PDF-KungFoo with Ghostscript & Co.

100 Tips and Tricks for Clever PDF Creation and Handling

Kurt Pfeifle

PDF-KungFoo with Ghostscript & Co.

100 Tips and Tricks for Clever PDF Creation and Handling

Kurt Pfeifle

This book is for sale at http://leanpub.com/pdfkungfoo

This version was published on 2014-04-13

Tweet This Book!

Please help Kurt Pfeifle by spreading the word about this book on Twitter!

The suggested hashtag for this book is #pdfkungfoo.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#pdfkungfoo

Also By Kurt Pfeifle

PDF-KungFoo Workshop I

Contents

Metadata

```
Title:            100 Tipps and Tricks with Ghostscript &amp; Co.
Rights:           Copyright 2013  by <kurt.pfeifle@gmail.com>
Copyright:        Copyright 2013  by <kurt.pfeifle@gmail.com>
Coverage:         Practical Examples and Commandlines Explained
Creator:          <kurt.pfeifle@gmail.com>
Type:             PDF, Fonts, Images + Prepress Knowhow
Description:      A Prepress Professional's Brain Backup
Version:          v0.1.06
Publication date: 2013-08-20
Publisher:        Self-Published
Language:         en
Author:           <kurt.pfeifle@gmail.com>
Affiliation:      Kurt Pfeifle IT-Beratung
Affiliation:      PDF Association
Affiliation:      PDF/A Competence Center
Affiliation:      PDF/X Ready Switzerland
Affiliation:      ForschungsGesellschaft Druck e.V. (Fogra)
Languages:        english
Date:             2013, July 31st
Identifier UUID:  urn:uuid:8cc9b6b0-3849-4624-8d19-bf76d5948875
```

Changelog (major changes only)

Version	Date	Change Description
v0.1.00	*2013-07-07*	Initial version (unreleased)
v0.1.01	*2013-07-08*	Added chapter "How can I convert PCL to PDF?"
v0.1.02	*2013-07-09*	Added chapter "How can I convert XPS to PDF?"
v0.1.03	*2013-07-10*	Added a Changelog
		Added a Table of Content
v0.1.04	*2013-07-10*	Added chapter "Why doesn't Acrobat Distiller embed all fonts fully?"
		Added chapter "How can I unittest a Python function that draws PDF graphics?"
v0.1.05	*2013-07-11*	Added chapter "How can I query Ghostscript default settings?"
		Added chapter "How can I compare 2 PDFs on the commandline?"
		Added chapter "How can I remove white margins from PDF?"
v0.1.06	*2013-07-11*	First publicly available version (released); ca. 10% complete
v0.1.07	*2013-07-12*	Added chapter "How can I extract embedded fonts from a PDF as valid font files?"
v0.1.08	*2013-07-13*	Added appendix "About the author"
v0.1.09	*2013-07-14*	Added chapter "How can I add annotations to a PDF?"
v0.1.10	*2013-07-15*	Added chapter "How can I let Ghostscript determine the number of PDF pages?"
v0.1.11	*2013-08-29*	Added chapter "How do I make Ghostscript show all fonts it can find on my local system?"
v0.1.12	*2013-08-31*	Updated chapter "Downloading the tools" to reflect GS 9.09 release
v0.1.13	*2013-09-01*	Second publicly available version (released); ca. 11% complete
v0.1.14	*2013-09-02*	Added chapter "How can I convert fonts to outlines in an existing PDF?"
v0.1.15	*2013-09-03*	Added chapter "Hints for Linux, Windows, Mac OS X and Unix Users"
v0.1.16	*2013-09-04*	Added chapter "How can I use invisible fonts in a PDF?"
v0.1.17	*2013-09-05*	Added chapter "How can I make invisible OCR information in scanned PDFs visible?"
v0.1.18	*2013-09-06*	Added illustrations to newly added chapter
v0.1.19	*2013-09-07*	Re-wrote chapter "How can I compare 2 PDFs on the commandline?"
		Added illustrations and examples to overhauled chapter for readers to reproduce
		Third publicly available version (released); ca. 16% complete
v0.1.20	*2013-09-08*	Added chapter "How can I convert a color PDF into grayscale?"
v0.1.21	*2013-09-09*	Added illustrations to newly added chapter
v0.1.22	*2013-09-10*	Added chapter "How can I understand what this funny 'pdfmark' stuff is about?"
v0.1.23	*2013-09-11*	Added chapter "How can I use `pdfmark` with Ghostscript to change PDF metadata?"
v0.1.24	*2013-09-12*	Added illustrations to newly added chapter
v0.1.25	*2013-09-13*	Added chapter "How can I use Ghostscript as a calculator inside the shell?"
v0.1.26	*2013-09-14*	Added chapter "Do you also use non-FOSS tools for your PDF-related work? If so, which?"
		4th publicly available version (released); ca. 21% complete
v0.1.27	*2013-09-22*	Added chapter "How can I re-order pages in a PDF?"
		5th publicly available version (released); ca. 22% complete
v0.1.28	*2013-10-24*	Added chapter "Why do you call Apple's Preview.app *'evil, malicious and ambidextrous to unsuspecting users'*?"
v0.1.29	*2014-04-02*	Updated chapter "Downloading the tools" to reflect GS 9.14 release and other updates
v0.1.30	*2014-04-13*	6th publicly available version (released); ca. 23% complete

Introduction

> *You do not want to read introductionary blah-blah? Have Ghostscript and other tools already installed? Want to immediately dive into the thick of PDF KungFoo? Read immediately a chapter with the real stuff? Then you could click to jump to…*
>
> - *…the **first one how to convert PCL files to PDF**. But my recommendation is…*
> - *…the **method to make the invisible OCR text visible**.*

This book is still "work in progress". It summarizes some of the practical solutions I applied to real-world problems encountered by my clients.

Most of the book's chapters deal with Ghostscript commands. But sometimes I also refer to other helper utilities, which I employ when Ghostscript isn't the right tool for the job.

Each chapter is intended to be of immediate practical value, and each one can stand on its own, giving the reader a basic or more advanced "recipe" that can be applied and adapted to his own situation, while at the same time giving additional background information and highlighting technical concepts in context.

While this book is still work in progress, readers are encouraged to submit their own suggestions and questions about topics to be included into the final version.

My experience in the prepress world and in the printing industry spans over 2 decades. To date, I've used Ghostscript and other Free Software tools for more than 15 years. Most of the 'problems' and practical tasks I describe here have been posed to me…

- …either from paying customers, whom I helped through consulting, troubleshooting, training or software development activities,
- …or from emails I received (sometimes from people I have not heard of before or after) asking me some particular question about a problem,
- …or via some public internet forum, newsgroup or platform where people ask IT- or programming related questions, most prominently on StackOverflow.com.

Luckily I kept a record of the most interesting and of the most commonly asked things.

What you can read here is a condensed summary from my archives. Sometimes I didn't write paragraphs completely from scratch, but copied them straight from my old mails. So, if you come across some sentence in the "Question" or the "Answer" section of the coming chapters which sounds familiar to you: maybe it's because *you* sent me the question before, or because you received the same answer from me years ago. Over time, I may decide to edit, polish and straighten many of the original, still "raw" pieces in this book. However, this may also depend on readers' general feedback.

Be warned though: this document is not necessarily a comprehensive, systematic tutorial! Some of the snippets explained in different chapters may be duplicates and therefor could be seen as redundant. However, should you end up reading and working through all chapters of the booklet, you'll remember these parts better and you may have gained a rather complete picture of Ghostscript's capabilities :-)

While I didn't do a precise count: I'm pretty sure that a newbie Ghostscript user will easily find 100 different pieces of practical Ghostscript usage snippets here, even if the book currently does not (yet) contain 100 distinct chapters. Experienced users will also be able to find one or the other 'gem of wisdom'.

All in all I hope you'll find my 'PDF-KungFoo – 100 Tips + Tricks for Ghostscript & Co.' useful. I intend to expand and update this document over time. Readers will be entitled to free updates. So I hope, in a year or two, you will have a document which could rather be named '100 Chapters with 1000 Tipps + Tricks for Ghostscript & Co.'

* – Kurt Pfeifle

I 100 Tipps and Tricks

1 Downloading the tools

My own preferred work environments are Linux and Mac OS X. However, most of the methods explained in the following chapters can be applied to Windows too. Readers will benefit most from this book if they reproduce and play with some of the example commandlines given. To do that they should install the following tools.

1.1 Windows

These are the preferred download sites. They are the ones which are offered by the *respective developers themselves*. Do not ever use any other, third-party download links, unless you really know what you do!

- Download **Ghostscript**[1]:
  http://downloads.ghostscript.com/public/[2]
  Currently available are installer files for Windows:
  - gs914w32.exe[3] (for all Windows OS 32bit, but also works on Windows 64 bit)
  - gs914w64.exe[4] (does not work on Windows 32bit – but Ghostscript developers warn anyway: the 64bit version may even run slower on 64bit than does the 32bit version on 64bit Windows!)
  - Keep your eyes open for newer versions appearing in that directory!
- Download **GhostPCL**[5]:
  http://downloads.ghostscript.com/public/binaries/[6]
  Currently available are pre-compiled 32-bit binaries for Windows embedded in a *.zip file:
  - ghostpcl-9.14-win32.zip[7]
  - Keep your eyes open for newer versions appearing in that directory!
- Download **GhostXPS**[8]:
  http://downloads.ghostscript.com/public/binaries/[9]
  Currently available are pre-compiled 32-bit binaries for Windows embedded in a *.zip file:
  - 32-bit Version: ghostxps-9.14-win32.zip[10]
  - Keep your eyes open for newer versions appearing in that directory!
- Download **XPDF-Utils**[11] (CLI tools: pdffonts, pdfinfo, pdfimages, pdftotext, pdftops...)
  http://www.foolabs.com/xpdf/download.html[12]
  Current file version:

[1] http://www.ghostscript.com/download/gsdnld.html
[2] http://downloads.ghostscript.com/public/
[3] http://downloads.ghostscript.com/public/gs914w32.exe
[4] http://downloads.ghostscript.com/public/gs914w64.exe
[5] http://www.ghostscript.com/download/gpcldnld.html
[6] http://downloads.ghostscript.com/public/binaries/
[7] http://downloads.ghostscript.com/public/binaries/ghostpcl-9.14-win32.zip
[8] http://www.ghostscript.com/download/gxpsdnld.html
[9] http://downloads.ghostscript.com/public/binaries/
[10] http://downloads.ghostscript.com/public/binaries/ghostxps-9.14-win32.zip
[11] http://www.foolabs.com/xpdf/about.html
[12] http://www.foolabs.com/xpdf/download.html

  - xpdfbin-win-3.03.zip[13]
    The _.zip contains pre-compiled binaries for Windows. It’s not an installer: just unpack anywhere you want, modify the %PATH% variable to find the binaries and start them from an CMD window._
    You may want to additionally add one or more of the *‘Language Support Packages’.
  - Keep your eyes open for newer versions appearing in that directory!
- Download **qpdf**[14]:
  32-bit Version: http://sourceforge.net/projects/qpdf/files/qpdf/5.1.1/qpdf-5.1.1-bin-mingw32.zip[15]
  64-bit Version: http://sourceforge.net/projects/qpdf/files/qpdf/5.1.1/qpdf-5.1.1-bin-mingw64.zip[16]
  64-bit Version: http://sourceforge.net/projects/qpdf/files/qpdf/5.1.1/qpdf-5.1.1-bin-msvc64.zip[17]
- Download **pdftk**[18]:
  http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/pdftk_server-2.02-win-setup.exe[19]
- Download **ImageMagick**[20]:
  http://www.imagemagick.org/download/binaries/[21] (make sure to select the correct and most current download for your system)

A Warning Note about Downloading from Sourceforge

Sourceforge.net used to be a resource that was very useful for the Open Source community. However, in recent years the site has become more and more overloaded with ads. Some of the more recent stories on the Internet even suggest that the site may be poisoned with links that lead you to third party *‘drive-by’* malware download sites.

Unfortunately, some of the tools advertised in this eBook (such as `qpdf`) still do host their source code or their Windows binaries on this platform. I’m hoping the developers of these tools will find some other hosting soon…

:-(

1.2 Mac OS X

My recommendation is to use the ‘MacPorts’ framework for installing additional software packages:

- http://www.macports.org/install.php[22]

After you have put MacPorts in place, open a Terminal.app window and start installing the packages by typing:

[13] ftp://ftp.foolabs.com/pub/xpdf/xpdfbin-win-3.03.zip
[14] http://qpdf.sf.net/
[15] http://sourceforge.net/projects/qpdf/files/qpdf/5.1.1/qpdf-5.1.1-bin-mingw32.zip/download
[16] http://sourceforge.net/projects/qpdf/files/qpdf/5.1.1/qpdf-5.1.1-bin-mingw64.zip/download
[17] http://sourceforge.net/projects/qpdf/files/qpdf/5.1.1/qpdf-5.1.1-bin-msvc64.zip/download
[18] http://www.pdflabs.com/tools/pdftk-server/
[19] http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/pdftk_server-2.02-win-setup.exe
[20] http://www.imagemagick.org/
[21] http://www.imagemagick.org/download/binaries/
[22] http://www.macports.org/install.php

```
sudo port -p install ghostscript poppler ImageMagick coreutils qpdf pdftk
```

Be aware that this may take a while. Ghostscript depends on additional packages for its functionality, like *libpng*, *jpeg*, *tiff*, *zlib* and more. The same applies for the other tools. The `port` command downloads, compiles and installs all these dependencies automatically, so this may take quite a while…

1.3 Linux

Debian, Ubuntu, …

You can run this command in a terminal window:

```
sudo apt-get install ghostscript poppler-utils ImageMagick coreutils qpdf
```

Or you use the package manager of your choice to find and install the packages…

RedHat, Fedora

Command in a terminal window:

```
sudo yum install ghostscript poppler-utils ImageMagick coreutils qpdf
```

Slackware

Command in a terminal window:

```
TODO
```

1.4 Documentation

```
TODO
```

2 How can I convert PCL to PDF?

Is it possible to use Ghostscript for converting PCL print files to PDF format? If so, how?

2.1 Answer

No, it's not possible with Ghostscript itself…

But yes!, it's very well possible with another cool piece of workhorse software from the same barn: its name ist *GhostPCL*.

Ghostscript developers in recent years integrated their sister products *GhostXPS*, *GhostPCL* and *GhostSVG* into their main Ghostscript source code tree[1], which switched from Subversion to Git some time ago. The complete family of products is now called *GhostPDL*. So all of these additional functionalities (load, render and convert XPS, PCL and SVG) are now available from there.

Previously, though GhostPCL was available as a source code tarball, it was hardly to be found on the 'net. The major Linux distributions (Debian, Ubuntu, Redhat, Fedora, OpenSUSE,…) don't provide packages for their users either. On MacPorts it is missing too.

