf.hocrtransform.**HocrTransformError**

class ocrmypdf.hocrtransform.**Rect**(*x1: Any*, *y1: Any*, *x2: Any*, *y2: Any*)

A rectangle for managing PDF coordinates.

property x1
Alias for field number 0

property x2
Alias for field number 2

property y1
Alias for field number 1

property y2
Alias for field number 3

16.5 ocrmypdf.pdfa

Utilities for PDF/A production and confirmation with Ghostspcript.

ocrmypdf.pdfa.**file_claims_pdfa**(*filename: pathlib.Path*)

Determines if the file claims to be PDF/A compliant.

This only checks if the XMP metadata contains a PDF/A marker. It does not do full PDF/A validation.

ocrmypdf.pdfa.**generate_pdfa_ps**(*target_filename: pathlib.Path*, *icc: str = 'sRGB'*)

Create a Postscript PDFMARK file for Ghostscript PDF/A conversion

pdfmark is an extension to the Postscript language that describes some PDF features like bookmarks and annotations. It was originally specified Adobe Distiller, for Postscript to PDF conversion.

Ghostscript uses pdfmark for PDF to PDF/A conversion as well. To use Ghostscript to create a PDF/A, we need to create a pdfmark file with the necessary metadata.

This function takes care of the many version-specific bugs and pecularities in Ghostscript's handling of pdfmark.

The only information we put in specifies that we want the file to be a PDF/A, and we want to Ghostscript to convert objects to the sRGB colorspace if it runs into any object that it decides must be converted.

Parameters

- **target_filename** – filename to save
- **icc** – ICC identifier such as 'sRGB'

References

Adobe PDFMARK Reference: https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/pdfmark_reference.pdf

16.6 ocrmypdf.quality

Utilities to measure OCR quality

class ocrmypdf.quality.**OcrQualityDictionary**(*, *wordlist: Iterable[str]*)

Manages a dictionary for simple OCR quality checks.

measure_words_matched(*ocr_text: str*) → float

Check how many unique words in the OCR text match a dictionary.

Words with mixed capitalized are only considered a match if the test word matches that capitalization.

Returns number of words that match / number

16.7 ocrmypdf.subprocess

Wrappers to manage subprocess calls

ocrmypdf.subprocess.**check_external_program**(*, *program: str*, *package: str*, *version_checker: Union[str, Callable]*, *need_version: str*, *required_for: Optional[str] = None*, *recommended=False*, *version_parser: Type[distutils.version.Version] = <class 'distutils.version.LooseVersion'>*)

Check for required version of external program and raise exception if not.

Parameters

- **program** – The name of the program to test.
- **package** – The name of a software package that typically supplies this program. Usually the same as program.
- **version_check** – A callable without arguments that retrieves the installed version of program.
- **need_version** – The minimum required version.
- **required_for** – The name of an argument of feature that requires this program.
- **recommended** – If this external program is recommended, instead of raising an exception, log a warning and allow execution to continue.
- **version_parser** – A class that should be used to parse and compare version numbers. Used when version numbers do not follow standard conventions.

ocrmypdf.subprocess.**get_version**(*program: str*, *, *version_arg: str = '--version'*, *regex='(\\d+(\\.\\d+)*)'*, *env=None*)

Get the version of the specified program

Parameters

- **program** – The program to version check.
- **version_arg** – The argument needed to ask for its version, e.g. `--version`.
- **regex** – A regular expression to parse the program's output and obtain the version.
- **env** – Custom `os.environ` in which to run program.

ocrmypdf.subprocess.**run**(*args*, *, *env=None*, *logs_errors_to_stdout: bool = False*, ***kwargs*) → subprocess.CompletedProcess

Wrapper around `subprocess.run()`

The main purpose of this wrapper is to log subprocess output in an orderly fashion that indentifies the responsible subprocess. An additional task is that this function goes to greater lengths to find possible Windows locations of our dependencies when they are not on the system PATH.

Arguments should be identical to `subprocess.run`, except for following:

Parameters **logs_errors_to_stdout** – If True, indicates that the process writes its error messages to stdout rather than stderr, so stdout should be logged if there is an error. If False, stderr is logged. Could be used with stderr=STDOUT, stdout=PIPE for example.

ocrmypdf.subprocess.**run_polling_stderr**(*args*, *, *callback: Callable[[str], None]*, *check: bool = False*, *env=None*, ***kwargs*) → subprocess.CompletedProcess

Run a process like `ocrmypdf.subprocess.run`, and poll stderr.

Every line of produced by stderr will be forwarded to the callback function. The intended use is monitoring progress of subprocesses that output their own progress indicators. In addition, each line will be logged if debug logging is enabled.

Requires stderr to be opened in text mode for ease of handling errors. In addition the expected encoding= and errors= arguments should be set. Note that if stdout is already set up, it need not be binary.

CHAPTER

SEVENTEEN

CONTRIBUTING GUIDELINES

Contributions are welcome!

17.1 Big changes

Please open a new issue to discuss or propose a major change. Not only is it fun to discuss big ideas, but we might save each other's time too. Perhaps some of the work you're contemplating is already half-done in a development branch.