This means you have to build the programs yourself from the sources. You could even compile the so-called *language switching* binary, `pspcl6`. This binary, in theory, can consume PCL, PDF and PostScript and convert this input to a host of other formats. Just run `make ls-product` in the top level Ghostscript source directory in order to build it. The resulting binary will end up in the *./language-switch/obj/* subdirectory. Run `make ls-install` in order to install it.

> ⚠ ***WARNING:*** While it worked for me whenever I needed it, Ghostscript developers recommend to stop using the language switching binary (since it's *'almost non-supported'* as they say, and it will possibly go away in the future).
>
> Instead they recommend to use the explicit binaries:
>
> - `pcl6` or `pcl6.exe` for PCL input,
> - `gsvg` or `gsvg.exe` for SVG input (also 'almost non-supported' but it may work better than the language switching binary `pspcl6`) and
> - `gxps` or `gxps.exe` for XPS input (support status unclear to me).

So for *'converting PCL code to PDF format'* as the request reads, you could use the `pcl6` command line utility, the PCL consuming sister product to Ghostscript's `gs` (Linux, Unix, Mac OS X) and `gswin32c.exe`/`gswin64c.exe` (Windows) which are PostScript and PDF input consumers.

[1] http://git.ghostscript.com/?p=ghostpdl.git;a=summary

Sample commandline (Windows):

```
pcl6.exe            ^
  -o output.pdf     ^
  -sDEVICE=pdfwrite ^
   [...more parameters as required (optional)...] ^
  -f input.pcl
```

Sample Commandline (Linux, Unix, Mac OS X):

```
pcl6                \
 -o output.pdf      \
 -sDEVICE=pdfwrite  \
  [...more parameters as required (optional)...] \
 -f input.pcl
```

Explanation

`-o output.pdf`

The `-o` parameter determines the location of the output. In this case it will be the file *output.pdf* in the current directory (since we did not specify any path prefix for the filename). At the same time, using `-o` saves us from typing `-dBATCH -dNOPAUSE -dSAFER`, because `-o` implicitly does also set these parameters.

`-sDEVICE=pdfwrite`

This parameter determines which kind of output to generate. In our current case it will be PDF. If you wanted to produce a multipage grayscale TIFF with CCITT compression, you would change that to `-sDEVICE=tiffg4` (don't forget to modify the output file name accordingly too: `-o output.tif`).

`-f input.pcl`

This parameter determines which file to read as input. In this case it is the file *input.pcl* in the current directory.

Update: The Ghostscript website now at least for Windows users offers pre-compiled 32-bit binaries for GhostPCL and GhostXPS.

- http://downloads.ghostscript.com/public/binaries/ghostpcl-9.14-win32.zip[2]
- http://downloads.ghostscript.com/public/binaries/ghostxps-9.14-win32.zip[3]

[2] http://downloads.ghostscript.com/public/binaries/ghostpcl-9.14-win32.zip

[3] http://downloads.ghostscript.com/public/binaries/ghostxps-9.14-win32.zip

See also the hints in *’[How can I convert XPS to PDF?](#convert-xps-to-pdf)’*.

TODO! Hint about the GhostPCL licensing. Esp. important: hint about the URW fonts which are not GPL (they require commercial licensing for commercial use).

3 How can I to convert XPS to PDF?

How can I convert XPS to PDF? Is this possible with Ghostscript?

3.1 Answer

Ghostscript developers in recent years have integrated a sister product named *GhostXPS* into their main Ghostscript source code tree[1], which is based on Git now. (They have also included two other products, named *GhostPCL* and *GhostSVG*.) The complete family of products is now called *GhostPDL*. So all of these additional functionalities (load, render and convert XPS, PCL and SVG) are now available from one location.

Unfortunately, none of the major Linux distributions (Debian, Ubuntu, Redhat, Fedora, OpenSUSE,...) do currently provide packages for their users. On MacPorts GhostXPS is missing too, as are GhostPCL and GhostSVG.

This means you have to build the programs yourself from the sources – unless you are a Windows user. In this case you are lucky: there is a *.zip container on the Ghostscript website, which contains a pre-compiled Win32 binary (which also runs on Windows 64 bit!):

- http://downloads.ghostscript.com/public/binaries/ghostxps-9.07-win32.zip[2]

While you're at it and build the code yourself, you could even build a so-called *language switching* binary. The Makefile has targets prepared for that. This binary can consume PCL, PDF and PostScript. It converts these input formats to a host of other file types. Just run `make ls-product && make ls-install` in the top level Ghostscript source directory in order to get it installed.

WARNING: While it worked for me whenever I needed it, Ghostscript developers recommend to stop using the language switching binary (since it's *'almost non-supported'* as they say, and it will possibly go away in the future).

Instead they recommend to use the explicit binaries, also supported as build targets in the Makefile:

- `pcl6` or `pcl6.exe` for PCL input,
- `gsvg` or `gsvg.exe` for SVG input (also 'almost non-supported') and
- `gxps` or `gxps.exe` for XPS input (support status unclear to me).

[1] http://git.ghostscript.com/?p=ghostpdl.git;a=summary

[2] http://downloads.ghostscript.com/public/binaries/ghostxps-9.07-win32.zip

Sample commandline (Windows):

```
gxps.exe            ^
 -o output.pdf      ^
 -sDEVICE=pdfwrite  ^
  [...more parameters as required (optional)...] ^
 -f input.xps
```

Sample commandline (Linux, Unix, Mac OS X):

```
gxps                \
 -o output.pdf      \
 -sDEVICE=pdfwrite  \
  [...more parameters as required (optional)...] \
 -f input.xps
```

Explanation

`-o output.pdf`

The `-o` parameter determines the location of the output. In this case it will be the file *output.pdf* in the current directory (since we did not specify any path prefix for the filename). At the same time, using `-o` saves us from typing `-dBATCH -dNOPAUSE -dSAFER`, because `-o` implicitly does also set these parameters.

`-sDEVICE=pdfwrite`

This parameter determines which kind of output to generate. In our current case that’s PDF. If you wanted to produce a PostScript level `2` file, you would change that to `-sDEVICE=ps2write` (don’t forget to modify the output file name accordingly too: `-o output.ps`).

`-f input.pcl`

This parameter determines which file to read as input. In this case it is the file *input.pcl* in the current directory.

See also the hints in *[’How can I convert PCL to PDF?’](#convert-pcl-to-pdf.html)*.

TODO! Hint about the GhostPCL licensing. Esp. important: hint about the URW fonts which are not GPL (they require commercial licensing for commercial use).

4 How can I unit test a Python function that draws PDF graphics?

> I'm writing a CAD application that outputs PDF files using the Cairo graphics library. A lot of the unit testing does not require actually generating the PDF files, such as computing the expected bounding boxes of the objects. However, I want to make sure that the generated PDF files "look" correct after I change the code.
>
> Is there an automated way to do this? How can I automate as much as possible? Do I need to visually inspect each generated PDF? How can I solve this problem without pulling my hair out?

4.1 Answer

I'm doing the same thing using a shell script on Linux that wraps

1. ImageMagick's `compare` command
2. the `pdftk` utility
3. Ghostscript (optionally)

(It would be rather easy to port this to a `.bat` Batch file for DOS/Windows.)

I have a few reference PDFs created by my application which are "known good". Newly generated PDFs after code changes are compared to these reference PDFs. The comparison is done pixel by pixel and is saved as a new PDF. In this PDF, all unchanged pixels are painted in white, while all differing pixels are painted in red.

This method utilizes three different building blocks: `pdftk`, `compare` (part of ImageMagick) and Ghostscript.

pdftk

Use this command to split multipage PDF files into multiple singlepage PDFs:

```
pdftk  reference.pdf  burst  output  somewhere/reference_page_%03d.pdf
pdftk  comparison.pdf burst  output  somewhere/comparison_page_%03d.pdf
```

compare

Use this command to create a “diff” PDF page for each of the pages:

```
compare                                  \
      -verbose                           \
      -debug coder -log "%u %m:%l %e"    \
       somewhere/reference_page_001.pdf  \
       somewhere/comparison_page_001.pdf \
      -compose src                       \
       somewhereelse/reference_diff_page_001.pdf
```

Ghostscript

Because of automatically inserted meta data (such as the current date+time), PDF output is not working well for MD5hash-based file comparisons.

If you want to automatically discover all cases which consist of purely white pages, you could also convert to a meta-data free bitmap format using the `bmp256` output device. You can do that for the original PDFs (reference and comparison), or for the diff-PDF pages:

```
gs                               \
  -o reference_diff_page_001.bmp \
  -r72                           \
  -g595x842                      \
  -sDEVICE=bmp256                \
   reference_diff_page_001.pdf

md5sum reference_diff_page_001.bmp
```

If the MD5sum is what you expect for an all-white page of `595x842` PostScript points, then your unit test passed.

5 How can I compare 2 PDFs on the commandline?

> **I'm looking for a Linux command line tool to compare two PDF files and save the diffs to a PDF outfile. The tool should create diff-PDFs in a batch-process. The PDF files are construction plans, so pure text-compare doesn't work.**
>
> **Something like:**
>
> ```
>    <tool> file1.pdf file2.pdf -o diff-out.pdf
> ```
>
> **Most of the tools I found convert the PDFs to images and compare them, but only with a GUI.**
>
> **Any other solution is also welcome.**

5.1 Answer

What you want can be achieved with using ImageMagick[1]'s `compare` command. And this will work on all important operating system platforms: Windows, Mac OS X, Linux and various Unix variations.

The basic command is very simple:

```
compare  file1.pdf  file2.pdf  delta1.pdf
```

First, please note: this only works well for PDFs which use the same page/media size.

The comparison is done pixel by pixel between the two input PDFs. In order to get the pixels, the pages are rendered to raster images first, by default using a resolution of 72 ppi (*pixels per inch*). The resulting file is an image showing the "diff" like this:

- Each pixel that is identical on each input file becomes white.
- Each pixel that is different between the two input files is painted in red.
- The 'source' file (the first one named in the command) will, for context, be used to provide a gray-scale background to the diff output.

[1] http://www.imagemagick.org/

The above command outputs a PDF file, `delta.pdf`. Should you prefer a PNG image or a JPEG image instead of a PDF, simply change the suffix of the 'delta' filename:

```
compare  file1.pdf  file2.pdf  delta2.png
compare  file1.pdf  file2.pdf  delta3.jpeg
```

In some cases the default resolution of 72 ppi used to render the PDF pages may be insufficient to uncover subtle differences. Or, on the contrary, it may over-emphasize differences which are triggered by extremely minimal shifts of individual characters or lines of text caused by some computational rounding of real numbers.

So, if you want to increase the resolution, add the `-density NNN` parameter to the commandline. To get 720 ppi images, use this:

```
compare  -density 720  file1.pdf  file2.pdf  delta4.pdf
```

Note, increasing the density/resolution of the output files also increases processing time and output file formats accordingly. A 10-fold increase in density leads to a 100-fold increase in the number of total pixels that need to be compared and processed.

All of the above examples do only work for 1-page PDF files. For multi-page PDFs you need to add a `[N]` notation to the file name, where `N` is the zero-based page number (page 1 is noted as `[0]`, page 2 as `[1]`, page 3 as `[2]`, and so forth). The following compares page 4 of `file1.pdf` with page 18 of `file2.pdf`:

```
compare  file1.pdf[3]  file2.pdf[17]  delta5.pdf
```

If you do not want the gray-scale background created from the source file, use a modified command:

```
compare  file1.pdf  file2.pdf  -compose src  delta1.pdf
```

This modification changes the output to purly red/white: all pixels which are identical between the two base files are red, identical pixels are white.

In case you do not like the red and white default colors to visualize the pixel differences, you can add the following commandline parameters:

- **`-highlight-color blue`** (change default color for pixel differences from ‘red’ to ‘blue’)
- **`-lowlight-color yellow`** (change default color for identical pixels from ‘white’ to ‘yellow’)

or any other color combination you desire. Allowed names for colors include `#RRGGBB` values for RGB shades.

Note, ImageMagick’s `compare` command does not process the PDF input files directly. `compare` originally was designed to process raster images only. You can easily test this by replacing the PDFs in above commands with some image files – just make sure that the files are ‘similar enough’ to give sensible results, and also ensure, that the compared images do have the same dimensions in width and height.

To process PDFs, ImageMagick needs to resort to Ghostscript as its *‘delegate’* program for processing PDF input. Ghostscript gets called behind the curtains by `compare` in order to create the raster files which then `compare` does its magic on.

To see the exact commandline parameters that ImageMagick uses for Ghostscript call, just add a `-verbose` parameter to the `compare` commands. The output on the terminal/console will be much more verbose and reveal what you want to know.

Examples

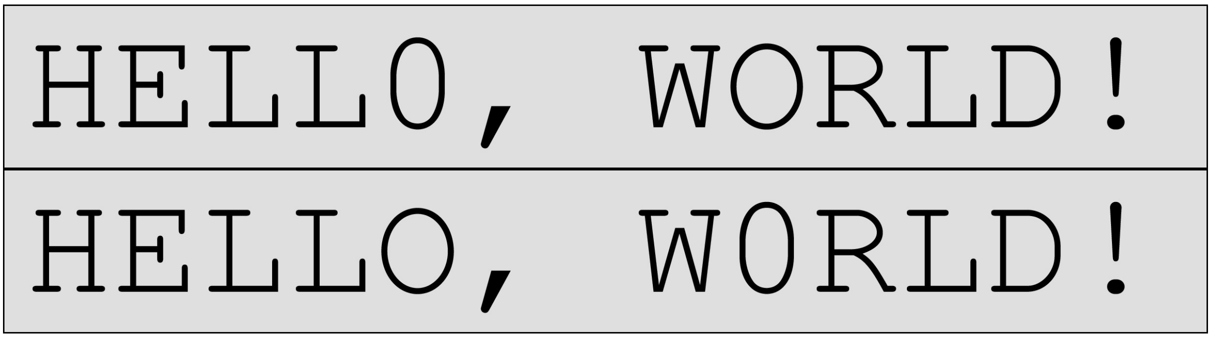
I’m using this very same method for example to discover minimal page display differences when font substitution in PDF processing comes into play.

It can easily be the case, that there is no visible difference between two PDFs, though they are extremely different in MD5 hashes, file sizes or internal PDF code structure. In this case the `delta1.pdf` output PDF page from the above command would become all-white. You could automatically discover this condition, so you only have to visually investigate the non-white PDFs by deleting the all-white ones automatically.

To give you a more visual impression about the way this comparison works, I’ve constructed a few different input files. I used Ghostscript to do this. (The exact commands I used are documented at the end of this chapter.)

Example 1

The following image shows two PDF pages side by side. Most people will notice from a quick look the differences between these two pages:



HELL0, WORLD!

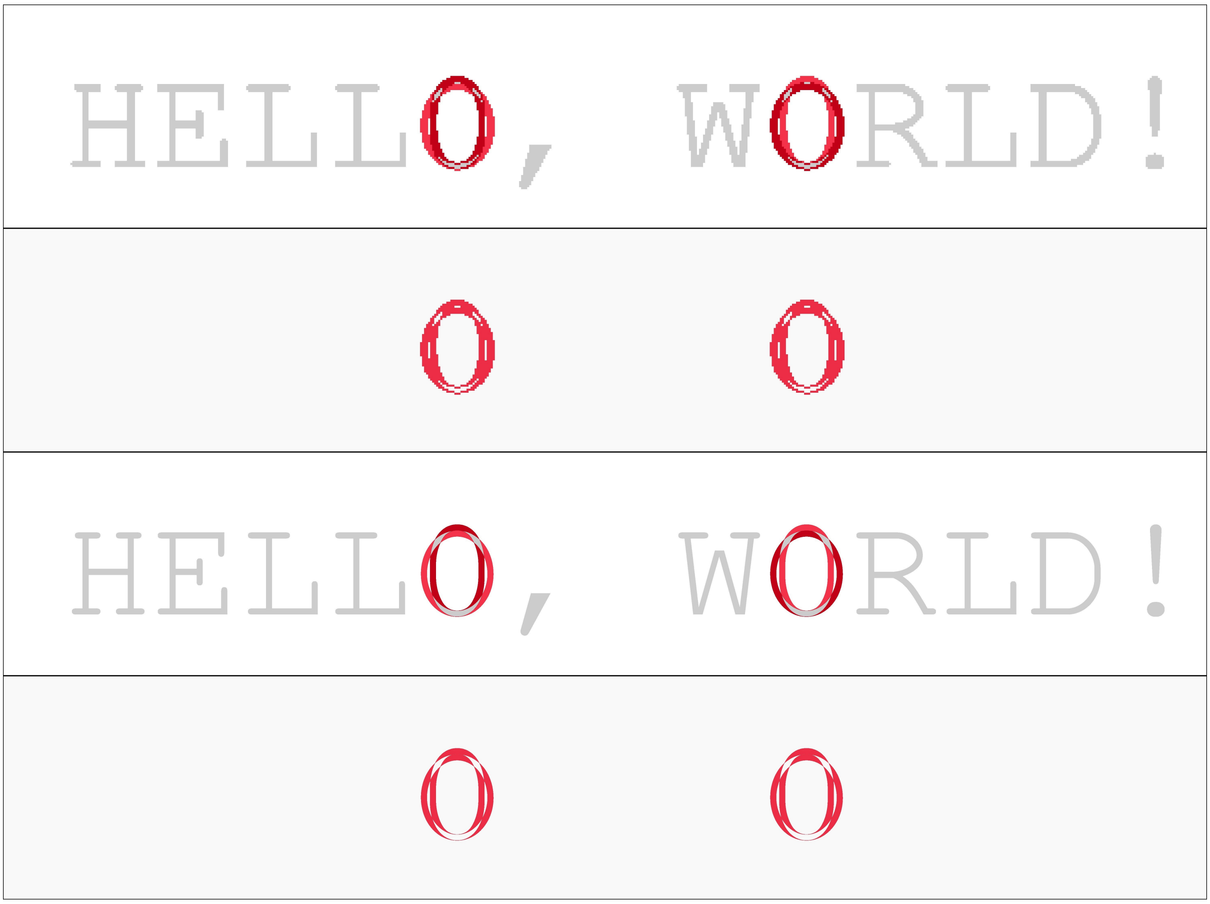
HELLO, W0RLD!

Two PDF pages which do differ – differences can be spotted by looking twice…

Now use the following commands to create a few different visualization of the ‘deltas’:

```
compare  file1.pdf  file2.pdf  delta1.png                 # default resolution, 72 ppi
compare  file1.pdf  file2.pdf  -compose src  delta2.png   # default resolution, 72 ppi
compare  -density 720  file1.pdf  file2.pdf  delta3.png   # resolution of 720 ppi
compare  -density 720  file1.pdf  file2.pdf  -compose src  delta4.png   # 720 ppi
```

The resulting 'delta' images are shown in the following picture.

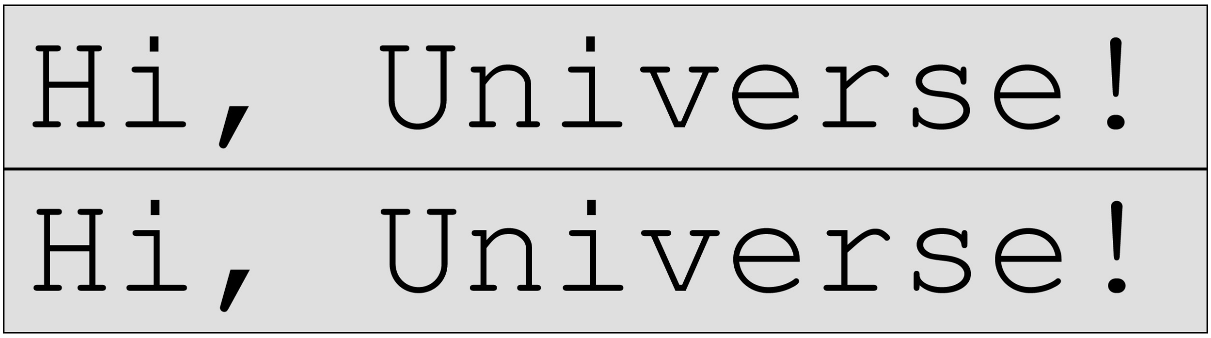


Four different visualizations of differences. The top two use a 72 ppi resolution, the bottom two a 720 ppi resolution. The 2nd and the 4th do not show a grayscale context background, but only white and red pixels.

As you can easily see, the 72 ppi-based comparison of the two input PDFs shows a clearly visible 'pixelization' of the results (top two images). Zoom in to see this in more detail. The 720 ppi version appears to come out much more smoothly. However, for this specific case 72 ppi would be 'good enough' to discover that in the two PDFs there was used a '0' (number zero) instead of an 'O' (capital letter 'o') at two different spots.

Example 2

The following image shows two other PDF pages side by side. Hardly anybody will be able to spot the differences between these, but some people will:



Hi, Universe!

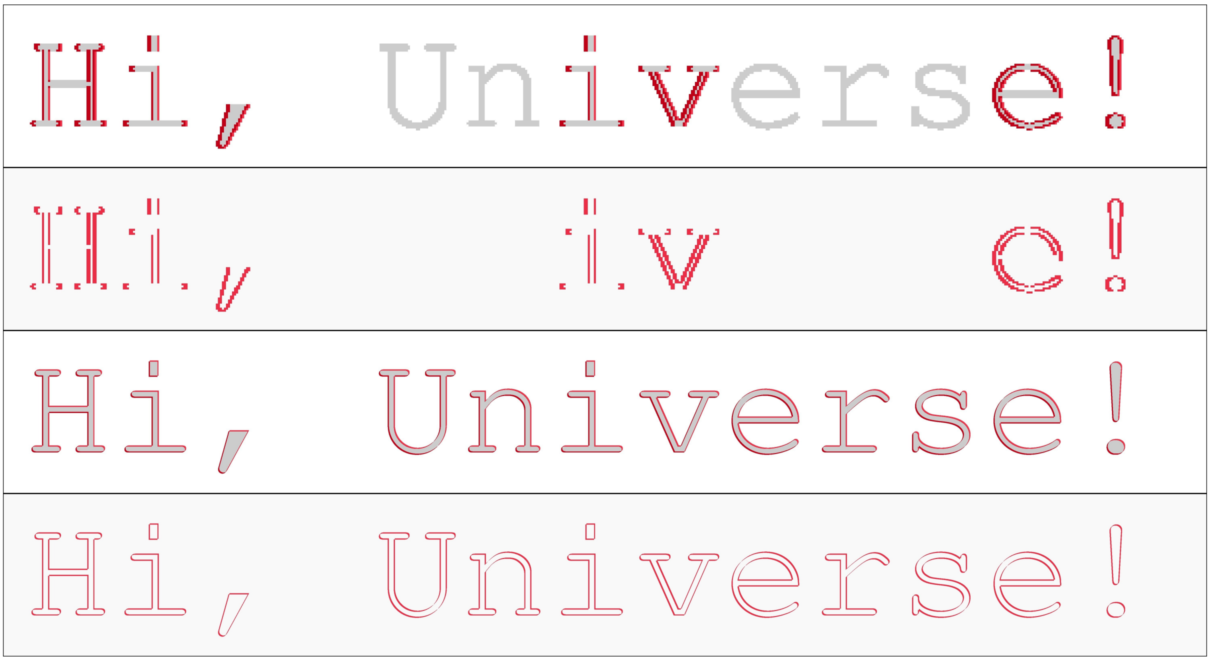
Hi, Universe!

Two PDF pages which do differ – differences can only be be spotted by looking *very* closely.

Now use the following commands to create a few different visualization of the ‘deltas’:

```
compare                file3.pdf  file4.pdf                delta5.pdf
compare                file3.pdf  file4.pdf  -compose src  delta6.pdf
compare  -density 720  file3.pdf  file4.pdf                delta7.pdf
compare  -density 720  file3.pdf  file4.pdf  -compose src  delta8.pdf
```

The resulting differences are shown in the following picture.

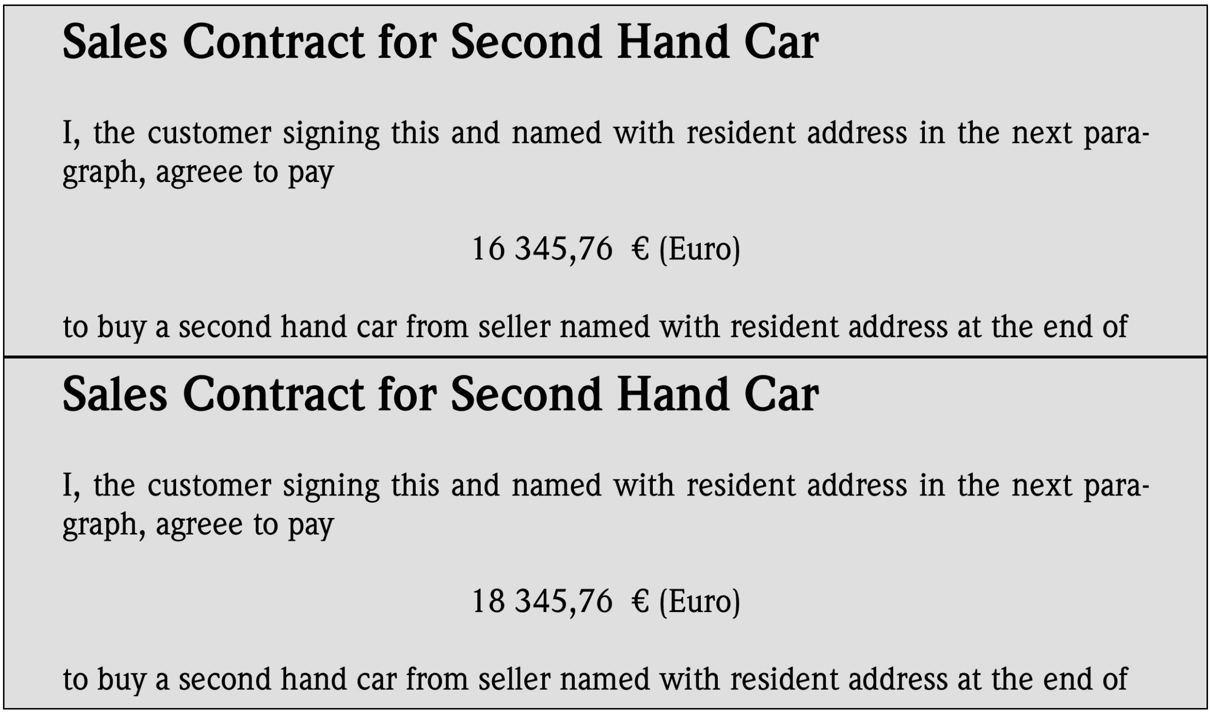


Four different ways to visualize the differences between the last two input files. Again a 72 ppi resolution for the top two and a 720 ppi resolution for the bottom ones. The 1st and the 3rd do show a grayscale context background, the others do not. Please zoom in to spot the finer pixel differences between the different resolutions…

Again, the 72 ppi-based comparison of the two input PDFs shows a clearly visible ‘pixelization’ of the results (top two images). The 720 ppi version does show the differences much more clearly: it is just that the text is shifted slightly to the left and to the top in the case of the second input. If you zoom in enough into the 720 ppi versions, you can even count the number of pixels: the shift for each single character of the text is constistenlty 5 pixels to the right and 5 pixels to the top. The 72 ppi version cannot bring out this subtle difference so clearly: at this resolution the shift is only 1/2 pixel to the right and 1/2 pixel to the top. This means that for some characters there is no shift occuring at all, and other characters move by a full pixel in either direction. This becomes clearly visible in the fact that some characters do not look changed at all while others clearly do.

Example 3

The following image shows two other PDF documents. Can you spot the difference?