17.2 Code style

We use PEP8, `black` for code formatting and `isort` for import sorting. The settings for these programs are in `pyproject.toml` and `setup.cfg`. Pull requests should follow the style guide. One difference we use from "black" style is that strings shown to the user are always in double quotes (") and strings for internal uses are in single quotes (').

17.3 Tests

New features should come with tests that confirm their correctness.

17.4 New Python dependencies

If you are proposing a change that will require a new Python dependency, we prefer dependencies that are already packaged by Debian or Red Hat. This makes life much easier for our downstream package maintainers.

Python dependencies must also be license-compatible. GPLv3 or AGPLv3 are likely incompatible with the project's license, but LGPLv3 is compatible.

17.5 New non-Python dependencies

OCRmyPDF uses several external programs (Tesseract, Ghostscript and others) for its functionality. In general we prefer to avoid adding new external programs.

17.6 Style guide: Is it OCRmyPDF or ocrmypdf?

The program/project is OCRmyPDF and the name of the executable or library is ocrmypdf.

17.7 Known ports/packagers

OCRmyPDF has been ported to many platforms already. If you are interesting in porting to a new platform, check with Repology to see the status of that platform.

Packager maintainers, please ensure that the command line completion scripts in `misc/` are installed.

17.8 Copyright and license

For contributions over 10 lines of code, please include your name to list of copyright holders for that file. The core program is licensed under MPL-2.0, test files and documentation under CC-BY-SA 4.0, and miscellaneous files under MIT. Please contribute code only that you wrote and you have the permission to contribute or license to us.

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

ocrmypdf.exceptions, 120
ocrmypdf.hocrtransform, 122
ocrmypdf.pdfa, 123
ocrmypdf.quality, 124
ocrmypdf.subprocess, 124

INDEX

Symbols

__str__() (*ocrmypdf.pluginspec.OcrEngine method*), 116

A

add_options() (*in module ocrmypdf.pluginspec*), 111
already_done_ocr (*ocrmypdf.exceptions.ExitCode attribute*), 120
angle (*ocrmypdf.pluginspec.OrientationConfidence attribute*), 116

B

bad_args (*ocrmypdf.exceptions.ExitCode attribute*), 120
BadArgsError, 120
baseline() (*ocrmypdf.hocrtransform.HocrTransform class method*), 122

C

check_external_program() (*in module ocrmypdf.subprocess*), 124
check_options() (*in module ocrmypdf.pluginspec*), 111
child_process_error (*ocrmypdf.exceptions.ExitCode attribute*), 120
confidence (*ocrmypdf.pluginspec.OrientationConfidence attribute*), 116
configure_logging() (*in module ocrmypdf*), 108
creator_tag() (*ocrmypdf.pluginspec.OcrEngine static method*), 116
ctrl_c (*ocrmypdf.exceptions.ExitCode attribute*), 120

D

debug (*ocrmypdf.Verbosity attribute*), 108
debug_all (*ocrmypdf.Verbosity attribute*), 108
default (*ocrmypdf.Verbosity attribute*), 108
deprecated() (*in module ocrmypdf.helpers*), 121
DpiError, 120

E

element_coordinates() (*ocrmypdf.hocrtransform.HocrTransform class method*), 122
encrypted_pdf (*ocrmypdf.exceptions.ExitCode attribute*), 120
EncryptedPdfError, 120
environment variable
    OMP_THREAD_LIMIT, 84
    TESSDATA_PREFIX, 84
exit_code (*ocrmypdf.exceptions.BadArgsError attribute*), 120
exit_code (*ocrmypdf.exceptions.DpiError attribute*), 120
exit_code (*ocrmypdf.exceptions.EncryptedPdfError attribute*), 120
exit_code (*ocrmypdf.exceptions.ExitCodeException attribute*), 120
exit_code (*ocrmypdf.exceptions.InputFileError attribute*), 121
exit_code (*ocrmypdf.exceptions.MissingDependencyError attribute*), 121
exit_code (*ocrmypdf.exceptions.OutputFileAccessError attribute*), 121
exit_code (*ocrmypdf.exceptions.PdfMergeFailedError attribute*), 121
exit_code (*ocrmypdf.exceptions.PriorOcrFoundError attribute*), 121
exit_code (*ocrmypdf.exceptions.SubprocessOutputError attribute*), 121
exit_code (*ocrmypdf.exceptions.TesseractConfigError attribute*), 121
exit_code (*ocrmypdf.exceptions.UnsupportedImageFormatError attribute*), 121
ExitCode (*class in ocrmypdf.exceptions*), 120
ExitCodeException, 120

F

file_access_error (*ocrmypdf.exceptions.ExitCode attribute*), 120
file_claims_pdfa() (*in module ocrmypdf.pdfa*), 123
filter_ocr_image() (*in module ocrmypdf.pluginspec*), 114
filter_page_image() (*in module ocrmypdf.pluginspec*), 114
filter_pdf_page() (*in module ocrmypdf.pluginspec*),