Sales Contract for Second Hand Car

I, the customer signing this and named with resident address in the next paragraph, agreee to pay

16 345,76 € (Euro)

to buy a second hand car from seller named with resident address at the end of

Sales Contract for Second Hand Car

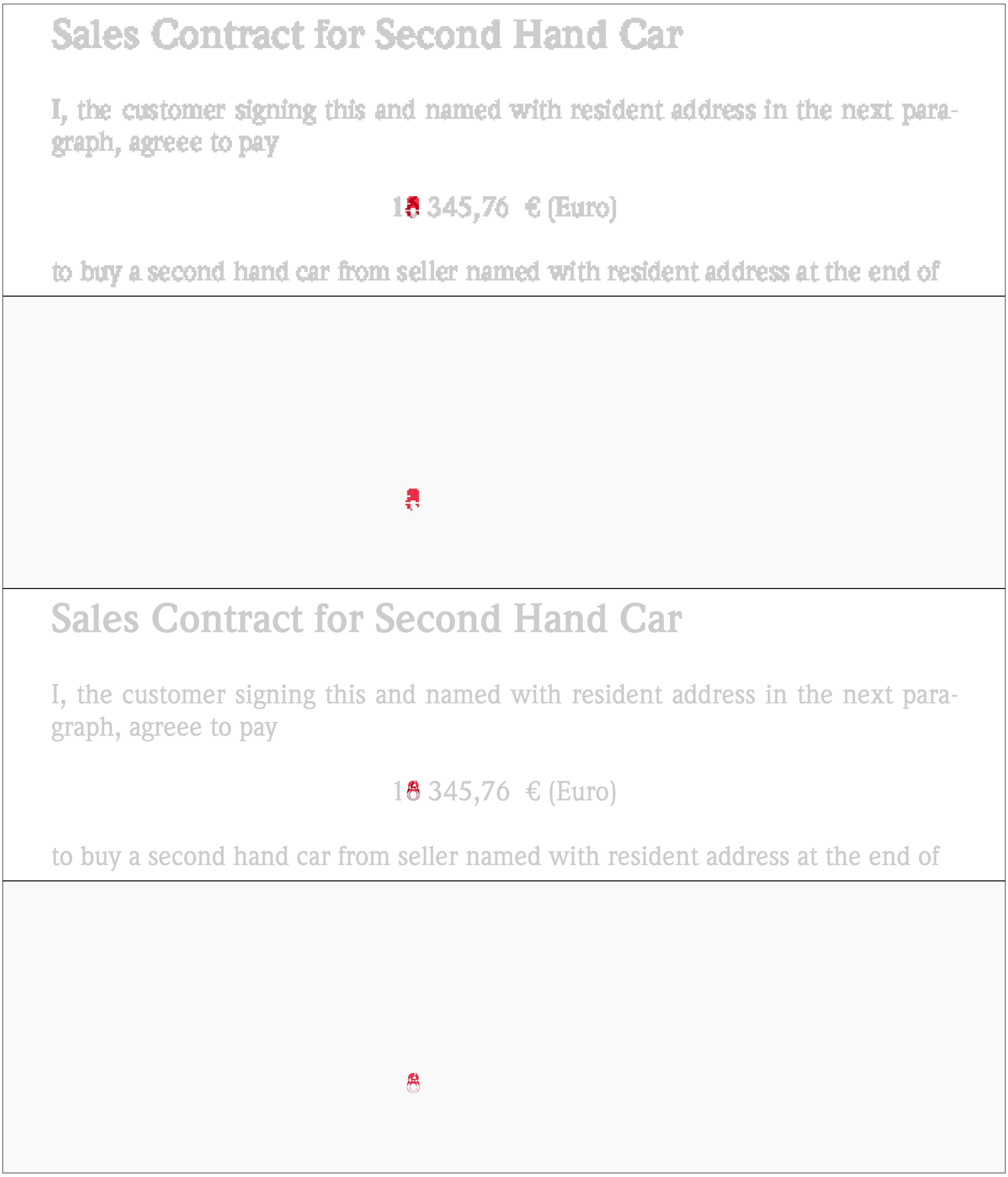
I, the customer signing this and named with resident address in the next paragraph, agreee to pay

18 345,76 € (Euro)

to buy a second hand car from seller named with resident address at the end of

Two PDF documents which do differ. Try to spot the difference!

Creating visualizations in red/white pixels will give the following results.



Sales Contract for Second Hand Car

I, the customer signing this and named with resident address in the next paragraph, agreee to pay

18 345,76 € (Euro)

to buy a second hand car from seller named with resident address at the end of

Sales Contract for Second Hand Car

I, the customer signing this and named with resident address in the next paragraph, agreee to pay

18 345,76 € (Euro)

to buy a second hand car from seller named with resident address at the end of

Four different ways to visualize the differences between the last two input files. Again a 72 ppi resolution for the top two and a 720 ppi resolution for the bottom ones. The 1st and the 3rd do show a grayscale context background, the others do not…

If you have access to the original delta files and zoom in on no. 3 you can clearly see that the second document contains a changed prize: going up by 2.000 $US by change the original '6' to an '8'.

Update

For those of you who want to reproduce the commands shown above, you'd also need access to the same source files I used. That's easy: I used Ghostscript to create these example input PDFs. Here are the commands for this:

```
gs                                            \
   -o file1.pdf                               \
   -sDEVICE=pdfwrite                          \
   -g5950x1100                                \
   -c "/Courier findfont 72 scalefont setfont \
       30   30   moveto (HELL0, WORLD\!) show \
       showpage"

gs                                            \
   -o file1.pdf                               \
   -sDEVICE=pdfwrite                          \
   -g5950x1100                                \
   -c "/Courier findfont 72 scalefont setfont \
       30   30   moveto (HELLO, W0RLD\!) show \
       showpage"

gs                                            \
   -o file1.pdf                               \
   -sDEVICE=pdfwrite                          \
   -g5950x1100                                \
   -c "/Courier findfont 72 scalefont setfont \
       30   30   moveto (Hi, Universe\!) show \
       showpage"

gs                                            \
   -o file1.pdf                               \
   -sDEVICE=pdfwrite                          \
   -g5950x1100                                \
   -c "/Courier findfont 72 scalefont setfont \
       30.5 30.5 moveto (Hi, Universe\!) show \
       show showpage"
```

6 How can I remove white margins from PDF pages?

> I would like to know a way to remove white margins from a PDF file. Just like Adobe Acrobat X Pro does. I understand it will not work with every PDF file.
>
> I would guess that the way to do it, is by getting the text margins, then cropping out of that margins.
>
> PyPdf is preferred.

```
iText finds text margins based on this code:



    public void addMarginRectangle(String src, String dest)
        throws IOException, DocumentException {
        PdfReader reader = new PdfReader(src);
        PdfReaderContentParser parser = new PdfReaderContentParser(reader);
        PdfStamper stamper = new PdfStamper(reader, new FileOutputStream(RESULT));
        TextMarginFinder finder;
        for (int i = 1; i <= reader.getNumberOfPages(); i++) {
            finder = parser.processContent(i, new TextMarginFinder());
            PdfContentByte cb = stamper.getOverContent(i);
            cb.rectangle(finder.getLlx(), finder.getLly(),
                finder.getWidth(), finder.getHeight());
            cb.stroke();
        }
        stamper.close();
    }
```

6.1 Answer

I'm not too familiar with PyPDF, but I know Ghostscript will be able to do this for you. Here are links to some other answers on similar questions:

1. **Convert PDF 2 sides per page to 1 side per page**[1] (SuperUser.com)
2. **Freeware to split a pdf's pages down the middle?**[2] (SuperUser.com)
3. **Cropping a PDF using Ghostscript 9.01**[3] (StackOverflow.com)

The third answer is probably what made you say '*I understand it will not work with every PDF file*'. It uses the *pdfmark* command to try and set the `/CropBox` into the PDF page objects.

[1] http://superuser.com/a/189109/40894

[2] http://superuser.com/a/235401/40894

[3] http://stackoverflow.com/a/6184547/359307

The method of the first two answers will most likely succeed where the third one fails. This method uses a PostScript command snippet of `<</PageOffset [NNN MMM]>> setpagedevice` to shift and place the PDF pages on a (smaller) media size defined by the `-gNNNNxMMMM` parameter (which defines device width and height in pixels).

If you understand the concept behind the first two answers, you'll easily be able to adapt the method used there to crop margins on all 4 edges of a PDF page:

An example command to crop a letter sized PDF (8.5x11in == 612x792pt) by half an inch (==36pt) on each of the 4 edges (command is for Windows):

```
gswin32c.exe         ^
   -o cropped.pdf    ^
   -sDEVICE=pdfwrite ^
   -g5400x7200       ^
   -c "<</PageOffset [-36 -36]>> setpagedevice" ^
   -f input.pdf
```

The resulting page size will be 7.5x10in (== 540x720pt). To do the same on Linux or Mac, use:

```
gs                   \
   -o cropped.pdf    \
   -sDEVICE=pdfwrite \
   -g5400x7200       \
   -c "<</PageOffset [-36 -36]>> setpagedevice" \
   -f input.pdf
```

Update: How to determine 'margins' with Ghostscript

A comment asked for 'automatic' determination of the white margins. You can use Ghostscript's too for this. Its `bbox` device can determine the area covered by the (virtual) ink on each page (and hence, indirectly the whitespace for each edge of the canvas).

Here is the command:

```
gs                     \
  -q -dBATCH -dNOPAUSE \
  -sDEVICE=bbox        \
   input.pdf
```

Output (example):

```
%%BoundingBox: 57 29 562 764
%%HiResBoundingBox: 57.265030 29.347046 560.245045 763.649977
%%BoundingBox: 57 28 562 667
%%HiResBoundingBox: 57.265030 28.347046 560.245045 666.295011
```

The `bbox` device renders each PDF page in memory (without writing any output to disk) and then prints the BoundingBox and HiResBoundingBox info to `stderr`. You may modify this command like that to make the results more easy to parse:

```
gs                       \
    -q -dBATCH -dNOPAUSE \
    -sDEVICE=bbox        \
     input.pdf           \
     2>&1                \
 | grep -v HiResBoundingBox
```

Output (example):

```
%%BoundingBox: 57 29 562 764
%%BoundingBox: 57 28 561 667
```

This would tell you...

- ...that the lower left corner of the content rectangle of **Page 1** is at coordinates `[57 29]` with the upper right corner is at `[562 741]`
- ...that the lower left corner of the content rectangle of **Page 2** is at coordinates `[57 28]` with the upper right corner is at `[561 667]`

This means:

- **Page 1** uses a whitespace of 57pt on the left edge (`72pt == 1in == 25,4mm`).
- **Page 1** uses a whitespace of 29pt on the bottom edge.
- **Page 2** uses a whitespace of 57pt on the left edge.
- **Page 2** uses a whitespace of 28pt on the bottom edge.

As you can see from this simple example already, the whitespace is not exactly the same for each page. Depending on your needs (you likely want the same size for each page of a multi-page PDF, no?), you have to work out what are the minimum margins for each edge across all pages of the document.

Now what about the right and top edge whitespace? To calculate that, you need to know the original page size for each page. The most simple way to determine this: the `pdfinfo` utility. Example command for a 5 page PDF:

```
pdfinfo      \
  -f 1       \
  -l 5       \
   input.pdf \
| grep "Page "
```

Output (example):

```
Page    1 size: 612 x 792 pts (letter)
Page    2 size: 612 x 792 pts (letter)
Page    3 size: 595 x 842 pts (A4)
Page    4 size: 842 x 1191 pts (A3)
Page    5 size: 612 x 792 pts (letter)
```

This will help you determine the required canvas size and the required (maximum) white margins of the top and right edges of each of your new PDF pages.

These calculations can all be scripted too, of course.

But if your PDFs are all of a uniq page size, or if they are 1-page documents, it all is much easier to get done…

7 Using Ghostscript to get page size

> Is it possible to get the page size (from e.g. a PDF document page) using Ghostscript?
>
> I have seen the `bbox` device, but it returns the bounding box (it differs per page), not the TrimBox (or CropBox) of the PDF pages. (See Prepressure website[a] for info about page boxes.) Any other possibility?
>
> [a] http://www.prepressure.com/pdf/basics/page_boxes

7.1 Answer 1

Unfortunately it doesn't seem quite easy to get the (possibly different) page sizes (or *Boxes for that matter) inside a PDF with the help of Ghostscript.

But since you asked for other possibilities as well: a rather reliable way to determine the media sizes for each page (and even each one of the embedded {Trim,Media,Crop,Bleed}Boxes) is the commandline tool pdfinfo.exe. This utility is part of the XPDF tools from http://www.foolabs.com/xpdf/download.html[1]. You can run the tool with the `-box` parameter and tell it with `-f 3` to start at page 3 and with `-l 8` to stop processing at page 8.

Example output

```
C:\downloads>pdfinfo -box -f 1 -l 3 _IXUS_850IS_ADVCUG_EN.pdf
Creator:        FrameMaker 6.0
Producer:       Acrobat Distiller 5.0.5 (Windows)
CreationDate:   08/17/06 16:43:06
ModDate:        08/22/06 12:20:24
Tagged:         no
Pages:          146
Encrypted:      no
Page    1 size: 419.535 x 297.644 pts
Page    2 size: 297.646 x 419.524 pts
Page    3 size: 297.646 x 419.524 pts
Page    1 MediaBox:     0.00     0.00   595.00   842.00
Page    1 CropBox:     87.25   430.36   506.79   728.00
Page    1 BleedBox:    87.25   430.36   506.79   728.00
Page    1 TrimBox:     87.25   430.36   506.79   728.00
Page    1 ArtBox:      87.25   430.36   506.79   728.00
Page    2 MediaBox:     0.00     0.00   595.00   842.00
Page    2 CropBox:    148.17   210.76   445.81   630.28
Page    2 BleedBox:   148.17   210.76   445.81   630.28
Page    2 TrimBox:    148.17   210.76   445.81   630.28
```

[1] http://www.foolabs.com/xpdf/download.html

```
Page    2 ArtBox:     148.17   210.76   445.81   630.28
Page    3 MediaBox:     0.00     0.00   595.00   842.00
Page    3 CropBox:    148.17   210.76   445.81   630.28
Page    3 BleedBox:   148.17   210.76   445.81   630.28
Page    3 TrimBox:    148.17   210.76   445.81   630.28
Page    3 ArtBox:     148.17   210.76   445.81   630.28
File size:      6888764 bytes
Optimized:      yes
PDF version:    1.4
```

7.2 Answer 2

Meanwhile I found a different method. This one uses Ghostscript only (just as you required). No need for additional third party utilities.

This method uses a little helper program, written in PostScript, shipping with the source code of Ghostscript. Look in the *toolbin* subdir for the `pdf_info.ps` file.

The included comments say you should run it like this in order to list fonts used, media sizes used

```
gswin32c -dNODISPLAY   ^
   -q                  ^
   -sFile=____.pdf     ^
    [-dDumpMediaSizes] ^
    [-dDumpFontsUsed [-dShowEmbeddedFonts]] ^
    toolbin/pdf_info.ps
```

I did run it on a local example file, with commandline parameters that ask for the media sizes only (not the fonts used). Here is the result:

```
C:\> gswin32c    ^
     -dNODISPLAY ^
     -q          ^
     -sFile=c:\downloads\_IXUS_850IS_ADVCUG_EN.pdf ^
     -dDumpMediaSizes ^
      C:/gs8.71/lib/pdf_info.ps


 c:\downloads\_IXUS_850IS_ADVCUG_EN.pdf has 146 pages.
 Creator: FrameMaker 6.0
 Producer: Acrobat Distiller 5.0.5 (Windows)
 CreationDate: D:20060817164306Z
 ModDate: D:20060822122024+02'00'
```

```

      Page 1 MediaBox: [ 595 842 ] CropBox: [ 419.535 297.644 ]
      Page 2 MediaBox: [ 595 842 ] CropBox: [ 297.646 419.524 ]
      Page 3 MediaBox: [ 595 842 ] CropBox: [ 297.646 419.524 ]
      Page 4 MediaBox: [ 595 842 ] CropBox: [ 297.646 419.524 ]
      [....]
```

II Fonts

8 How can I extract embedded fonts from a PDF as valid font files?

> I'm aware of the `pdftk.exe` utility that can indicate which fonts are used by a PDF, and whether they are embedded or not.
>
> Now the problem: given I had PDF files with embedded fonts – how can I extract those fonts in a way that they are re-usable as regular font files? Are there (preferably free) tools which can do that? Also: can this be done programmatically with, say, iText?

You have several options. All these methods work on Linux as well as on Windows or Mac OS X. However, be aware that most PDFs do not include to full, complete fontface when they have a font embedded. Mostly they include just the *subset* of glyphs used in the document.

8.1 Method 1: Using `pdftops`

One of the most frequently used methods to do this on *nix systems consists of the following steps:

1. Convert the PDF to PostScript, for example by using XPDF's **`pdftops`**[1] (on Windows: `pdftops.exe` helper program.
2. Now fonts will be embedded in `.pfa` (PostScript) format + you can extract them using a **text editor**.
3. You may need to convert the `.pfa` (ASCII) to a `.pfb` (binary) file using the `t1utils` and `pfa2pfb`.
4. In PDFs there are never `.pfm` or `.afm` files (font metric files) embedded (because PDF viewer have internal knowledge about these). Without these, font files are hardly usable in a visually pleasing way.

8.2 Method 2: Using `fontforge`

Another method is to use the Free font editor **FontForge**[2]:

1. Use the *"Open Font"* dialogbox used when opening files.
2. Then select *"Extract from PDF"* in the filter section of dialog.
3. Select the PDF file with the font to be extracted.

[1] http://www.foolabs.com/xpdf/download.html

[2] http://fontforge.sourceforge.net/

4. A *“Pick a font”* dialogbox opens – select here which font to open.

Check the FontForge manual. You may need to follow a few specific steps which are not necessarily straightforward in order to save the extracted font data as a file which is re-usable.

8.3 Method 3: Using `mupdf`

Next, **MuPDF**[3]. This application comes with a utility called `pdfextract` (on Windows: `pdfextract.exe`) which can extract fonts and images from PDFs. (In case you don’t know about MuPDF, which still is relatively unknown and new: *“MuPDF is a Free lightweight PDF viewer and toolkit written in portable C.”*, written by Artifex Software developers, the same company that gave us Ghostscript.)

Note: pdfextract.exe is a command-line program. To use it, do the following:

```
c:\>  pdfextract.exe  c:\path\to\filename.pdf        # (on Windows)
$>    pdfextract  /path/tofilename.pdf                # (on Linux, Unix, Mac OS X)
```

This command will dump all of the extractable files from the pdf file referenced into the current directory. Generally you will see a variety of files: images as well as fonts. These include PNG, TTF, CFF, CID, etc. The image names will be like *img-0412.png* if the PDF object number of the image was 412. The fontnames will be like *FGETYK+LinLibertineI-0966.ttf*, if the font’s PDF object number was 966.

CFF (*Compact Font Format*) files are a recognized format that can be converted to other formats via a variety of converters for use on different operating systems.

Again: be aware that most of these font files may have only a *subset* of characters and may not represent the complete typeface.

Update: (Jul 2013) Recent versions of `mupdf` have seen an internal reshuffling and renaming of their binaries, not just once, but several times. The main utility used to be a ‘swiss knife’-alike binary called `mubusy` (name inspired by busybox?), which more recently was renamed to `mutool`. These support the sub-commands `info`, `clean`, `extract`, `poster` and `show`. Unfortunatey, the official documentation for these tools isn’t up to date (yet). If you’re on a Mac using ‘MacPorts’: then the utility was renamed in order to avoid name clashes with other utilities using identical names, and you may need to use `mupdfextract`.

To achieve the (roughly) equivalent results with `mutool` as its previous tool `pdfextract` did, just run `mubusy extract ....`*

So to extract fonts and images, you may need to run one of the following commandlines.

On Windows:

[3] http://mupdf.com/

```
c:\>  mutool.exe extract filename.pdf
```

On Linux, Unix, Mac OS X:

```
$>    mutool     extract filename.pdf
```

8.4 Method 4: Using gs (Ghostscript)

Finally, **Ghostscript**[4] can also extract fonts directly from PDFs. However, it needs the help of a special utility program named **`extractFonts.ps`**[5], written in PostScript language, which is available from the Ghostscript source code repository[6].

Now use it, you need to run both, this file `extractFonts.ps` and your PDF file. Ghostscript will then use the instructions from the PostScript program to extract the fonts from the PDF. It looks like this on Windows (yes, Ghostscript understands the 'forward slash', /, as a path separator also on Windows!):

```
gswin32c.exe                  ^
  -q -dNODISPLAY              ^
   c:/path/to/extractFonts.ps ^
  -c "(c:/path/to/your/PDFFile.pdf) extractFonts quit"
```

or on Linux, Unix or Mac OS X:

```
gs                          \
  -q -dNODISPLAY            \
   /path/to/extractFonts.ps \
  -c "(/path/to/your/PDFFile.pdf) extractFonts quit"
```

I've tested the Ghostscript method a few years ago. At the time it did extract *.ttf (TrueType) just fine. I don't know if other font types will also be extracted at all, and if so, in a re-usable way. I don't know if the utility does block extracting of fonts which are marked as protected.

[4] http://www.ghostscript.com/releases/

[5] http://git.ghostscript.com/?p=ghostpdl.git;a=blob_plain;f=gs/toolbin/extractFonts.ps

[6] http://git.ghostscript.com/?p=ghostpdl.git;a=tree;f=gs

8.5 *Caveats:*

- ***In any case you need to follow the license that applies to the font. Some font licences do not allow free use and/or distribution. Pirating fonts is like pirating any software or other copyrighted material.***
- ***Most PDFs which are in the wild out there do not embed the full font anyway, but only subsets. Extracting a subset of a font is only useful in a very limited scope, if at all.***

Please do also read the following about Pros and (more) Cons regarding font extraction efforts:

- http://typophile.com/node/34377

9 How can I get Ghostscript to use embedded fonts in PDF?

Here is the command I use:

```
gs                         \
  -o output.pdf            \
  -dCompatibilityLevel=1.4 \
  -dPDFSETTINGS=/screen    \
  -sDEVICE=pdfwrite        \
  -sOutputFile=output.pdf  \
   input.pdf
```

I am using (trying anyway) to use Ghostscript to reduce my PDF file size. The command above looks like it works, it reduces file size greatly, but then several of the fields are garbled. As for as I can track it down, it's doing font substitution. IE, the same text = same garbled text.

The fonts are embedded in the PDF when it gets to me. Additionally, I have tried to add all the fonts to the Fontmap.

Any ideas, Ideally I would like it to use the embedded fonts without me having to update the gs system fonts/edit fontmap, etc. I'm using Ubuntu 9.10 and the Fonts embedded are windows fonts, Arial/TimesNewRoman.

9.1 Answer

Embedding fonts retrospectivly which were not embedded in the original PDF does increase the file size, not decrease it.

However, there may still be a chance to reduce the overall file size by reducing the resolution of embedded images… depends on your preferences and needs.

You can try with variations of the following commandline. It will embed all fonts (even the "Base 14" ones), but embed required glyphs only (a "subset" of the original font), and also compress the fonts:

```
gs                          \
   -o output.pdf            \
   -dCompatibilityLevel=1.4 \
   -dPDFSETTINGS=/screen    \
   -dCompressFonts=true     \
   -dSubsetFonts=true       \
   -sDEVICE=pdfwrite        \
   -c ".setpdfwrite <</NeverEmbed [ ]>> setdistillerparams" \
   -f input.pdf
```

You will have noticed that I did use the `-o output.pdf` convention instead of `-sOutputFile=output.pdf`. I also didn’t include `-dBATCH -dNOPAUSE` in my command. The reason is that both methods are equivalent, since `-o ...` silently also sets `-dBATCH -dNOPAUSE`:

‘Traditional’ Ghostscript option:

```
-sOutputfile=output.pdf -dBATCH -dNOPAUSE
```

‘Modern’ Ghostscript options

```
 -o output.pdf
```

However, the modern shortcut way of writing the command does not work for older Ghostscript versions.

If you look into reducing the file size of PDFs only and have now particularly compelling reason to set `-dPDFSETTINGS=/screen`, then the chapter “How can I convert a color PDF into grayscale?” may also be something to consider.

III Scanned Pages and PDF

10 How can I make the invisible OCR information on a scanned PDF page visible?

> **I have a PDF which is the result of scanned pages. It contains lots of numbers.**
>
> **In our organization's workflow, we usually scan incoming mail delivered by the postal service, archive them and then scrap the original papers.**
>
> **Having read some recent news about PDFs resulting from scans made with a certain brand of scanners mangling numbers badly, I want to check if this can happen with OCR too.**
>
> **My knowledge about OCR of scanned pages is rather limited. My only info about it is that it uses some hidden layer to store the text. How can I un-hide this hidden layer?**

10.1 Answer

No, OCR information about scanned pages is not stored in a hidden **layer**. Layers in a PDF are quite a different concept.

But OCR-ed text nevertheless is 'hidden' – but hidden alongside the same layer as the rest of the page content.

I suggest you read the chapter of this book named "How can I use invisible fonts in a PDF?" first. It gives you a short theoretical background of "invisible text" regarding PDF.

The OCR text in your PDF uses *Text Rendering Mode 3* ('Neither fill nor stroke glyph shapes'). In order to make this text visible, you have to change this text rendering mode to one of the other modes:

- `0 Tr` (fill text)
- `1 Tr` (stroke text)
- `2 Tr` (fill, then stroke text)
- `4 Tr` (fill text and add to path for clipping)
- `5 Tr` (stroke text and add to path for clipping)

My favorite mode for this job would be `1 Tr`. It will just draw the outline shape of the glyphs without filling them. I recommend to do this using a very thin red line. This way you will be able to see the exact positioning of the text relative to the scanned image when you zoom in to the page.

Unfortunately I do not know of any commandline tool that can achieve this. You'll have to dive into the PDF source code and manipulate it with a text editor.

Fortunately this is much more easy than it sounds at first. We will use three steps for this:

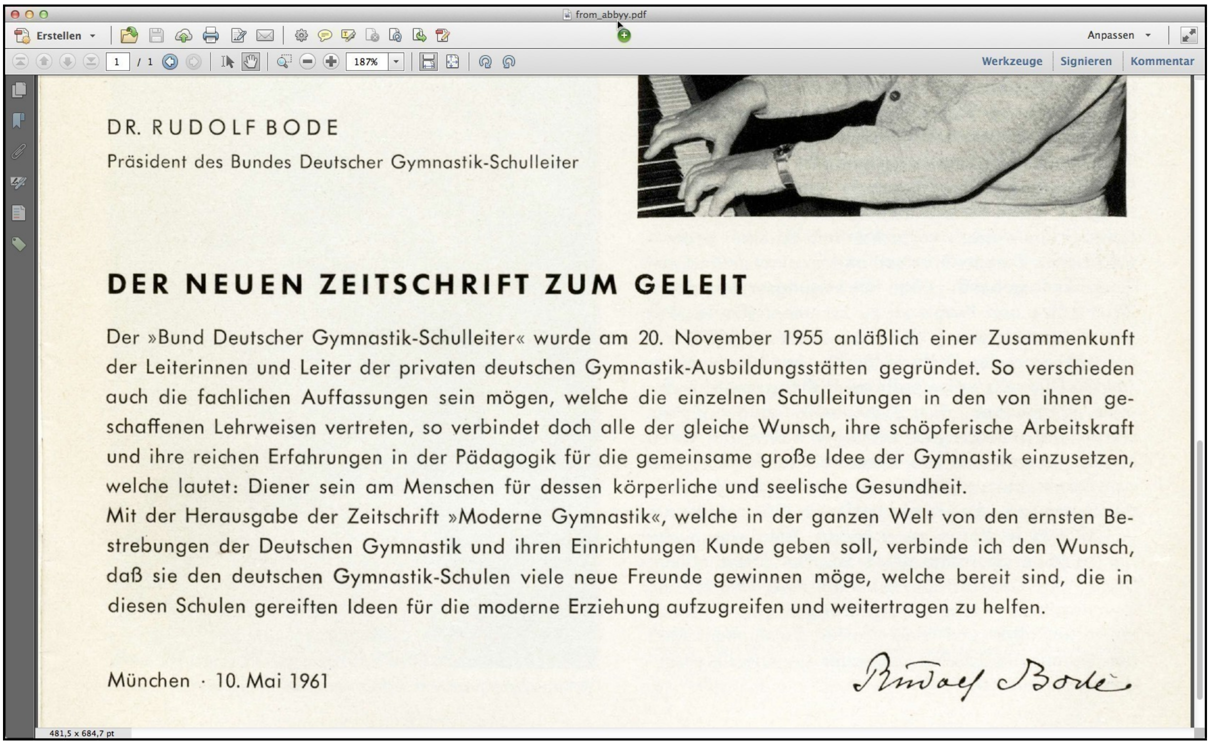
1. Expand the original PDF source code of the OCR/scanned PDF using **`qpdf`**[1].

[1] http://qpdf.sf.net/

2. Open the expanded PDF source code in a simple text editor and manipulate it.
3. 'Repair' the PDF source code (which has become 'corrupted' through our editing) and copress it again.

Step 1: Expand the original PDF

Looking at the scanned PDF page may show a view like the one in the following image.



DR. RUDOLF BODE

Präsident des Bundes Deutscher Gymnastik-Schulleiter

DER NEUEN ZEITSCHRIFT ZUM GELEIT

Der »Bund Deutscher Gymnastik-Schulleiter« wurde am 20. November 1955 anläßlich einer Zusammenkunft der Leiterinnen und Leiter der privaten deutschen Gymnastik-Ausbildungsstätten gegründet. So verschieden auch die fachlichen Auffassungen sein mögen, welche die einzelnen Schulleitungen in den von ihnen geschaffenen Lehrweisen vertreten, so verbindet doch alle der gleiche Wunsch, ihre schöpferische Arbeitskraft und ihre reichen Erfahrungen in der Pädagogik für die gemeinsame große Idee der Gymnastik einzusetzen, welche lautet: Diener sein am Menschen für dessen körperliche und seelische Gesundheit.
Mit der Herausgabe der Zeitschrift »Moderne Gymnastik«, welche in der ganzen Welt von den ernsten Bestrebungen der Deutschen Gymnastik und ihren Einrichtungen Kunde geben soll, verbinde ich den Wunsch, daß sie den deutschen Gymnastik-Schulen viele neue Freunde gewinnen möge, welche bereit sind, die in diesen Schulen gereiften Ideen für die moderne Erziehung aufzugreifen und weitertragen zu helfen.

München · 10. Mai 1961

Rudolf Bode

Screenshot showing the original scanned/OCR-ed PDF page opened in Acrobat.

If you've read other chapters of this book already, you may be familiar with `qpdf`. It can expand PDF source code and transform it into a mode that makes it more easy to process for human brains (if these brains have acquired some PDF knowhow beforehand, or if they are guided with the help of a book like this one). Here is the command to use:

```
qpdf  --qdf  --object-streams=disable  original-scan.pdf  qdf---original-scan.pdf
```

This created a new PDF file named `qdf---original-scan.pdf` which can easily be opened and manipulated by a text editor.