115

G

generate_hocr() (*ocrmypdf.pluginspec.OcrEngine static method*), 116
generate_pdf() (*ocrmypdf.pluginspec.OcrEngine static method*), 116
generate_pdfa() (*in module ocrmypdf.pluginspec*), 117
generate_pdfa_ps() (*in module ocrmypdf.pdfa*), 123
get_executor() (*in module ocrmypdf.pluginspec*), 112
get_logging_console() (*in module ocrmypdf.pluginspec*), 112
get_ocr_engine() (*in module ocrmypdf.pluginspec*), 115
get_orientation() (*ocrmypdf.pluginspec.OcrEngine static method*), 116
get_page_contexts() (*ocrmypdf.PdfContext method*), 119
get_path() (*ocrmypdf.PageContext method*), 119
get_path() (*ocrmypdf.PdfContext method*), 119
get_progressbar_class() (*in module ocrmypdf.pluginspec*), 112
get_version() (*in module ocrmypdf.subprocess*), 124

H

HocrTransform (*class in ocrmypdf.hocrtransform*), 122
HocrTransformError, 123

I

input_file (*ocrmypdf.exceptions.ExitCode attribute*), 120
InputFileError, 121
invalid_config (*ocrmypdf.exceptions.ExitCode attribute*), 120
invalid_output_pdf (*ocrmypdf.exceptions.ExitCode attribute*), 120

L

languages() (*ocrmypdf.pluginspec.OcrEngine static method*), 116

M

measure_words_matched() (*ocrmypdf.quality.OcrQualityDictionary method*), 124
message (*ocrmypdf.exceptions.EncryptedPdfError attribute*), 120
message (*ocrmypdf.exceptions.ExitCodeException attribute*), 120
message (*ocrmypdf.exceptions.PdfMergeFailedError attribute*), 121
message (*ocrmypdf.exceptions.TesseractConfigError attribute*), 121
missing_dependency (*ocrmypdf.exceptions.ExitCode attribute*), 120
MissingDependencyError, 121
module
    ocrmypdf.exceptions, 120
    ocrmypdf.hocrtransform, 122
    ocrmypdf.pdfa, 123
    ocrmypdf.quality, 124
    ocrmypdf.subprocess, 124

O

ocr() (*in module ocrmypdf*), 107
OcrEngine (*class in ocrmypdf.pluginspec*), 115
ocrmypdf.exceptions
    module, 120
ocrmypdf.hocrtransform
    module, 122
ocrmypdf.pdfa
    module, 123
ocrmypdf.quality
    module, 124
ocrmypdf.subprocess
    module, 124
OcrQualityDictionary (*class in ocrmypdf.quality*), 124
ok (*ocrmypdf.exceptions.ExitCode attribute*), 120
options (*ocrmypdf.PageContext attribute*), 119
options (*ocrmypdf.PdfContext attribute*), 119
OrientationConfidence (*class in ocrmypdf.pluginspec*), 116
origin (*ocrmypdf.PageContext attribute*), 119
origin (*ocrmypdf.PdfContext attribute*), 119
other_error (*ocrmypdf.exceptions.ExitCode attribute*), 120
OutputFileAccessError, 121

P

PageContext (*class in ocrmypdf*), 119
pageinfo (*ocrmypdf.PageContext attribute*), 119
pageno (*ocrmypdf.PageContext attribute*), 119
pdfa_conversion_failed (*ocrmypdf.exceptions.ExitCode attribute*), 120
PdfContext (*class in ocrmypdf*), 119
pdfinfo (*ocrmypdf.PdfContext attribute*), 120
PdfMergeFailedError, 121
plugin_manager (*ocrmypdf.PageContext attribute*), 119
plugin_manager (*ocrmypdf.PdfContext attribute*), 120
PriorOcrFoundError, 121
pt_from_pixel() (*ocrmypdf.hocrtransform.HocrTransform method*), 122

Q

quiet (*ocrmypdf.Verbosity attribute*), 108

R

rasterize_pdf_page() (*in module ocrmypdf.pluginspec*), 113
Rect (*class in ocrmypdf.hocrtransform*), 123
replace_unsupported_chars() (*ocrmypdf.hocrtransform.HocrTransform class method*), 122
run() (*in module ocrmypdf.subprocess*), 125
run_polling_stderr() (*in module ocrmypdf.subprocess*), 125

S

SubprocessOutputError, 121

T

TesseractConfigError, 121
to_pdf() (*ocrmypdf.hocrtransform.HocrTransform method*), 122

U

UnsupportedImageFormatError, 121

V

validate() (*in module ocrmypdf.pluginspec*), 113
Verbosity (*class in ocrmypdf*), 108
version() (*ocrmypdf.pluginspec.OcrEngine static method*), 116

X

x1 (*ocrmypdf.hocrtransform.Rect property*), 123
x2 (*ocrmypdf.hocrtransform.Rect property*), 123

Y

y1 (*ocrmypdf.hocrtransform.Rect property*), 123
y2 (*ocrmypdf.hocrtransform.Rect property*), 123