Note, in case your original PDF had binary data sections (such as images, fonts or color profiles), these will not be expanded and will still be contained in binary form in your expanded PDF. It is only the other components which were expanded. So your text editor should be able to not get a hangover from these binary parts and save your edited version without damaging these.

Step 2: Open the expanded PDF with a text editor

Now open the new PDF file in your favorite text editor. Search for all spots where you find the text string `3 Tr`. It could look like this:

```
[....]
/F16 7.500 Tf
3 Tr
1.180 Tc
[....]
```

Modify these text strings and replace them by the following: `1 0 0 RG 0.1 w 1 Tr`. The resulting PDF code could then look like this:

```
[....]
/F16 7.500 Tf
1 0 0 RG 0.1 w 1 Tr
1.180 Tc
[....]
```

This modification will have the following effects:

- **`1 Tr`** : this switches the text rendering mode to *'Stroke text'*.
- **`0.1 w`** : this sets the stroking line for the text rendering mode to a very thin one, *0.1 points* only.
- **`RG`** : this sets the RGB color mode for stroking operations.
- **`1 0 0 RG`** : this sets the color to 'red' for RGB colors.

Now save this modified PDF under a new name like `qdf---edited-scan.pdf`.

Step 3: 'Repair' the modified PDF and compress it again

Our editing manipulations will very likely have 'corrupted' the PDF. Because we inserted some 15 additional characters (*1 0 0 RG 0.1 w *), the PDF's cross reference table (which holds a list of all object addresses based as byte offsets from the files start) will no longer be correct. You can use `qpdf` to check for this problem:

```
qpdf --check  qdf---edited-scan.pdf
```

The output will be similar to this:

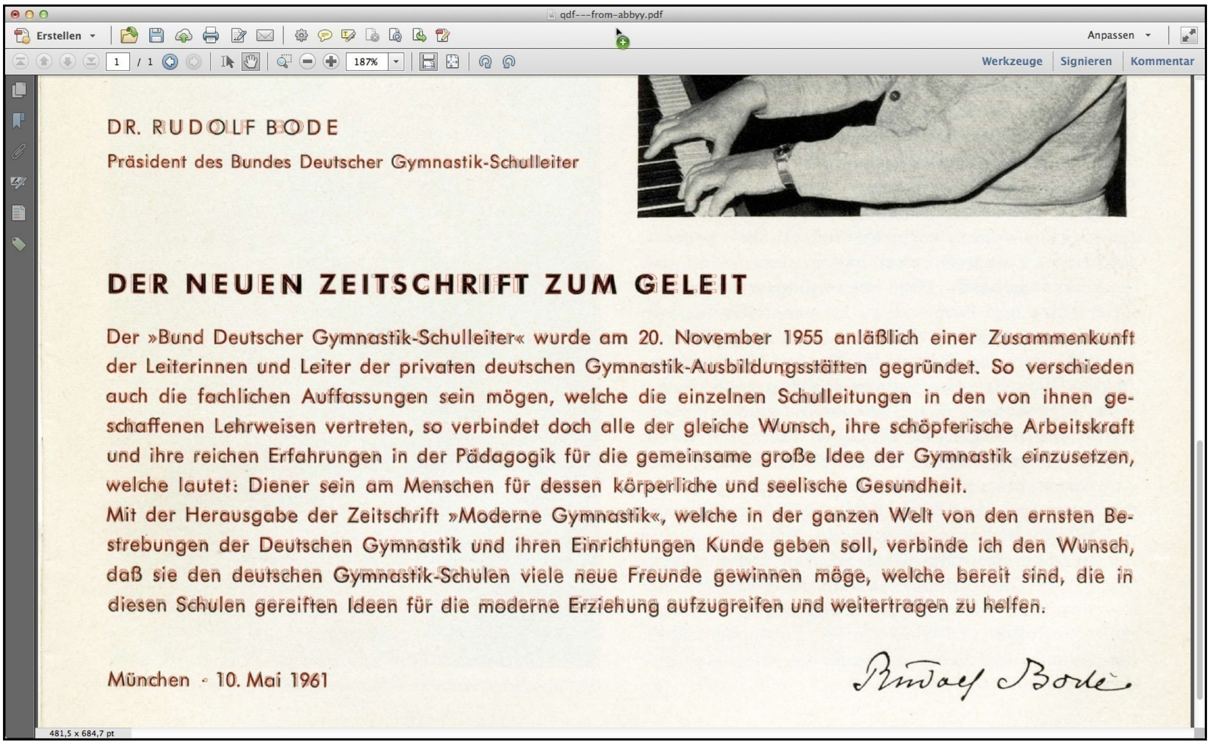
```
WARNING: qdf---edited-scan.pdf: file is damaged
WARNING: qdf---edited-scan.pdf (file position 717011): xref not found
WARNING: qdf---edited-scan.pdf: Attempting to reconstruct cross-reference table
checking qdf---edited-scan.pdf
PDF Version: 1.3
File is not encrypted
File is not linearized
```

Fortunately, many PDF viewers will not have major problems with this – they'll automatically (and often silently) calculate a new `xref` section for the PDF and use that instead of the one embedded in the file. You can try to open the file as is with your PDF viewer and see if it does or does not cause a problem.

But to play it save and make sure that each and every viewer will open the manipulated PDF without choking, we will use `qpdf` again in order to fix this problem:

```
qpdf  qdf---edited-scan.pdf  ocr-made-visible-in-scan.pdf
```

If you look at the resulting file, `ocr-made-visible-in-scan.pdf`, you should see something like this now:



DR. RUDOLF BODE

Präsident des Bundes Deutscher Gymnastik-Schulleiter

DER NEUEN ZEITSCHRIFT ZUM GELEIT

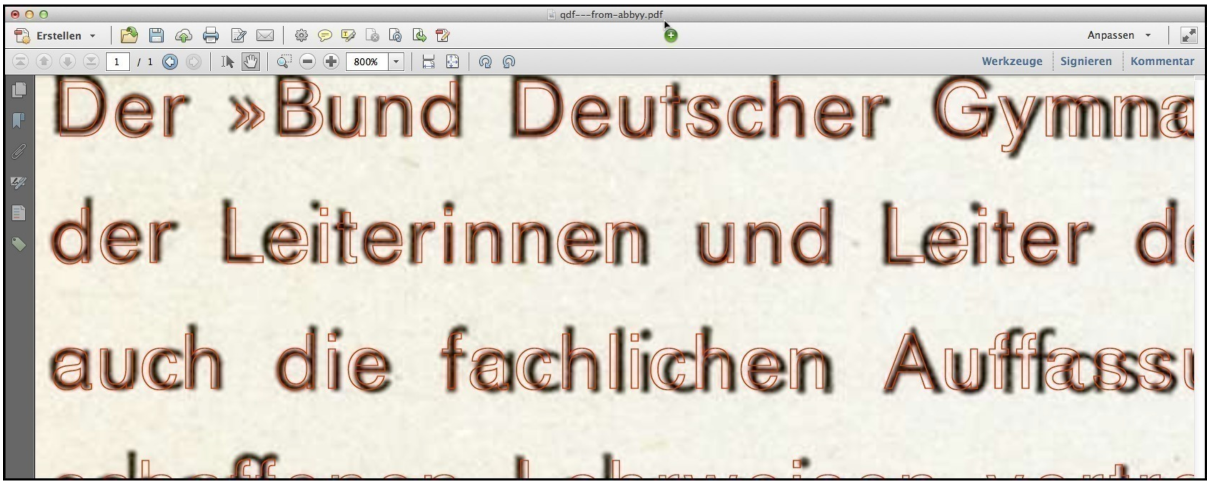
Der »Bund Deutscher Gymnastik-Schulleiter« wurde am 20. November 1955 anläßlich einer Zusammenkunft der Leiterinnen und Leiter der privaten deutschen Gymnastik-Ausbildungsstätten gegründet. So verschieden auch die fachlichen Auffassungen sein mögen, welche die einzelnen Schulleitungen in den von ihnen geschaffenen Lehrweisen vertreten, so verbindet doch alle der gleiche Wunsch, ihre schöpferische Arbeitskraft und ihre reichen Erfahrungen in der Pädagogik für die gemeinsame große Idee der Gymnastik einzusetzen, welche lautet: Diener sein am Menschen für dessen körperliche und seelische Gesundheit.

Mit der Herausgabe der Zeitschrift »Moderne Gymnastik«, welche in der ganzen Welt von den ernsten Bestrebungen der Deutschen Gymnastik und ihren Einrichtungen Kunde geben soll, verbinde ich den Wunsch, daß sie den deutschen Gymnastik-Schulen viele neue Freunde gewinnen möge, welche bereit sind, die in diesen Schulen gereiften Ideen für die moderne Erziehung aufzugreifen und weitertragen zu helfen.

München · 10. Mai 1961

Rudolf Bode

Screenshot showing the manipulated scanned/OCR-ed PDF page opened in Acrobat. The hidden OCR text is now made visible as thin red outlines. Zooming in to the image will reveal more details.



Der »Bund Deutscher Gymna

der Leiterinnen und Leiter de

auch die fachlichen Auffassu

Zooming into the manipulated scanned/OCR-ed PDF page at 800% in Acrobat.

Nice, isn’t it? You’ve just earned your *yellow belt* in PDF-KungFoo mastership. ;-)

IV Colors

11 How can I convert a color PDF into grayscale?

> **I have a bunch of PDF documents with lots of color images inside. I want to get them printed and want to make sure that no color is used. So I thought converting them to all-grayscale PDFs would be a good idea. Also I want to offer these documents for download – so conversion to grayscale seems to be a way to reduce the filesize. Is this correct?**
>
> **How can I achieve this with Ghostscript?**

11.1 Answer

Yes, it is correct that grayscale instead of color images inside PDFs in general do reduce the filesize. I will show this with a small example.

A command to do that is the following:

```
gs                               \
  -o gray.pdf                    \
  -sDEVICE=pdfwrite              \
  -sColorConversionStrategy=Gray \
  -sProcessColorModel=DeviceGray \
   color.pdf
```

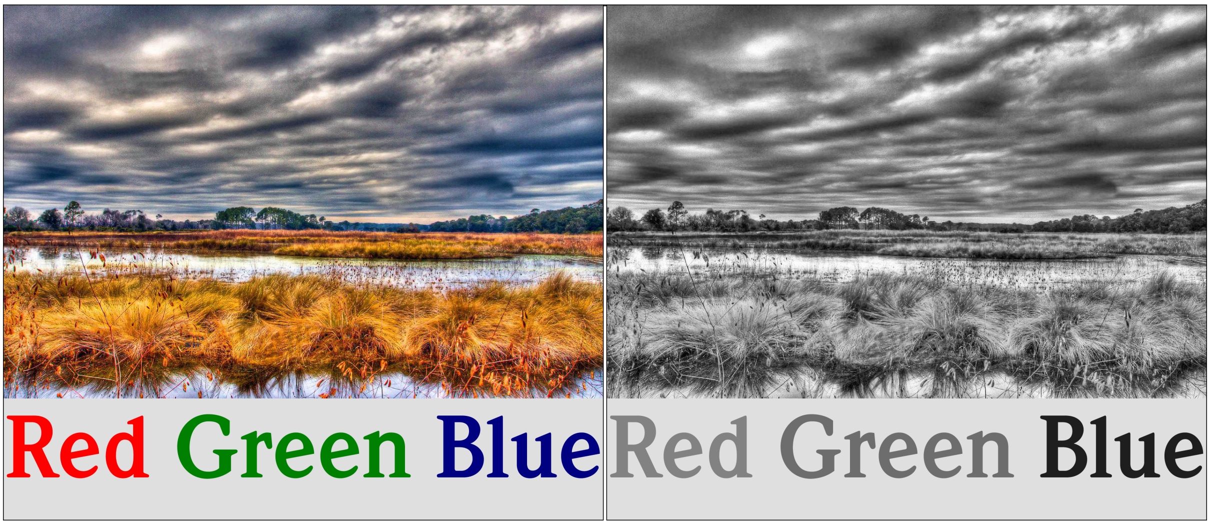
As long as the image's resolution remains unchanged, a color-to-gray conversion should significantly reduce the size:

- RGB images use 3 color channels (red, green, blue)
- CMYK images use 4 color channels (cyan, magenta, yellow)
- Gray images use only one color channel

Assuming the same level of *color depth* for each image type, this means that ratios for the raw amount of uncompressed image data withequally-dimensioned images *1:3:4* for *gray:rgb:cmyk* images.

Of course, in the real life images are compressed. Depending on their actual contents, different compression algorithms will change this ratios – but for a first approximation they are an important consideration to make.

The picture below[1] shows the original input file, *color.pdf*, vs. the resulting grayscale output, *gray.pdf*.



Left: **original color PDF – *Right:* grayscale PDF converted with Ghostcript. (Color image used in the PDF is by Craig ONeal ("minds-eye"), licensed under Creative Commons 'BY-SA 2.0'.)**

Comparing the file sizes of these two files shows this:

- **`color.pdf`** : 2,6 MByte
- **`gray.pdf`** : 172 kBbyte

So in this specific case the conversion reduced the file size to roughly 6% of the original.

Note, not every single color-to-gray conversion will show the same amount of file size reduction.** **The reduction ratio very much depends on the amount of color images used in the original PDF vs. the amount of text or other elements.

To further analyse what has happened to the image during conversion, we can use `pdfimages` and `pdfinfo` like this:

```
pdfimages -list color.pdf
  page   num  type   width height color comp bpc  enc interp  object ID
  ---------------------------------------------------------------------
     1     0 image    1280   825  rgb     3   8  image  no         4  0


pdfimages -list gray.pdf
  page   num  type   width height color comp bpc  enc interp  object ID
  ---------------------------------------------------------------------
     1     0 image    1280   825  gray    1   8  jpeg   no        10  0


```

[1] For details about the original color image see chapter "Acknowlegements"

```
pdfinfo color.pdf | grep "Page size:"
  Page size:      595 x 510 pts
```

The image in the PDF uses the RGB color space. Since the width of the page is 595 PostScript points (where 72 pt == 1 inch), and the width of the image (borderless on the page) is 1280 pixels, the resolution of the image as embedded in the PDF page can easily be calculated. In the current case that resolution for both, *color.pdf* and *gray.pdf* is 152 ppi (*pixels per inch*).

Assuming the original color image had been of a higher resolution. With 300 ppi (4 times the number of pixels than at 150 ppi) the PDF size could easily have exceeded 10 MByte. With 600 ppi (16 times the number of pixels than at 150 ppi) it could have exceeded 40 MByte.

Converting these high-resolution color images to grayscale would also significantly reduce the file size. But when doing this conversion, you could at the same time downsample all high resolution images to, say, 150 ppi. Here is how you'd achieve this:

```
gs                                   \
 -o 150ppi-gray.pdf                  \
 -sDEVICE=pdfwrite                   \
 -sColorConversionStrategy=Gray      \
 -sProcessColorModel=DeviceGray      \
 -dDownsampleMonoImages=true         \
 -dMonoImageResolution=150           \
 -dMonoImageDownsampleType=/Bicubic  \
 -dMonoImageFilter=/CCITTFaxEncode   \
 -dMonoImageDownsampleThreshold=1.0  \
 -dDownsampleGrayImages=true         \
 -dGrayImageResolution=150           \
 -dGrayImageDownsampleType=/Bicubic  \
 -dGrayImageFilter=/DCTEncode        \
 -dGrayImageDownsampleThreshold=1.0  \
 -dDownsampleColorImages=true        \
 -dColorImageResolution=150          \
 -dColorImageDownsampleType=/Bicubic \
 -dColorImageFilter=/DCTEncode       \
 -dColorImageDownsampleThreshold=1.0 \
  600ppi-color.pdf
```

As you can see from the various command line parameters, you could differentiate between color, grayscale as well as mono images and give different parameters for each type. Above I used the same ones for each.

Three parameters which deserve special mention here are the *{Mono,Gray,Color}DownsampleThreshold* ones. Their default value is `1.5`. This means that downsampling will only happen, if the original's image resolution is 1.5 times as high or higher than the target image's resolution. If you have an image of 250 ppi embedded in a PDF page its resolution will remain unchanged for a 150 ppi target. Images will only be downsampled if they are at 225 ppi or above. Setting the *{Mono,Gray,Color}DownsampleThreshold*s to

`1.0` enforces the downsampling of each and every image that has a higher resolution than the target.

V Using pdfmarks

12 How can I use `pdfmark` to insert bookmarks into PDF? (CONTENT STILL MISSING)

VI Text extraction

13 How can I extract text from PDF? (CONTENT STILL MISSING)

VII Miscellaneous

14 How to recognize PDF format?

> Given a stream of bytes, how can I tell if this stream contains a PDF document or something else?
>
> I am using .NET and C# but it does not matter.

14.1 Answer

It all depends of how well/reliable you want the detection working.

Here my selection of most important bits+pieces from the 756 page long official definition, straight from the horse's mouth (PDF 32000:1-2008[1]):

> *A basic conforming PDF file shall be constructed of following four elements (see Figure 2):*
>
> - *A one-line header identifying the version of the PDF specification to which the file conforms*
> - *A body containing the objects that make up the document contained in the file*
> - *A cross-reference table containing information about the indirect objects in the file*
> - *A trailer giving the location of the cross-reference table and of certain special objects within the body of the file*
>   *[....]*
>
> *The first line of a PDF file shall be a header consisting of the 5 characters %PDF– followed by a version number of the form 1.N, where N is a digit between 0 and 7.* A conforming reader shall accept files with any of the following headers:*
> *%PDF-1.0*
> *%PDF-1.1*
> *%PDF-1.2*
> *%PDF-1.3*
> *%PDF-1.4*
> *%PDF-1.5*
> *%PDF-1.6*
> *%PDF-1.7*
> *[…]*
>
> *If a PDF file contains binary data, as most do (see 7.2, "Lexical Conventions"), the header line shall be immediately followed by a comment line containing at least four binary characters—that is, characters whose codes are 128 or greater. This ensures proper behaviour of file transfer applications that inspect data near the beginning of a file to determine whether to treat the file's contents as text or as binary.*

[1] http://www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf#page46

> ***Trailer***
> *[….] The last line of the file shall contain only the end-of-file marker, %%EOF. The two preceding lines shall contain, one per line and in order, the keyword startxref and the byte offset in the decoded stream from the beginning of the file to the beginning of the xref keyword in the last cross-reference section.*

Summary

Two of the most important things to remember:

(a) The first ‘header line’

```
%PDF-1.X
```

[where X in 0..7] must appear on a line of its own be followed by a newline. This line must appear within the first 4096 Bytes, not necessarily on the very first line. The preceding lines may contain non-PDF content, but *printer job languange commands* (PJL) or comments.

(b) The very next line must be four binary bytes if the PDF contains binary data.

Just parsing for ‘%PDF-1.’, relying on this and not looking for anything else, has bitten a lot of people already….

VIII Some Topics in Depth

15 Can I query the default settings Ghostscript uses for an output device (such as ‘pdfwrite’ or ‘tiffg4’)?

> **In this answer to ‘Ghostscript command line parameters to convert EPS to PDF’[a], it is stated that the default resolution for the `pdfwrite` device of Ghostscript is `720x720`, which I initially found unbelievable!**
>
> **Is there a way to list the default options of a Ghostscript device?**
>
> [a] http://stackoverflow.com/a/3461186/277826

15.1 Answer

Since Ghostscript is a full-blown PostScript interpreter, you can also send PostScript snippets to it which do not cause the drawing of page elements, but which query it for its internal state.

If you want to know what the default settings of the *Display* are, when you ask it via `gs some.pdf` to just display a PDF on screen, you could try this:

Sample command line (Linux, Unix, Mac OS X):

```
gs                                                   \
  -c "currentpagedevice {exch ==only ( ) print == } forall"
```

On Windows this becomes:

```
gswin32c.exe                                           ^
   -c "currentpagedevice {exch ==only ( ) print == } forall"
```

The result is a list of `/SomeName somevalue` pairs which describe the settings used for rendering pages to the current screen.

This is so because usually the display is the default device for Ghostscript to send its output to. Now you may notice that you’ll see an empty Ghostscript window pop up, which you’ll have to close.... Ah, how about adding some options to avoid the popup window?

```
gs                                                          \
   -o /dev/null                                             \
   -dNODISPLAY                                              \
   -c "currentpagedevice {exch ==only ( ) print == } forall"
```

Or, on Windows:

```
gswin32c.exe                                                ^
   -o nul                                                   ^
   -dNODISPLAY                                              ^
   -c "currentpagedevice {exch ==only ( ) print == } forall"
```

But this will change the query return values, because you (unintentionally) changed the output device settings:

```
gs -c "currentpagedevice {exch ==only ( ) print == } forall" | grep Resolution
```

Result:

```
HWResolution [86.5426483 86.5426483]
/.MarginsHWResolution [1152.0 1152.0]
```

Compare this to

```
gs                                                         \
  -o /dev/null                                             \
  -dNODISPLAY                                              \
  -c "currentpagedevice {exch ==only ( ) print == } forall" \
| grep Resolution
```

Result:

```
/HWResolution [72.0 72.0]
/.MarginsHWResolution [72.0 72.0]
```

So, please avoid this trap. I successfully fell into it a few years ago, and didn't even notice it for quite a long time…

Now assuming you want to query for the default settings of the PDF writing device, run this one:

```
gs                                                          \
  -o /dev/null                                              \
  -sDEVICE=pdfwrite                                         \
  -c "currentpagedevice {exch ==only ( ) print == } forall" \
| tee ghostscript-pdfwrite-default-pagedevice-settings.txt
```

You'll now have all settings for the pdfwrite device in a *.txt file. And you may repeat that with some other interesting Ghostscript devices and then compare them for all their detailled differences:

```
for _dev in                                                 \
  pswrite ps2write pdfwrite                                 \
  tiffg3 tiffg4 tiff12nc tiff24nc tiff32nc tiff48nc tiffsep \
  jpeg jpeggray jpegcmyk                                    \
  png16 png16m png256 png48 pngalpha pnggray pngmono;       \
do                                                            \
  gs                                                          \
    -o /dev/null                                              \
    -sDEVICE=${_dev}                                          \
    -c "currentpagedevice {exch ==only ( ) print == } forall" \
   | sort                                                     \
   | tee ghostscript-${_dev}-default-pagedevice-settings.txt; \
done
```

It's rather interesting to compare the settings for, say, the pswrite and ps2write devices like this (and also discover parameters which are available for the one, but not the other device). One method that gives you a quick overview is to apply the commandline tool sdiff to the two text files:

```
sdiff -sbB ghostscript-ps{2,}write-default-pagedevice-settings.txt
```

On my Mac OS X system, this yields the following result (left column: `ps2write`, right column: `pswrite` output):

```
/AllowIncrementalCFF false                    <
/AllowPSRepeatFunctions true                  <
/AutoFilterColorImages true                   | /AutoFilterColorImages false
/AutoFilterGrayImages true                    | /AutoFilterGrayImages false
/AutoPositionEPSFiles true                    <
/CalCMYKProfile (None)                        | /CalCMYKProfile ()
/CalGrayProfile (None)                        | /CalGrayProfile ()
/CalRGBProfile (None)                         | /CalRGBProfile ()
/CannotEmbedFontPolicy /Error                 | /CannotEmbedFontPolicy /Warning
/CenterPages false                            <
/ColorImageDownsampleType /Bicubic            | /ColorImageDownsampleType /Subsample
/ColorImageResolution 600                     | /ColorImageResolution 150
/CompatibilityLevel 1.2                       <
/CompressEntireFile false                     <
/CompressFonts true                           <
/CoreDistVersion 5000                         <
/CreateJobTicket false                        <
/DSCEncodingToUnicode []                      <
/DetectDuplicateImages true                   <
/DoNumCopies false                            <
/DocumentTimeSeq 0                            <
/DocumentUUID ()                              <
/DownsampleColorImages true                   | /DownsampleColorImages false
/DownsampleGrayImages true                    | /DownsampleGrayImages false
/DownsampleMonoImages true                    | /DownsampleMonoImages false
/EmitDSCWarnings false                        <
/EncryptionR 0                                <
/FirstObjectNumber 1                          <
/FitPages false                               <
/GrayImageDownsampleType /Bicubic             | /GrayImageDownsampleType /Subsample
/GrayImageResolution 600                      | /GrayImageResolution 150
/HaveCIDSystem false                          <
/HaveTransparency true                        <
/HaveTrueTypes true                           <
/HighLevelDevice true                         <
/ImageMemory 524288                           | /ImageMemory 500000
/InstanceUUID ()                              <
/IsDistiller true                             <
/KeyLength 0                                  <
                                              > /LanguageLevel 2.0
/MaxClipPathSize 12000                        <
/MaxInlineImageSize -1                        <
/MaxShadingBitmapSize 256000                  <
/MaxViewerMemorySize -1                       <
/MonoImageDownsampleThreshold 1.5             | /MonoImageDownsampleThreshold 2.0
/MonoImageDownsampleType /Bicubic             | /MonoImageDownsampleType /Subsample
/MonoImageResolution 1200                     | /MonoImageResolution 300
/Name (ps2write)                              | /Name (pswrite)
/NoEncrypt ()                                 <
```

```
/OffOptimizations 0                           <
/Optimize true                                <
/OutputDevice /ps2write                       | /OutputDevice /pswrite
/OwnerPassword ()                             <
/PDFA false                                   <
/PDFACompatibilityPolicy 0                    <
/PDFEndPage -1                                <
/PDFStartPage 1                               <
/PDFX false                                   <
/PDFXBleedBoxToTrimBoxOffset [0.0 0.0 0.0 0.0]<
/PDFXSetBleedBoxToMediaBox true               <
/PDFXTrimBoxToMediaBoxOffset [0.0 0.0 0.0 0.0]<
/ParseDSCComments true                        <
/ParseDSCCommentsForDocInfo true              <
/PatternImagemask false                       <
/Permissions -4                               <
/PreserveCopyPage true                        <
/PreserveDeviceN true                         <
/PreserveEPSInfo true                         <
/PreserveHalftoneInfo true                    | /PreserveHalftoneInfo false
/PreserveOPIComments true                     | /PreserveOPIComments false
/PreserveOverprintSettings true               | /PreserveOverprintSettings false
/PreserveSMask false                          <
/PreserveSeparation true                      <
/PreserveTrMode false                         <
/PrintStatistics false                        <
/ProduceDSC true                              <
/ReAssignCharacters true                      <
/ReEncodeCharacters true                      <
/RotatePages false                            <
/SetPageSize false                            <
/UCRandBGInfo /Preserve                       | /UCRandBGInfo /Remove
/UsePrologue false                            <
/UserPassword ()                              <
/WantsToUnicode false                         <
/sRGBProfile (None)                           | /sRGBProfile ()
```

How to interpret this output?

- Lines with param key entries on both halves indicate: there is the same key available for both output devices, but each one uses a different default value.
- Lines with an entry for one half only indicate: this parameter key is unknown to the other output device.

One example is the `/GrayImageResolution` key: `ps2write` has this set to `600` by default whereas `pswrite` uses `150`. Another example is `/LanguageLevel`: `ps2write` has set it to `2.0`, while `pswrite` doesn’t know about this setting. (It produces PostScript language level 1 only). The third example is `/CompressFonts`: `ps2write` will compress fonts by default. (You could override this, by specifying a different behavior on the commandline and force uncompressed fonts in the PostScript output.) `pswrite` does not support this setting at all.

15.2 Update

As you may imagine this is also a great way to compaare different Ghostscript versions, and track how default settings may have changed for different devices in recent releases. This is especially interesting if you want to find out about all the newly implemented color profile and ICC support which is now present in Ghostscript.

Also, to avoid the return of just `-dict-` for certain key values, use the === instead of == macro. === acts like == but also prints the *content* of dictionaries.

So here is the example output for the `pdfwrite` device. Remember, Ghostscript’s `pdfwrite` device is meant to provide mostly the same functionality as *Adobe Acrobat Distiller* (with the additional feature that it does not only accept PostScript as input, but also PDFs, so you can sort of *redistill* existing PDF files in order to repair, improve or otherwise manipulate them). Therefore, Ghostscript’s `pdfdevice` honors most of the `setdistillerparams` operator which the original Distiller also supports. This is the command to use:

```
gs                                                           \
  -o /dev/null                                               \
  -sDEVICE=pdfwrite                                          \
  -c "currentpagedevice {exch ==only ( ) print === } forall" \
| sort
```

On my system, this produces the following output. I include it here in full, because this book will also serve as my personal lookup reference for certain info – in this is one I do need quite frequently:

```
/%MediaDestination 0
/%MediaSource 0
/.AlwaysEmbed []
/.HWMargins [0.0 0.0 0.0 0.0]
/.IgnoreNumCopies false
/.LockSafetyParams false
/.MarginsHWResolution [720.0 720.0]
/.MediaSize [612.0 792.0]
/.NeverEmbed [                                                                      \
               /Courier /Courier-Bold /Courier-Oblique /Courier-BoldOblique         \
               /Helvetica /Helvetica-Bold /Helvetica-Oblique /Helvetica-BoldOblique \
               /Times-Roman                                                         \
               /Times-Bold /Times-Italic /Times-BoldItalic                          \
               /Symbol /ZapfDingbats                                                \
             ]
/ASCII85EncodePages false
/AllowIncrementalCFF false
/AllowPSRepeatFunctions false
/AlwaysEmbed []
```

```
/AntiAliasColorImages false                                                        [*]
/AntiAliasGrayImages false                                                         [*]
/AntiAliasMonoImages false                                                         [*]
/AutoFilterColorImages true
/AutoFilterGrayImages true
/AutoPositionEPSFiles true
/AutoRotatePages /PageByPage
/BeginPage {--.callbeginpage--}
/Binding /Left                                                                     [*]
/BitsPerPixel 24
/BlueValues 256
/CalCMYKProfile (None)                                                             [*]
/CalGrayProfile (None)                                                             [*]
/CalRGBProfile (None)                                                              [*]
/CannotEmbedFontPolicy /Warning                                                    [*]
/CenterPages false
/ColorACSImageDict << /Blend 1 /VSamples [2 1 1 2] /QFactor 0.9 /HSamples [2 1 1 2] >>
/ColorConversionStrategy /LeaveColorUnchanged
/ColorImageDepth -1
/ColorImageDict    << /Blend 1 /VSamples [2 1 1 2] /QFactor 0.9 /HSamples [2 1 1 2] >>
/ColorImageDownsampleThreshold 1.5
/ColorImageDownsampleType /Subsample
/ColorImageFilter /DCTEncode
/ColorImageResolution 150
/ColorValues 16777216
/Colors 3
/CompatibilityLevel 1.4
/CompressEntireFile false
/CompressFonts true
/CompressPages true
/ConvertCMYKImagesToRGB false
/ConvertImagesToIndexed true
/CoreDistVersion 5000
/CreateJobTicket false                                                             [*]
/DSCEncodingToUnicode []
/DefaultRenderingIntent /Default
/DetectBlends true                                                                 [*]
/DetectDuplicateImages true
/DeviceGrayToK true
/DeviceLinkProfile ()
/DoNumCopies false
/DoThumbnails false                                                                [*]
/DocumentTimeSeq 0
/DocumentUUID ()
/DownsampleColorImages false
/DownsampleGrayImages false
/DownsampleMonoImages false
/EmbedAllFonts true
/EmitDSCWarnings false                                                             [*]
/EncodeColorImages true
/EncodeGrayImages true
/EncodeMonoImages true
/EncryptionR 0
/EndPage {--.callendpage--}                                                        [*]
```

```
/FirstObjectNumber 1
/FitPages false
/ForOPDFRead false
/GraphicICCProfile ()
/GraphicIntent 0
/GraphicsAlphaBits 1
/GrayACSImageDict << /Blend 1 /VSamples [2 1 1 2] /QFactor 0.9 /HSamples [2 1 1 2] >>
/GrayImageDepth -1
/GrayImageDict    << /Blend 1 /VSamples [2 1 1 2] /QFactor 0.9 /HSamples [2 1 1 2] >>
/GrayImageDownsampleThreshold 1.5
/GrayImageDownsampleType /Subsample
/GrayImageFilter /DCTEncode
/GrayImageResolution 150
/GrayValues 256
/GreenValues 256
/HWResolution [720.0 720.0]
/HWSize [6120 7920]
/HaveCIDSystem false
/HaveTransparency true
/HaveTrueTypes true
/HighLevelDevice true
/ImageICCProfile ()
/ImageIntent 0
/ImageMemory 524288                                                                [*]
/ImagingBBox null
/InputAttributes << \
                    0 << /PageSize [612.0 792.0] >> \
                    1 << /PageSize [ 792 1224] >> \
                    2 << /PageSize [ 612  792] >> \
                    3 << /PageSize [ 792 1224] >> \
                    4 << /PageSize [1224 1585] >> \
                    5 << /PageSize [1585 2448] >> \
                    6 << /PageSize [2448 3168] >> \
                    7 << /PageSize [2016 2880] >> \
                    8 << /PageSize [2384 3370] >> \
                    9 << /PageSize [1684 2384] >> \
                   10 << /PageSize [  73  105] >> \
                   11 << /PageSize [1191 1684] >> \
                   12 << /PageSize [ 842 1191] >> \
                   13 << /PageSize [ 595  842] >> \
                   14 << /PageSize [ 595  842] >> \
                   15 << /PageSize [ 420  595] >> \
                   16 << /PageSize [ 297  420] >> \
                   17 << /PageSize [ 210  297] >> \
                   18 << /PageSize [ 148  210] >> \
                   19 << /PageSize [ 105  148] >> \
                   20 << /PageSize [ 648  864] >> \
                   21 << /PageSize [ 864 1296] >> \
                   22 << /PageSize [1296 1728] >> \
                   23 << /PageSize [1728 2592] >> \
                   24 << /PageSize [2592 3456] >> \
                   25 << /PageSize [2835 4008] >> \
                   26 << /PageSize [2004 2835] >> \
                   27 << /PageSize [1417 2004] >> \
                   28 << /PageSize [1001 1417] >> \
                   29 << /PageSize [ 709 1001] >> \
                   30 << /PageSize [ 499  709] >> \
                   31 << /PageSize [ 354  499] >> \
                   32 << /PageSize [2599 3677] >> \
                   33 << /PageSize [1837 2599] >> \
                   34 << /PageSize [1298 1837] >> \
                   35 << /PageSize [ 918 1298] >> \
                   36 << /PageSize [ 649  918] >> \
                   37 << /PageSize [ 459  649] >> \
                   38 << /PageSize [ 323  459] >> \
                   39 << /PageSize [ 612  936] >> \
                   40 << /PageSize [ 612  936] >> \
                   41 << /PageSize [ 283  420] >> \
                   42 << /PageSize [ 396  612] >> \
                   43 << /PageSize [2835 4008] >> \
                   44 << /PageSize [2004 2835] >> \
                   45 << /PageSize [1417 2004] >> \
                   46 << /PageSize [1001 1417] >> \
                   47 << /PageSize [ 709 1001] >> \
                   48 << /PageSize [ 499  709] >> \
                   49 << /PageSize [ 354  499] >> \
                   50 << /PageSize [2920 4127] >> \
                   51 << /PageSize [2064 2920] >> \
                   52 << /PageSize [1460 2064] >> \
                   53 << /PageSize [1032 1460] >> \
                   54 << /PageSize [ 729 1032] >> \
                   55 << /PageSize [ 516  729] >> \
                   56 << /PageSize [ 363  516] >> \
                   57 << /PageSize [1224  792] >> \
                   58 << /PageSize [ 612 1008] >> \
                   59 << /PageSize [ 612  792] >> \
                   60 << /PageSize [ 612  792] >> \
                   61 << /PageSize [ 612  792] >> \
                   62 << /PageSize [ 595  792] >> \
                   63 << /PageSize [ 792 1224] >> \
                   64 << /PageSize [0 0 524287 524287] >> \
                  >>
/Install {--.callinstall--}
/InstanceUUID ()
/IsDistiller true
/KeyLength 0
/LZWEncodePages false
/Margins [0.0 0.0]
/MaxClipPathSize 12000
/MaxInlineImageSize 4000
/MaxPatternBitmap 0
/MaxSeparations 3
/MaxShadingBitmapSize 256000
/MaxSubsetPct 100
/MaxViewerMemorySize -1
/MonoImageDepth -1
/MonoImageDict << /K -1 >>
/MonoImageDownsampleThreshold 1.5
```

```
/MonoImageDownsampleType /Subsample
/MonoImageFilter /CCITTFaxEncode
/MonoImageResolution 300
/Name (pdfwrite)
/NeverEmbed [                                                                     \
                /Courier /Courier-Bold /Courier-Oblique /Courier-BoldOblique         \
                /Helvetica /Helvetica-Bold /Helvetica-Oblique /Helvetica-BoldOblique \
                /Times-Roman /Times-Bold /Times-Italic /Times-BoldItalic             \
                /Symbol /ZapfDingbats                                                \
               ]
/NoEncrypt ()
/NoT3CCITT false
/NumCopies null
/OPM 1
/OffOptimizations 0
/Optimize false                                                                    [*]
/OutputAttributes << 0 << >> >>
/OutputDevice /pdfwrite
/OutputFile (/dev/null)
/OutputICCProfile (default_rgb.icc)
/OwnerPassword ()
/PDFA 0
/PDFACompatibilityPolicy 0
/PDFEndPage -1
/PDFStartPage 1
/PDFX false
/PDFXBleedBoxToTrimBoxOffset [0.0 0.0 0.0 0.0]
/PDFXSetBleedBoxToMediaBox true
/PDFXTrimBoxToMediaBoxOffset [0.0 0.0 0.0 0.0]
/PageCount 0
/PageDeviceName null
/PageOffset [0 0]
/PageSize [612.0 792.0]
/ParseDSCComments true
/ParseDSCCommentsForDocInfo true
/PatternImagemask false
/Permissions -4
/Policies <<                                                                   \
                /PolicyReport                                                     \
                       {--dup-- /.LockSafetyParams --known--                      \
                       {/setpagedevice --.systemvar-- /invalidaccess signalerror} \
                       --if-- --pop--                                             \
                  }                                                               \
                /PageSize       0                                                 \
                /PolicyNotFound 1                                                 \
             >>
/PreserveCopyPage true                                                                [*]
/PreserveDeviceN true
/PreserveEPSInfo true                                                                 [*]
/PreserveHalftoneInfo false                                                           [*]
/PreserveOPIComments true                                                             [*]
/PreserveOverprintSettings true
/PreserveSMask true
/PreserveSeparation true
```

```
/PreserveTrMode true
/PrintStatistics false
/ProcessColorModel /DeviceRGB
/ProduceDSC true
/ProofProfile ()
/ReAssignCharacters true
/ReEncodeCharacters true
/RedValues 256
/RenderIntent 0
/RotatePages false
/SeparationColorNames []
/Separations false
/SetPageSize false
/SubsetFonts true
/TextAlphaBits 1
/TextICCProfile ()
/TextIntent 0
/TransferFunctionInfo /Preserve
/UCRandBGInfo /Preserve
/UseCIEColor false
/UseFastColor false
/UseFlateCompression true
/UsePrologue false                                                                  [*]
/UserPassword ()
/WantsToUnicode true
/sRGBProfile (None)                                                                 [*]
```

[*] Notes about the above lists:

According to the official Ghostscript documentation, the following settings (which are supported by Adobe Acrobat Distiller) currently on Ghostscript can be set and queried, *but setting them does have no effect:*

```
/AntiAliasColorImages
/AntiAliasGrayImages
/AntiAliasMonoImages
/AutoPositionEPSFiles
/Binding
/CalCMYKProfile
/CalGrayProfile
/CalRGBKProfile
/CannotEmbedFontPolicy
/ConvertImagesToIndexed
/CreateJobTicket
/DetectBlends
/DoThumbnails
/EmitDSCWarnings
/EndPage
/ImageMemory
```

```
/LockDistillerParams
/Optimize
/PreserveCopyPage
/PreserveEPSInfo
/PreserveHalftoneInfo
/PreserveOPIComments
/sRGBProfile
/StartPage
/UsePrologue
```

You may also want to read the chapter explaining the **purpose of Ghostscript dictionaries**.

Appendix

About the Author

Kurt has been coined *The Walking PDF Debugger* by several of his regular clients. They are right. Many of his problem solving skills in the last 10 years involved troubleshooting PDF processing systems in the Printing and Prepress Industry.

Kurt is a professional with more than 20 years of experience in the industry. After working for nearly 3 decades with the same employer (who in the process had 4 different names due to company mergers) he decided to freelance.

When working with customers, he prefers to use Free and Open Source Software whereever it works best. He is a commandline addict. As operating systems he prefers unix-oid types like Linux, Mac OS X and Solaris, but he is just as familiar with Windows and its `cmd.exe` too. These preferences were not predetermined from the start: up until 1998 he used Windows 95 exclusively. His first tentative adventures with Linux started in that very year. In 1999, still very much a newbie with Open Source, he became one the first users and beta testers of a new printing subsystem called CUPS (Common Unix Printing System). In the following years, CUPS very fast became the pre-dominant printing interface in the Linux and Unix world and has meanwhile been adopted and even acquired by Apple for Mac OS X.

Kurt's "career" as an author of technical documentation started when he helped users with technical questions about printing in different internet forums and contributed written documentation to various FOSS projects, such as Samba, Linuxprinting.org and KDE.

He currently is the all-time top scorer on StackOverflow.com when it comes to some of his favorite topics:

- His answers tagged as "`[pdf]`[1]" – Score, current[2]
- His answers tagged as "`[ghostscript]`[3]" – Score, current[4]
- His answers tagged as "`[imagemagick]`[5]" – Score, current[6]

Kurt is available for contract work:

- Kurt's LinkedIn Profile[7]
- Kurt's Xing Profile[8]

[1] http://stackoverflow.com/search?tab=votes&q=user%3a359307%20%5bpdf%5d

[2] http://stackoverflow.com/tags/pdf/topusers

[3] http://stackoverflow.com/search?tab=votes&q=user%3a359307%20%5bghostscript%5d

[4] http://stackoverflow.com/tags/ghostscript/topusers

[5] http://stackoverflow.com/search?tab=votes&q=user%3a359307%20%5bimagematick%5d

[6] http://stackoverflow.com/tags/imagemagick/topusers

[7] https://de.linkedin.com/pub/kurt-pfeifle/0/95/2a2

[8] https://www.xing.com/profile/Kurt_Pfeifle

Acknowledgements

This book would not exist without my customers. They were the ones who confronted me with problems, tasks and jobs that made me think of possible solutions and made me research stuff.

Also, it would not exist without the wonderful StackExchange[9] and Stackoverflow[10] platforms which draws together users and programmers, experts and interested people who ask questions, share solutions and discuss ideas.

Contributors

A number of people have sent me suggestions and corrections. I like to thank them all:

- Markus Wolf
- Adrian Pfeifle
- *… (your name could be here if you notified me of anything you want to be improved, added or corrected in future releases of this book!)*

Images

This eBook uses some images which were not created by myself. Here is a list:

- **Chapter "How can I convert a color PDF into grayscale?"**: The picture used in the demo PDF document for this chapter was made by Craig ONeal ("minds-eye") who hosts some of his work at Flickr[11]. This picture is licensed under *"Creative Commons Atrributions 2.0 Generic"* (attention!, not all of Craig's pictures use this license!). I downloaded it from Wikimedia.org[12] – See also here[13].

[9] http://www.stackexchange.com/

[10] http://www.stackoverflow.com/

[11] http://www.flickr.com/photos/craigoneal/

[12] http://upload.wikimedia.org/wikipedia/commons/thumb/e/e9/HDR_Savannah.jpg/1280px-HDR_Savannah.jpg

[13] http://commons.wikimedia.org/wiki/File:HDR_Savannah.jpg